Universiteit Utrecht

**Faculty of Bètasciences**

*Bsc. Micha van Grinsven*

# The automatization and adressal of processes such as corruption investigations and document analysis using Active Learning and Natural Language Processing.

Business Informatics Master Thesis

*Daily Supervisor*:

Ing. J. Snijder

Info Support

*First Supervisor*:

Dr. M.J.S. Brinkhuis

Department of Information and Computing Sciences

*Second Supervisor*:

Dr.ing. G.M. Krempl

Department of Information and Computing Sciences

January 11, 2023

**Abstract**

Active Learning is a relatively underused part of the machine learning domain in the real world for textual data that has shown better performance than Passive Learning. In this research, Active Learning is applied to two unbalanced datasets on the now-defunct energy company Enron and the Dutch oil company Shell. The Enron data is classified on the presence of information on logistics in documents whereas the Shell dataset is part of a current investigation into possible corruption by Follow The Money. This research attempts to aid this investigation by identifying documents belonging to a storyline in the dataset. Classification of documents is performed by looking only at the textual data in these datasets. To test the method the Enron dataset is used and after testing the method it is applied to the Shell dataset. It turns out that by using a combination of Active Learning and Natural Language Processing on the Shell data, an F1-score of 0.87 together with an accuracy of 91% can be achieved using only 5% of labeled data. Therefore, Active Learning can aid in the investigation of possible corruption. ASReview is used to facilitate this research. The setup presented in this research could be applied to almost any textual data classification problem.

# Contents

# 1 Introduction

Not having enough labeled data is a very common problem in supervised machine learning tasks. If a classifier needs to be properly trained, an abundance of labeled data is necessary. Usually, the more complex models, which can make sense of a wider range of data, need more labeled data than their simpler counterparts because they contain more hyperparameters that need to be tuned (Raviv, 2020). However, having labeled data is not common practice as labeling data is a costly and tedious task (Wallace, Trikalinos, Lau, Brodley, & Schmid, 2010). Because of this, much data that could be very interesting is left unused.

In the nineties, a different approach to supervised machine learning problems was suggested, namely Active Learning (Cohn, Atlas, & Ladner, 1994). Whereas passive supervised machine learning techniques randomly give training instances to the model, Active Learning tries to let the model choose from which data it learns in an attempt to lower the amount of labeled data necessary for generating good predictions. By doing this, supervised learning techniques can be trained with the same accuracy using 70-90% less labeled data (Das Bhattacharjee, Talukder, & Balantrapu, 2017; Lafferty, McCallum, & Pereira, 2001). This makes supervised machine learning a more applicable technique to fields that were previously left untouched because of lacking labels and the cost of actually getting these. Two of these fields, in which Active Learning has been suggested as a possible solution, are the field of corruption investigations and the field of document analysis for complex categories (Carcillo, Borgne, Caelen, & Bontempi, 2017; S. Kim et al., 2020). However, the use of Active Learning in these fields combined, to our knowledge, has never been researched in a natural language document-oriented way. By performing this research, interesting insights and opportunities could be found, as there are an enormous amount of documents that have never been properly scanned through. Because this for one takes too much time, and secondly requires a level of expertise only a very select amount of people have.

Scanning documents in an Active Learning-oriented approach has been researched before, in the area of structured literature reviews, and has provided very promising results (van de Schoot et al., 2020). A tool that was specifically created for this purpose is ASReview[1]. While this tool was originally designed for performing systematic reviews on academic papers, its design allows for use cases beyond this intended one. In this research, the tool is applied to the two fields we are interested in, namely corruption investigations and document analysis for complex categories. To apply the tool to these fields, two datasets will be used. The first dataset is the Shell papers dataset by Follow the Money (FTM, 2019), which is the subject of a current ongoing investigation into possible corruption between Royal Dutch Shell and the Dutch government. Note that Active Learning will not be used to identify possible cases of corruption directly but rather aid the investigation by helping identify the main storylines within the dataset. The second is the Enron dataset (Klimt & Yang, 2004), which is a very large email corpus. A part of this dataset contains labels for many criteria, such as whether an email contains information about logistics or not. Logistics in this case refers to emails containing information about meeting scheduling, technical support, or other logistical arrangements (*UC Berkeley Enron Email Analysis*, 2020). By applying the tool, it can be tested whether Active Learning could provide even more solutions than it has already given. This leads to the main research question of this paper: *To what extent can Active Learning in combination with Natural Language Processing be used to help identify possible cases of corruption and company logistics in business documents?*

To answer this main research question, first sub-questions need to be answered. First of all, it needs to be determined if using Active Learning to solve this problem is possible. Since the documents in question will contain or get labeled on properties that are sometimes explicit but mostly implicit and cannot be determined without expert knowledge, the properties that identify a document as belonging to such a specific class can be seen as complex. This leads to the first sub-question, *Is it possible to identify documents that adhere to complex criteria using Natural Language Processing in combination with Active Learning?* Second, the model used in the research should be as explainable as possible. To achieve this, features are divided into two groups, namely simple and complex. If a feature requires multiple computations to be obtained, it is a complex lexical feature. If it can be obtained with little to no computation, it is deemed a simple lexical feature. To determine which feature group explains the most, the second sub-question is: *What type of lexical features are more useful for training an Active Learning model to help identify storylines or information about company logistics in business documents?* Finally, an actively researched problem within the field of Active Learning is determining when the Active Learning should stop (Laws & Schätze, 2008; Vlachos, 2008). To contribute to solving this problem, the third and final sub-question is formulated as: *What are the aspects*

[1] https://asreview.nl/

*of the datasets that allow for efficient training of the model?* Since the labels that are searched for in this research are not evenly distributed among the available data, this could have a large influence on the training efficiency of the Active Learning model.

Note that the goal of this study is not to compare Active Learning to supervised learning approaches. Not only is this not possible in this case as there are not enough labels in the Shell dataset, but it has also already been attempted before (Das Bhattacharjee et al., 2017; Lafferty et al., 2001). Instead, the goal of this study is to find out if using just the information found in documents in an Active Learning way with Natural Language Processing is sufficient for pointing out documents that are part of a certain storyline to help investigations into possible corruption or contain other specific types of information. Note that in this study the Enron dataset will mostly be used for testing the method while the Shell dataset will be used for actually applying the method on a real-world dataset.

The rest of this study is laid out as follows. In section 2, related work is described to show where there is currently a gap in research. In section 3, a description of the data is given. In section 4, the method is discussed. In section 5, a test run of the method on the Enron dataset is discussed. In section 6 the results of the method on the Shell papers dataset are given. In section 7 the results and other parts of the research are discussed that were left out or were given less priority. Finally, in section 8 the conclusion is given together with suggestions for future research.

# 2 Related work

To prove a document is part of a specific storyline, or to prove that it contains other specific information, certain techniques are necessary. In this section, the background behind these techniques is given and state-of-the-art applications are discussed. First, research in the field of Active Learning is discussed. This is followed by research in the field of Natural Language Processing. More specifically, sentiment analysis, Named Entity Recognition, TF-IDF and bag of words are discussed, as features generated using these methods will be important for creating the model later on. After this, background is given on neural networks and transformers. Then, some examples of the combined application of Active Learning and Natural Language Processing are presented. The chapter is concluded with background on ASReview and a discussion of similar studies.

## 2.1 Active Learning

Active Learning (AL) is a form of machine learning that allows one to label data at a much faster rate than before while also obtaining supervised machine learning classifiers (Olsson, 2009). AL falls in between unsupervised and supervised learning, since it does require some labels to start training an algorithm, but abundantly less than any supervised learning technique such as random forests or neural networks. It features the benefits of both supervised and unsupervised learning techniques while having its own unique benefits as well. The key idea of AL is that if the model is allowed to choose from which data it learns, the same or even greater accuracy can be achieved using significantly fewer labels than supervised learning techniques (Settles, 2009). The term "Active Learning" was first coined in 1994 by Cohn et al. (1994) and research on the subject is still very prevalent today.

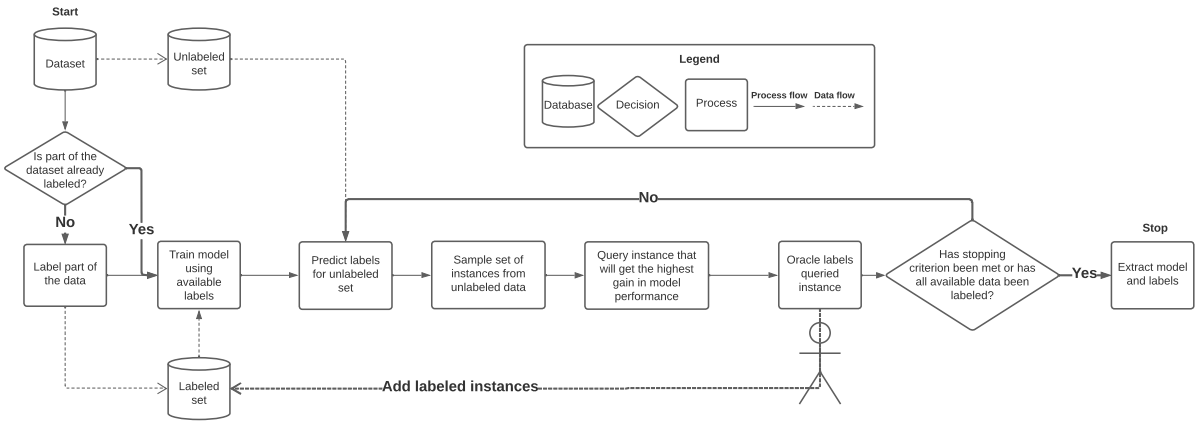



Figure 1: A process model of the AL process using pool-based sampling.

In Figure 1 an overview of the AL process is given. The process starts at the dataset. As labels are necessary to start the AL (Settles, 2012), a small part (10%) of the data is labeled if no labels are available. Without labels, AL cannot be performed. Once the dataset with its labeled slice is ready, a supervised machine learning algorithm, such as a neural network or another method, is trained on the labeled data. Once this is done, the obtained classifier is applied to the rest of the dataset that is unlabeled. However, when applying this newfound model to the data, the classifier also produces a confidence percentage. This number represents how certain the model is of the label that it has applied. The higher this number is, the more certain the model is of the label it has produced of the instance. Using this number and some kind of selection technique, which will be explained in section 2.1.2, the model asks an oracle to manually label the instances from which it expects to get the highest increase in accuracy. After the oracle, which is usually a human, has labeled the instance, the model is retrained and the process continues until some kind of stopping criterion is met or no unlabeled instances are left. In the end, the user ends up with a model that can predict new labels with good accuracy just like a supervised learning model would, while having used fewer labels than any supervised learning technique would require.

Going back to selecting the instances of which the model wants to learn, the process of retrieving the most valuable instances is called *querying*. There are many ways to perform this task, which can be divided into specific ways to query and scenarios in which querying can be applied. In the scenario process above, the pool-based selective scenario was described. However, more scenarios exist. The rationale behind this scenario and other scenarios together with different ways to perform querying is described in the next sections.

### 2.1.1 Query scenarios

There are three different scenarios in which the learner may be able to ask queries that have been considered in literature. These are membership query synthesis, stream-based selective sampling and pool-based AL.

With membership query synthesis, the model may ask labels for any unlabeled instance in the input space rather than those sampled from some underlying distribution. The only assumption is that the model has some definition of the input space available to it, for example, the feature dimensions and ranges (Settles, 2012). This works well in some instances but also falls short in some as outliers or other useless instances could be sampled as the model barely has no information about the distribution. In a study by Baum and Lang (1992), human oracles were asked to classify handwritten characters which were queried using membership query synthesis. However, since these queried images were not sampled using some kind of strategy, the images did not contain anything recognizable for the oracles. Therefore labeling was not properly possible. Similarly, Settles (2009) states that in this scenario producing labels for queries in Natural Language Processing tasks is hard as the likelihood is high that the model will produce an instance that contains gibberish. To combat these issues, stream-based and pool-based scenario's have been proposed.

With stream-based selective sampling, it is assumed that there is an underlying distribution from which the model can freely sample unlabeled instances one-by-one (Cohn et al., 1994). After sampling an instance, the model can then decide whether it wants to query it. This decision can be made in several ways. First, it can be done using some kind of informativeness measure or query strategy which are discussed later in this method (Settles, 2009). Second, it can be decided that only samples that come from a certain region of uncertainty are queried. Third and finally, a "version space" can be defined which is the set of hypotheses that can explain the labeled instances (Mitchell, 1982). This region takes the newly labeled instances into account, which is something that the region of uncertainty does not.

With pool-based scenarios, instead of just sampling from a distribution one by one, it is assumed that there is a small set of labeled data available which is known as $\mathcal{L}$ and a large pool of unlabeled data which is known as $\mathcal{U}$. This pool $\mathcal{U}$ is usually assumed to be static, but this is not necessary. Instances to be queried are selected from $\mathcal{U}$ based on an informativeness measure. This scenario is the most used in real-world instances (Settles, 2009).

All the previously discussed scenarios use querying to decide which instance will be shown to the oracle. There are several strategies that can be used for doing this. These are now explained.

### 2.1.2 Query strategies

While more than 20 query strategies exist (R. Hu, 2011; Kumar & Gupta, 2020; Settles, 2009, 2012; Settles & Craven, 2008), the tool that will be used to perform the AL part of this research only provides a few basic ones. Therefore, only the six strategies presented in the first AL literature survey by Settles (2009) are discussed. These strategies are uncertainty sampling, query-by-committee, expected model change, variance reduction, estimated error reduction and information density. Each strategy is now shortly elaborated upon. Note that in each strategy equation the $*$-sign stands for "best possible".

**Uncertainty Sampling**
The first and most widely used strategy is uncertainty sampling. With uncertainty sampling, the sample is queried about which the model is the least confident of the label it predicted. This confidence value can be evaluated using entropy for binary classification. The equation for entropy is:

$$x^*_{ENT} = \underset{x}{argmax} - \sum_i P(y_i|x;\theta) \log P(y_i|x;\theta) \tag{1}$$

In this equation $y_i$ stands for all possible labels, $x$ is the specific instance and $\theta$ is the model. With entropy,

the more uncertainty there is about a label, the higher the entropy will be. For multi-class classification it can be done by checking which instance has the lowest confidence for its best labeling in the following way:

$$x^*_{LC} = \underset{x}{argmin}\, P(y^*|x;\theta) \tag{2}$$

In this equation, $y^x$ is the most likely labeling for instance $x$. It is then determined for instance $x$ how confident model $\theta$ is.

**Query-by-committee**
A second and also widely used strategy is the query-by-committee strategy. With query-by-committee, instead of having just one model make a judgment on how uncertain it is about a sample, a committee of models is used. In this case, of all possible samples, the sample is chosen about which the committee disagrees the most. This sample is then queried. To measure the disagreement between the committee members, two ways have been proposed. These are *vote entropy* and *Kullback-Leibler divergence*. The equation for vote entropy is as follows:

$$x^*_{VE} = \underset{x}{argmax} - \sum_i \frac{V(y_i)}{C} \log \frac{V(y_i)}{C} \tag{3}$$

In this equation $V(y_i)$ is the number of votes a certain label has gotten and $C$ is the number of committee members. Again, the more disagreement there is between votes, the higher the entropy will be. The equation for Kullback-Leibler divergence is:

$$x^*_{KL} = \underset{x}{argmax}\, \frac{1}{C} \sum_{c=1}^{C} D(P_{\theta^{(c)}} || P_{\mathcal{C}}) \tag{4}$$

With Kullback-Leibler divergence, one tries to measure the distance between two probability distributions. In the case of the equation given in 4, $D(P_{\theta^{(c)}} || P_{\mathcal{C}})$ is the distance between a label given by a specific model and the rest of the models. By getting the highest average distance of the label given by all models in the committee of the specific instance, the most informative instance can be found.

What this strategy tries to do is in essence to minimize the hypothesis space $\mathcal{H}$, which is the current set of hypotheses that are still valid considering the labeled set $\mathcal{L}$ (Settles, 2012). The classifier that is finally returned by this strategy is the one that has the best hypothesis overall. A strategy that is roughly the same as query-by-committee is the use of redundant views. Instead of sampling randomly from the version space, separate smaller views are created that each represent the underlying distribution (Olsson, 2009). This strategy is explained more in-depth in section 2.1.3.

**Expected model change**
A third strategy is expected model change. This strategy tries to query the instance that would have the greatest positive effect on the accuracy of the model if its label was known (Fang, Li, & Cohn, 2017). This strategy works well for empirical studies but is computationally expensive if there is a large feature space and a large set of labels.

**Variance reduction**
A fourth strategy is variance reduction. With this strategy, the learner will try to query the instance which results in the greatest reduction in model variance. This strategy is not suited for problems with high dimensional data (Kumar & Gupta, 2020), which is the case in Natural Language Processing tasks. It is therefore not interesting for this study.

**Estimated error reduction**
A fifth strategy, called estimated error reduction, tries to minimize generalization error directly. This is different from all the other methods discussed so far, which all try to minimize the error by reducing model variance. The way this is done is by estimating the expected future error of the current hypothesis, using the posterior estimate, if an instance was added to $\mathcal{L}$ and then querying that instance for which this error is the lowest (Settles, 2009). This querying strategy has some major drawbacks, namely, that it is computationally expensive, it only works if the posterior estimate is good and finally a validation set is required (Kumar & Gupta, 2020).

**Information denstity strategy**
The sixth and final strategy is called the information density strategy. One of the problems with the uncertainty

and query-by-committee strategies is that they are prone to suggesting outliers for querying. While an outlier could be the instance about which the model is the most uncertain, the model will not learn much from it if is very far from the consensus. The information density strategy tries to battle this by also taking into account how well an instance represents the distribution it is from. This is mathematically done as follows:

$$x^*_{ID} = \underset{x}{argmax}\ \phi_A(x) \times \left( \frac{1}{U} \sum_{u=1}^{U} sim(x, x^{(u)}) \right)^{\beta} \tag{5}$$

In the equation, the first term represents the informativeness of a specific instance. The second term represents the similarity(sim) of the instance to the rest of the distribution by taking the average of the similarity of the instance to all other instances within the distribution. This second term is taken to the power of $\beta$ which controls the importance of this term. According to Settles and Craven (2008), this strategy out of all previously discussed strategies, performs the best overall for sequence labeling tasks.

In the case where the labels are non-binary, a different approach to querying needs to be taken. This was researched in the study by P. Hu, Lipton, Anandkumar, and Ramanan (2018), in which an image recognition algorithm was trained on different datasets that had multiple possible labels. In the case where there are multiple possible labels, it is mostly not possible to generate a correct label from a single yes/no question. Therefore multiple questions need to be asked. To store what is currently known about the label of the specific instance that is queried, a partial label is created. This label is updated each time a question is asked and once only a single class remains on the basis of the information found in the partial label, the process of asking the oracle question about the specific instance is stopped. It is found that the method proposed in the study is able to label all instances with 42% fewer binary questions in multi-class scenarios (P. Hu et al., 2018).

### 2.1.3 Views

As stated before, the main goal behind AL is to reduce the number of labels necessary for properly training a supervised machine learning model. Within AL, there is a concept that tries to lower the amount of labeled data necessary even more. This concept, which is used within the so-called redundant views approach (Olsson, 2009), tries to split the dataset into smaller subsets called "views". The idea is that if it is possible to split the dataset into smaller subsets that each equally represent the underlying distribution well enough, time can be saved. When a single model is used in this case, it will only need to be trained on a single view, since it should reach the same accuracy as compared to when it is trained on all views (Muslea, Minton, & Knoblock, 2003). As a result of this, one will only have to go through a smaller part of the data, saving effort in the process. When multiple classifiers are used, views can also improve the performance of the Active learner. Since each can be trained using a different view, this should result in a more robust set of classifiers since each will represent the entire dataset equally well while not having to use the entire dataset (Olsson, 2009). After a stopping criterion is reached, the classifiers are combined using one of the following approaches (Muslea et al., 2003):

- *Weighted vote*: The classifiers are combined using a weight that is formed based on their own confidence rates.
- *Majority vote*: The classifiers are combined in such a way that the label that is predicted by the final classifier conforms to the majority vote of all classifiers.
- *Winner-takes all*: The final classifier is the classifier from the pool of all classifiers that performed the best overall during the entire training process.

One requirement of these approaches is that the views are "strong" meaning that they provide enough data for a model to learn a target concept. The opposite of the "strong" view is the "weak" view, which only contains enough data for a model to learn a concept that is either more general or more specific than the target concept. There have also been approaches that include both strong and weak views, such as the so-called "co-testing". The co-testing approach has outperformed other AL algorithms in different tasks (Muslea, Minton, & Knoblock, 2006). Note that a requirement for using views with AL is that it is actually possible to split the data into views and that these views are not correlated to each other.

### 2.1.4 Stopping problem

As stated in section 2.1, AL is an active field of research. One of the problems in the field which has not been solved, but is also not likely to be solved in a general way, is the stopping problem (Zhu, Wang, Hovy, & Ma, 2010). The stopping problem concerns the search for the ideal moment of stopping the AL process when training the model. This is a vital issue, as it does not make sense to continue the AL process until every single one of the unlabeled instances has been labeled manually by the oracle. Instead, a trade-off needs to be made between model performance and labeling cost. One could take the moment when model performance drops or halts during the training phase. However, it may be that there is still a significant increase in model performance after such a moment (Zhu et al., 2010). One could also specify a " label-budget" which is the number of labels an oracle is allowed to give (Novak, Mladenič, & Grobelnik, 2006). Much research has been done to find the best stopping criterion (Laws & Schätze, 2008; Vlachos, 2008), however, it is an open problem due to the fact that finding the "perfect" stopping criterion is highly dependent on the structure of data and the type of models that are used.

## 2.2 Natural Language Processing

Human text is hard for computers to understand as it contains meanings that cannot easily be extracted if one does not properly understand the underlying grammar. Things such as sarcasm and figure of speech make this task even harder. However, text or natural language contains an enormous amount of interesting data, making it worth it to research ways to extract this information reliably. The research area that deals with finding these ways is called Natural Language Processing (NLP) (Chowdhury, 2020). There are a vast majority of tasks that fall under the NLP-umbrella. They can be subdivided into low- and high-level tasks. According to Nadkarni, Ohno-Machado, and Chapman (2011), low-level tasks include things such as sentence boundary detection, tokenization, part-of-speech tagging, morphological decomposition of compound words, shallow parsing and problem-specific segmentation.

There are also high-level tasks which build on the low-level tasks. These high-level tasks, again according to Nadkarni et al. (2011), include spelling/grammatical error identification and recovery, Named Entity Recognition, word sense disambiguation, negation and uncertainty identification, relationship extraction and information extraction. Other tasks that are not listed in this article are things such as sentiment analysis and topic modeling/mining. Of these high-level- tasks only a few are relevant to this research, namely sentiment analysis and Named Entity recognition. These will be discussed more in depth below.

### 2.2.1 Sentiment Analysis

Sentiment Analysis is the task of determining whether a piece of text, also called sentiment, has a positive or negative value towards some kind of product, service, organization, individual, event, issue or topic. These texts are expressed by humans and can contain emotions, appraisals, attitudes or opinions (Liu, 2020). Sentiment analysis can be performed at three levels, namely document-level, sentence-level and aspect-level sentiment analysis and is usually performed with a model that is pre-trained on related texts. With document-level sentiment analysis, an entire document is analyzed on its sentiment. This is done under the assumption that the document contains an opinion on one main object expressed by the author of the document (Feldman, 2013). Document-level sentiment analysis can be done both in a supervised and unsupervised way. With the supervised approach, a model is simply trained on a set of documents with a finite set of labels. The most simple case would be just two labels in the form of "negative" and "positive". Newly presented documents can then be classified using this model. A pre-trained model can also be used as stated before. With the unsupervised approach, the classification is done using the semantic orientation of specific phrases within the document (Feldman, 2013). If the average semantic orientation of these sentences within the document exceeds some kind of threshold, the document is deemed positive.

The idea behind sentence-level sentiment analysis is essentially the same as document-level sentiment analysis. The only difference is that instead of classifying an entire document, this task is performed at the level of a single sentence. Again, each sentence may only include a single opinion. If this is not the case, the sentence must be split up into smaller sentences which do contain just one opinion.

The third and final type of sentiment analysis is aspect-level sentiment analysis. Where the previous two types of analysis only took a single aspect of the entity into account when determining the sentiment value of

something, aspect-level sentiment analysis tries to take into account all possible aspects of the entity. An aspect can for example be for a food review if the review was positive in terms of the service or if it was positive in terms of the food. By doing this, a more informative sentiment analysis can be performed.

To actually perform sentiment analysis, a corpus is necessary. This corpus can be a preexisting one, e.g. a lexicon, which is usually the easiest but will not guarantee the highest quality of labels, or a generated one based on the texts that are available and used for classifying.

Sentiment analysis is most commonly used in the area of reviews of consumer products and services. Besides this application, it has also been used to analyze the stock market (Khedr & Yaseen, 2017; Mittal & Goel, 2012; Pagolu, Reddy, Panda, & Majhi, 2016; Rao & Srivastava, 2012). This is done by analyzing tweets about a certain stock and then based on the results found the movement of the stock is predicted. Furthermore, it has been applied in other fields that benefit from knowing the opinion of the public such as healthcare and politics (Liu, 2012).

### 2.2.2 Named Entity Recognition

Named Entity Recognition(NER) is "The problem of locating and categorizing important nouns and proper nouns in a text" (Mohit, 2014, p.221). NER is a sub-task of the previously mentioned high-level NLP task Information Extraction. With NER other more complex NLP tasks such as machine translation become easier to perform. There are essentially two ways two perform NER. The first, simpler, method makes use of a rule-based system. With this rule-based system, a set of named entity extraction rules are used in combination with a lexicon and an extraction engine that applies the rules to the text to extract entities. While rule-based systems are precise, they are not suited for larger domains. The second method, which is more complex, is able to work with bigger domains. This method, called statistical NER, makes use of labeled training data and a statistical model to represent the training data in a probabilistic way. While regular classification can be used to solve this problem, it is better to use a structured learning approach in which the labels for all words in a sentence are predicted at once (Schraagen, Brinkhuis, & Bex, 2017).

### 2.2.3 Bag of words and TF-IDF

While the NLP field consists of many tasks, two of which have been discussed above, it also consists of many useful metrics. Two of these which are important for this research are bag of words and TF-IDF. Bag of words is a way in which one can represent the contents of a document using a single array of numbers. Each number in the array represents how many times each word found in the entire corpus of all documents in a dataset appears in the document. While this sounds very simple, the array can get very large, making bag of words less usable in those scenarios. This does not take away from the fact that it is a very useful way to have a compact representation of a text (H. K. Kim, Kim, & Cho, 2017). Two ways in which the usability of bag of words for larger sets of documents can be improved are the use of TF-IDF and feature hashing. TF-IDF, which stands for Term Frequency - Inverse Document frequency, is a way to determine how unique a certain word is in a certain document. The higher the TF-IDF score, the more unique the word is in the document. This is calculated by taking the relative amount of times a word appears in a document, divided by the relative amount of times the word appears within all documents. While it is not specifically known in which paper TF-IDF was first mentioned, Hans Peter Luhn has provided great work on TF (Luhn, 1958) and Karen Spärck Jones contributed greatly to IDF (Sparck Jones, 1972). With this, the usability of bag of words can already be increased since with the use of this metric, the more unique words will stand out more. A second way in which usability can be increased is through the use of feature hashing (Weinberger, Dasgupta, Langford, Smola, & Attenberg, 2009). With feature hashing, the entire vocabulary of words that is found within a set of documents is hashed so that the size of the final array is downsized from 60000+ numbers to less than 500 numbers. With feature hashing, instead of using a word as a representation, a hash is used, which is a numerical representation of that word that is generated through a hashing function.

## 2.3 Neural networks

Many types of supervised machine learning methods exist. However, the most commonly used ones for NLP tasks usually involve some form of a neural network. In this research, only the transformer architecture is utilized in an NLP context. However, since this architecture is based on more simple neural networks such as

feed-forward and recurrent neural networks, these are introduced first in a general way. Their NLP-specific implementation is not discussed, as it is not used in the research. The NLP implementation of the transformer architecture is discussed but this happens in section 2.3.1.

Neural networks are a type of supervised machine learning technique that tries to mimic the human brain. A neural network consists of layers of sets of neurons which take an input to generate an output. A neural network essentially consists of three parts, namely a single input layer, one or more hidden layers and an output layer (Abiodun et al., 2018). One of the most simple types of a neural network without cycles is a Feed-Forward Neural network, of which an example can be found in Figure 2.

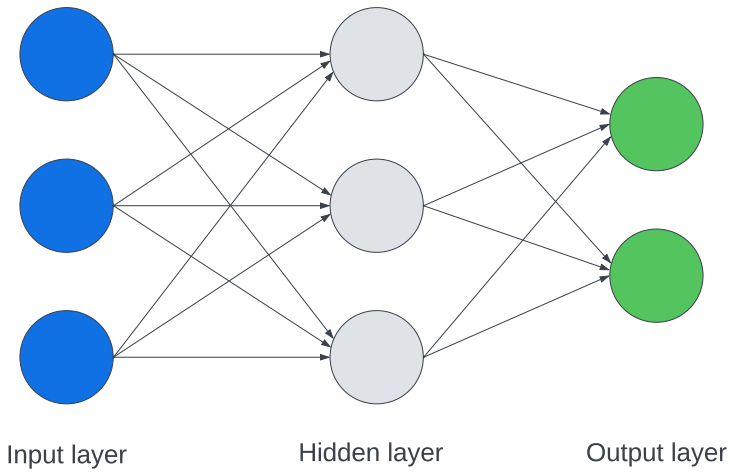



Figure 2: An example of a simple Feed-Forward Neural network.

Within each layer, certain neurons get activated based on the input they get. This activation works in the following way. Take for example the upper neuron in the middle layer in Figure 2. This neuron gets an input from three neurons in the previous layer. The inputs are first multiplied by an associated weight for that neuron, and then all weighted inputs are added up. If this sum exceeds some kind of threshold within the activation function, it will produce an output for the next layer. The mathematical representation for this activation is:

$$AV(\omega_0 x_0 + \omega_1 x_1 + ... \omega_n x_n + b) \tag{6}$$

In this formula, AV stands for activation function, $x$ stands for input, $\omega$ for the weight associated with the input and $b$ for the optional bias. The subscripts 0, 1 or $n$ indicate from which node a certain input is coming. An $n$ is used as opposed to a real number as there is no limit to the number of inputs a neural network can have. If the threshold of activation function $AV$ is exceeded, an output will be generated. Common activation functions include Sigmoid, Tanh, ReLU and Leaky ReLU (Sharma, Sharma, & Athaiya, 2020). It is also possible to include bias nodes in each layer, which make the network more flexible by allowing for easier activation of neurons in layers (Abiodun et al., 2018). In the final layer of the network, the probabilities for labels are computed. In the example shown in Figure 2 two labels are possible as there are two neurons in the final layer. However, this final layer can contain any amount of neurons corresponding to the desired amount of output labels.

Many types of neural networks exist, however the most commonly used ones for NLP are Recurrent Neural Networks (RNN) as they are much better at solving problems involving sequential data, of which NLP is one, than simple Feed-Forward neural networks (Goldberg, 2017). When dealing with sequential data it is useful to know what happened in the previous iteration, but Feed-Forward neural networks do not allow you to use this information. The way RNNs deal with the problems Feed-Forward neural networks have with sequential data is by using *hidden states*, which are a direct connection between the last iteration and the current iteration. By using hidden states, the formula shown in 6 is modified as follows:

$$AV(X^{(T)} + X^{(T-1)} + b) \tag{7}$$

In this formula, $X^{(T)}$ represents the sum of all separate inputs multiplied by their weights in the current iteration and $X^{(T-1)}$ represents the same but from the previous iteration also known as the hidden state. While this helps slightly with remembering near-previous states, e.g. short-memory, it does not help much with remembering far-previous states, e.g. long-term memory. In other words, they have great short-term memory, but bad long-term memory. A specific type of RNN that combats this problem is Long Short-Term Memory network (LSTM), which makes use of gates to determine what part of the information of the previous iteration is remembered and what part is forgotten. LSTMs were the golden standard for solving NLP tasks for a long time (Ghosh et al., 2016). However, relatively recently, a new architecture was designed that is even better than LSTMs. This architecture is called "Transformer", which will be explained in the following section.

### 2.3.1 Transformers

Transformers focus on only one defining aspect of the previously discussed neural networks, making the architecture much simpler. The transformer takes a complete sentence as input to generate, for example, the next word that follows in the sentence, a translation of the sentence or the emotional value of the sentence. In Figure 3, a visual representation of the architecture can be found.

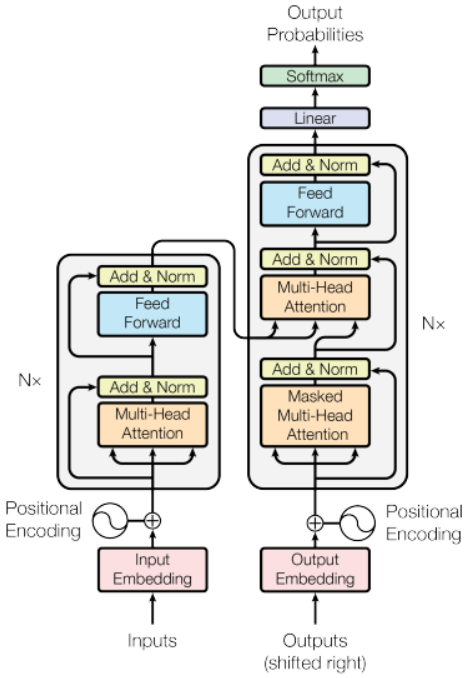



Figure 3: A visual representation from the transformer architecture designed by Vaswani et al. (2017).

In short, the transformer consists of an encoder and a decoder. The encoder tries to get a representation of the input sentence that contains as much useful information as possible in a way that a computer can understand it. The decoder, on the other hand, takes the target sentence, decodes it and uses the input from the encoder to generate an understanding of the sentence. The product of this is finally fed into a Feed-Forward Neural network after which it is converted into a softmax function so that, by using probabilities, the next word can be generated (Vaswani et al., 2017). While word generation was one of the first applications of the transformer architecture, it has been used for many other tasks such as sentiment analysis, question answering, NER and summarization and has achieved state-of-the-art performance for all of these tasks (Wolf et al., 2019). Note that every separate part, except the final linear Feed-Forward Neural network and softmax layer, has a residual connection to fight against the vanishing gradient problem (He, Zhang, Ren, & Sun, 2016). Where the transformer differs from other neural networks and in particular RNNs and LSTMs is the lack of recurring or convolutional features. Instead, it focuses on so-called "attention" which is also used in other neural networks but not as prominently.

Attention works as follows: First and foremost, the goal of attention is to find those parts of the sequence on which the focus should be. This is done through queries, keys and values. A key is simply a label of a word to distinguish between the word and other words. A key has an underlying value that represents the information it contains. Finally, a query is used to find another key in the sentence that matches the best to the key in question. The queries and keys are then multiplied to get a score value. The higher the score value, the more similar the words are. These score values are then passed through a softmax function to normalize the score vector so that its values add up to 1. By doing this, query-key pairs that are closer to each other will stand out more. Finally, the score vector is multiplied by the value vector. The resulting vector now represents the meaning of the word and its context. In the transformer architecture, this is not done just once. Instead, it is done multiple times to get the multi-head attention of the sentence. Note that not every single score-value vector is taken through the rest of the transformer. Instead, an average is taken of all the score-value vectors.

Attention is used three times in the transformer architecture. Both in the encoder and decoder, so-called "self-attention" is used. This type of attention is called self-attention, since the sentence is trying to pay attention to the important parts of itself. In Figure 3, the parts where this type of attention is used are the Multi-Head Attention module in the encoder (left part) and the Masked Multi-Head Attention module in the decoder (right part). The second type of attention that is used is Encoder-Decoder-attention, which is used in the Multi-Head Attention module decoder part of the module. This type of attention gets its name from the fact that the decoder tries to pay attention to the inputs it gets from the encoder (Vaswani et al., 2017). Note that this is only a high-level overview of the transformer architecture and its unique components.

One of the main reasons why Transformers perform much better than both RNNs and LSTMs is because they manage to avoid recursion completely. Because of this, the model also does not need to process the sentence word-for-word but instead handles the sentence in its entirety all at once (Vaswani et al., 2017). One limitation the transformer architecture still has is that it can only remember the dependencies between the sentences and words it is fed during a single iteration. This means that if a sentence of length 50 is fed into the transformer, it will only be able to remember the dependencies between up to 50 words. In 2019 a model called "Transformer-XL" was proposed to combat this issue by re-introducing the concept of hidden states on which the RNN architecture is based. This model performs much better for longer text, being to handle texts that are 450% longer than vanilla transformers (Dai et al., 2019).

## 2.4 Active Learning with NLP

AL is a very active research field, and many successful applications of it by itself but also in combination with NLP tasks have been documented. Since in this research NLP will also be performed to get features from the documents, some implementations will now be discussed.

In a study by Waltl et al. (2017) AL was used to classify German legal norms. In the method they used, which can be seen in Figure 4, a combination of a rule-based system and an AL process was used. To generate both the rules and assign labels to the queried instances, a legal expert was consulted. For classifying the legal norms in the AL part of the method three different classifiers were separately used, namely multinomial naive bayes, logistic regression and a multi-layer perceptron. It turned out that using each of the separate classifiers in an AL way resulted in not only an increased speed of learning, but also a higher maximum accuracy. It should be noted here that only after the earlier discussed "version space" was properly discovered, these significant improvements over standard supervised learning techniques became visible. After having labeled only 17% of all instances, a 20% higher accuracy was observed for the AL techniques.

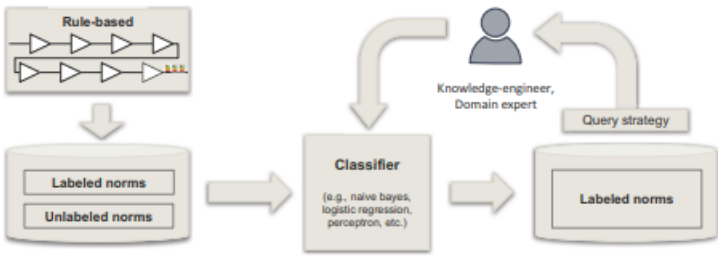



Figure 4: An overview of the method used in Waltl et al. (2017).

In the paper by Nguyen and Patrick (2014) a combination of AL and NLP was used to classify radiology reports. In this paper, a process was investigated in which, previously, all radiology documents were manually scanned through to find out if a patient had cancer or not. This was a very tedious process, as usually a person would have to scan at least nine documents to find a document relevant enough for further investigation. Within the research, four types of AL models were tested. The method used in the study consisted of a pre-processing, training and testing pipeline. In the pre-processing pipeline, a tokenizer was used to split the text into tokens, after which the tokens were validated using lexical verification. In the training pipeline, a conditional random fields model (Lafferty et al., 2001) was first used to help tag the documents in such a way that the medical information became more clear. After this, a Support Vector Machine model was trained, which was used for the actual classification of the documents. With the usage of AL it was found that, as compared to normal supervised learning, an F1-score of 95% or higher was achieved, compared to an F1-score of 72% for a supervised learning model with random sampling. This was done using up to 90% fewer data. One thing that should be noted from this study is that the entire dataset was labeled beforehand and this study simply tried to test if an appropriate model could be trained faster by using AL. This scenario is not the case for the Shell dataset used in this study which comes completely unlabeled. However, a completely custom and very detailed tagging system was built so that the classifier could understand the text it was given much better. This could be interesting for this study as well, depending on how complex the texts found are.

In the paper by Tran, Nguyen, Fujita, Hoang, and Hwang (2017) a combination of AL, self-learning and NLP was used to improve the task of performing NER on tweets by reducing the amount of time necessary to label instances. Since this is a continuous task, as Twitter is not a static platform but rather new tweets appear every second, it is a task that requires many man-hours if done manually. To select instances to be labeled and added to the training, a few methods were used. First, regular AL was used to select the most informative instances. Second, vector models created with Word2vec were used to measure the similarity score of the instances which helped in performing the NER (Mikolov, Chen, Corrado, & Dean, 2013). If the predicted label sequence of an instance had high confidence, that instance was also added to the training data. For the final classifier, a conditional random fields model was used, which can be used for labeling and segmenting sequential data. They also represent the probability of a hidden stage sequence, given the observations. In the end, it was found that the proposed hybrid method, achieved better results than other methods and was better than baselines by 5% in terms of accuracy and F1-score.

In the study by Das Bhattacharjee et al. (2017) a combination of NLP and AL was used to build a model that could detect fake news. While news datasets can be found in abundance online, most of these datasets did not contain labels, which made it very difficult to train a supervised learning model on the data. This is where AL came in, since it makes it possible to create a supervised model with much fewer labels. The news content used in the study came from Twitter and was either a short tweet or a longer paragraph. The model for the task consisted of two separate iterative parts which were finally fused to come to a decision. One part of the model applied lexical feature-based AL with weighted feature updates, whereas the other part performed deep learning with AL based fine-tuning. After applying the model and comparing it to regular supervised learning techniques, it turned out that the AL variant of the combined model had an average accuracy that was only 0.01% lower than the supervised approach, while on average requiring 70% less training samples.

## 2.5 ASReview

One of the ways of using AL that has not been discussed yet, but is the way it is used in this research, is the systematic reviewing of documents. This application, which stems from the issue that getting relevant documents from a larger set of documents is very time-consuming, has mainly been researched from the perspective of the academic world. To give an example, Wallace et al. (2010) performed a study to get an estimation of how long it takes an expert to review a single medical paper. It was found that this can take from 30 seconds up to several minutes. Note that while this paper specifically looked at reviewing medical papers, it can be assumed that reviewing complex documents on specific aspects, in general, takes large amounts of time. Now also take into consideration that usually a very small amount of the documents is actually relevant according to the specified aspect, also referred to as class imbalance (Hashimoto, Kontonatsios, Miwa, & Ananiadou, 2016). This illustrates how tedious the process of systematically reviewing a set of documents based on a set of requirements is. While supervised machine learning techniques have been used in the past to solve this problem, they are not the perfect solution as they require many labeled instances, which again take

time to generate. This is where the previously discussed AL comes in, as it allows us to label a large set of data using fewer labels given beforehand while training a supervised machine learning classifier in the process. A tool that was made for performing systematic reviews with AL is ASReview (van de Schoot et al., 2020). With ASReview, one can easily perform AL using a number of different supervised machine learning models such as naive Bayes, logistic regression, LSTM and random forests. The tool also includes methods for feature extraction such as Doc2Vec, TF-IDF and sBert. In Figure 5, the pipeline of the ASReview tool is shown.

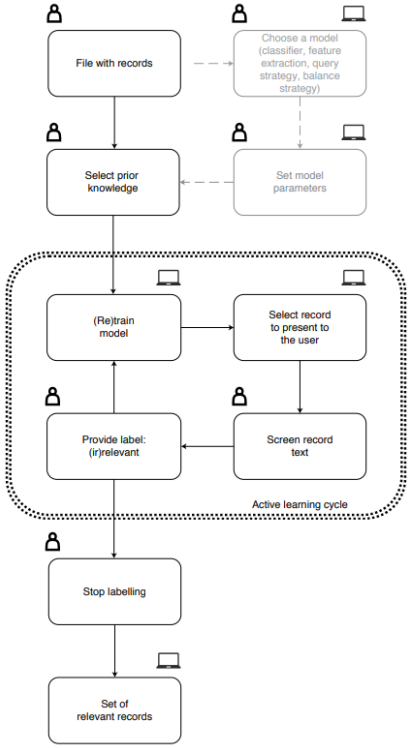



Figure 5: The pipeline behind the ASReview tool. Note that the symbols at each activity indicate whether the activity is performed by a human, a computer or both (van de Schoot et al., 2020).

So far, ASReview has been for projects such as a systematic review on the AI-aided Systematic reviews within clinical guideline development (van den Brand & van de Schoot, 2022), the evaluation of the performance of AL to support the selection of relevant publications within the content of medical guideline development (Harmsen et al., 2021) and a systematic review of data about depression (Brouwer et al., 2019).

## 2.6 Similar research

To give an idea of the current landscape, this section will discuss similar studies that have attempted to find very specific things such as corruption using NLP.

In the study by Rabuzin and Modrušan (2019), NLP techniques were used to detect corruption in public procurement bidding in Croatia. As one-third of the money spent by the state is spent through public procurement, and this money is mostly generated by the taxpayer, it is crucial to make sure this money generates value. Due to the dissatisfaction of the public, this study tried to find a new way to make sure no corruption was happening. This was attempted by performing text classification on certain tender documentation by using three different machine learning methods. These were linear regression, Support Vector Machines and naive bayes. By using these techniques, an average accuracy of just below 70% was achieved. While this was decent, the average ROC is just a bit lower, coming in at just under 0.6. After dividing the data into more subgroups, all models relatively performed better, with accuracy reaching above 80%. This shows that with the proper data setup, classifying text on very subjective and difficult concepts as corruption is certainly possible.

In another study, by Goel, Gangolly, Faerman, and Uzuner (2010), NLP was used to detect corruption in the annual reports of companies. This had previously only been done by looking at the quantitative information, while the qualitative data could also provide useful insights. For this study, annual reports of companies with and without known fraud were used. With the first baseline model, in which a bag-of-words approach was used, a naive bayes classifier achieved an accuracy of 56.75%. This accuracy was already increased when using a baseline Support Vector Machine approach, with an increase to 71.67%, which is something the researchers expected. Then, by using 4 sets of style and content features it was attempted to improve the model. The first set of features consisted of surface-level features such as the average length of the words, the standard deviation of the word length and the average length of the sentences. While simple, they were proven to be useful in the past (Goel et al., 2010). The second set of features consisted of voice, frequency of uncertainty markers, tone and readability. The third set of features consisted of deeper linguistic features such as vocabulary frequencies and vocabulary richness. The fourth and final set of features consisted of content-related features such as keywords, TFIDF scores and bigrams. By using a ranking of features based on the amount they contributed to the classifier performance, the top 10 features were selected. With these features in combination with a Support Vector Machine approach, an accuracy of 89.61% was achieved. This shows that with proper feature selection, identifying corruption by just looking at the text is very much possible.

To conclude, while AL for the scanning of documents has been researched before, and to aid the investigation into corruption AL has been suggested as a possible solution (Carcillo et al., 2017; S. Kim et al., 2020), as far as the researcher knows, they have not yet been combined with a focus on NLP features. This is what will be attempted in this study.

# 3 Data

Without any data, this research cannot be performed and the research questions cannot be answered. Within this research two datasets are used, one mostly for testing the initial method and fine-tuning it and the other for finding out if identifying complex things such as storylines is actually possible with Active Learning (AL). Some background behind the datasets and what data is left out of the research is discussed in this section.

## 3.1 Datasets

The first dataset belongs to Follow The Money (FTM) and contains roughly 7500 documents that supposedly contain information about the relationship between the Dutch government and Royal Dutch shell (FTM, 2019). FTM is a Dutch journalism platform that conducts in-depth investigations into people, systems and organizations who are suspected to misbehave in a financial-economic way and by doing this do damage to groups within society FTM (2020). This dataset called "Shell Papers" has been freely available online since 2019 and is the result of 17 WOO-requests that were made to get every single document in possession of the Dutch Government that was sent by, addressed to or concerned Royal Dutch Shell. A WOO-request, which is a Dutch acronym for Government Information Act, is a request that can be made to the Dutch government to publicly release certain documents (*Requesting information from the government (WOO request)*, 2022). In this case, this concerned documents, all of which are written in Dutch, that contain information about the relationship between Royal Dutch Shell and the Dutch government in the broadest sense of the word (FTM, 2019), possibly including text that can be seen as evidence of possible corruption. Since the creators of the dataset are not yet at the stage of identifying possible evidence of corruption within the dataset, this research will aid in the current step of the investigation, which is identifying the main storylines within the documents.

The second dataset is the Enron dataset and contains about 0.5 million emails, all written in English from the now-bankrupt energy company Enron (Klimt & Yang, 2004). This dataset has been used for analyzing emails before, as it has already been available for quite a while, being released in 2004. However, unlike many other datasets, small parts of it have been labeled. Next to that, the labels are rather simple, meaning that if the labeled set is not deemed large enough, the rest of the dataset can be labeled by people who are not necessarily experts in the field. The labeled part of the dataset has very specific labeling categories. For this research, however, only one of the most simple labels is used. Namely, the label that tells whether an email contains company logistics or not, or more specific information about meeting scheduling, technical support or other logistical arrangements (*UC Berkeley Enron Email Analysis*, 2020).

## 3.2 Shell papers storylines

Within the Shell papers, various storylines have already been identified. These are "Adviestraject" or advisory process, "Relatie Assen NAM" or the relationship between the city of Assen and the NAM (Nederlandse Aardolie Maatschappij or Dutch Oil Society), "Relatie Assen-Provincie" or the relationship between the city of Assen and the province in which Assen is situated, "Drenthe 4.0" and finally "Relatie provincie NAM" or the relationship between the province and NAM. All these storylines contain multiple labels, which means they all could be used for performing AL. Due to the fact that a human would need to label the documents on whether they belong to a storyline or not, and ASReview only allows binary labeling, a choice was made for a storyline to test the AL on. Since the expert in question was most knowledgeable on the storyline of the relationship between the city of Assen and the NAM, this storyline was chosen.

## 3.3 Exclusion of data

Within the Shell papers dataset, there are 10 types of document categories. These are:

- **Correspondentie(Correspondence)**, contains various kinds of correspondence.
- **Mail(Mail)**, contains relevant emails sent by civil servants, employees of third-party companies and employees of Royal Dutch Shell. Sometimes these emails contain very large attachments.
- **Document(Document)**, contains various types of documents.
- **Vergunning(Permit)**, contains permits and reports of analyses performed by external parties.

- **Rapport(Report)**, contains reports of certain activities commissioned by the Dutch government.
- **Bestuurlijk Besluit(Administrative decision)**, contains summarising reports of administrative decisions made by the government.
- **Overeenkomst(Agreement)**, contains a single agreement about the usage of fuel between the municipality of Assen and Royal Dutch Shell.
- **Onbekend(Unknown)**, contains a single document with a link to a 900+ page file with a reaction to a plan about extracting natural gas from a specific gas field in the Netherlands.

Of these categories, only "Correspondentie", "Mail" and "Document" are interesting to train an NLP model on for the following reasons.

First, these documents are rather uniform in their structure. This makes it so that it is possible to extract the data properly from most documents using just a single script. If the other categories were to be included, many extra custom data extraction scripts would need to be written. Second, these documents contain mostly natural language, which is essential for performing NLP tasks, as opposed to the excluded documents which contain mostly numbers and formatting commands. Finally, when using the aforementioned 3 subgroups, more than 90% of the dataset is captured. This is enough to sufficiently capture the attributes necessary for creating a classifier. It is therefore not worth it to try and get a higher percentage by including the other subgroups. For these reasons, the other subgroups will be left out of the research.

From the Enron dataset, no documents will be left out.

## 3.4 Data preprocessing and preparation

To actually use the data, it first needs to be properly prepared by means of cleaning and preprocessing. The Shell papers dataset requires more effort than the Enron dataset as this is the first time the Shell papers dataset is used for training a full-fledged supervised machine learning model, as opposed to the Enron dataset which has been used for many text analysis papers before (Nakano, Cerri, & Vens, 2020; Reyes, Morell, & Ventura, 2018; Wang, Feng, Shan, & Min, 2022). All code for data preprocessing and extraction can be found in appendix D.

### 3.4.1 Shell Papers

To be able to extract features from the data, preprocessing is necessary. Especially since the data concerns natural language. The dataset to be used is partially extracted from Github (Tokmetzis, 2021). This is about a quarter of all the documents found on (FTM, 2019). This subset is used for determining the right preprocessing steps. The other part is added later, as it is not directly available to the researcher. By combining these two datasets the total amount of documents becomes 7 576.

To start the preprocessing, the data is loaded in. Unnecessary columns are removed and columns containing missing values are dropped. This might seem risky, but not performing this type of cleaning can result in too sparse data. The content is then transformed from HTML to normal Python strings. Then the id, type and date are extracted from the title of the content if these are present. The dates are properly formatted after this. Next, the title of the document is cleaned of any irrelevant characters. Files that do not contain useful information for determining possible storylines are also removed. Examples of these kinds of files are posters or leftover checksums from the Github file import.

Next, more specific preprocessing is performed on the documents using the function splitdocuments. Note that this preprocessing method is mostly fine-tuned for the Github data and not the other data as the other data was received very late, but the method almost fully works on the latter part. This method not only extracts the useful text from a document, but also splits a document if the document contains more than one document. This is done by looking for specific Dutch words in the text that point to the beginning and end of texts that were found in a preprocessing exploration phase. The method is then applied to the relevant data that has been specified earlier. Finally, some extra cleaning is done on the text. A column is also added in which the length of the relevant abstract is put. After having split all the documents, the amount of instances in the dataset increases from 7 576 to 13 593.

After all the data is cleaned, a preliminary label column needs to be added to the dataset to be able to kick-start the AL. As specified in the previous section, ASReview only allows for binary labeling, and the expert that is involved with the AL process is most knowledgeable on the storyline that is about the relationship between the city of Assen and the NAM. The preliminary label column will therefore contain a 1 if the document in question belongs to this storyline and a 0 if it does not. Since the labels provided by FTM only contain 1's for each storyline, a different approach needs to be taken in order to also gather documents that can be labeled with a 0. This is done by looking at which documents have been labeled with 1's for storylines other than the relationship between the city of Assen and the NAM, and then giving the documents the label 0 if they do not appear in the storyline of the label but do appear in another storyline. By doing this, the dataset ends up containing 183 1's and 416 0's. Do note that the amount of 1's and 0's was largely increased due to the number of documents that were split. Without the splitting, there are 99 1's and 161 0's. For details on the exact code, please see appendix D.1.

### 3.4.2 Enron dataset

For the other dataset, which is the Enron emails dataset, less pre-processing is necessary. The entire Enron dataset contains 0.5 million emails, but these are not labeled (Klimt & Yang, 2004). Therefore, initially, a smaller subset is used, which has been labeled by members of the University of California, Berkeley (*UC Berkeley Enron Email Analysis*, 2020). If this subset is not deemed large enough, it will be attempted to manually label the rest of the dataset. The data can be loaded rather easily, as it has already been put in a readable format, namely MIME. The emails are read by using the Parser library in Python (Warsaw, Wouters, & Baxter, 2007). Only the body of the emails is used. The labels are read in by using a custom script. The labels are very in-depth, but for this research, only one of the most general labels is used, which contains information on whether the email was about company logistics or not. The labels and emails are in separate files and are finally combined into a single dataframe. For details on the exact code, please see appendix D.2.

# 4 Method

The main goal of this research is to be able to help identify documents that are part of a storyline to help the investigation into evidence of possible corruption or to identify documents that contain other specific types of information, using just the information found within the documents. Active Learning in combination with NLP techniques is used to attempt this. These documents will be coming from one of two datasets, which have been discussed in section 3. The main research question to be answered in this paper is: **To what extent can Active Learning in combination with Natural Language Processing be used to help identify possible cases of corruption and company logistics in business documents?**

The sub-research questions are:

- **SQ1:** Is it possible to identify documents that adhere to complex criteria using Natural Language Processing in combination with Active Learning?
- **SQ2:** What type of lexical features are more useful for training an Active Learning model to help identify storylines or information about company logistics in business documents?
- **SQ3:** What are the aspects of the datasets that allow for efficient training of the model?

To be able to answer the main research question and the-sub research questions, the CRISP-DM method is used. CRISP-DM, which stands for Cross Reference Industry Standard Process for Data Mining, is a method for performing data mining tasks that was conceptualized in 2000 (Chapman et al., 2000). A diagram of the method can be found in Figure 6.

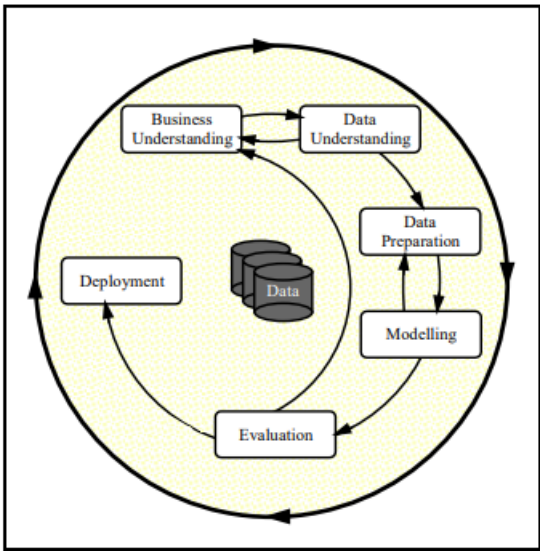



Figure 6: The CRISP-DM way of working (Chapman et al., 2000).

The CRISP-DM method is chosen as it has been proven to be a robust and reliable way to perform data mining projects of any kind. Besides, it is easy to understand, explain and implement.

Within this research, the business understanding phase has already been executed in the related works section and both the data understanding and data preparation have been discussed in the data section. This section will discuss feature extraction along with the model selection and feature selection, which are both part of the modeling phase. A description of the evaluation is also given. The actual evaluation can be found in the results. A description of the possible deployment can be found within the discussion. Note the word possible, as the deployment of the model within this research is not part of this research.

## 4.1 Feature extraction

To be able to perform Active Learning (AL), a supervised machine learning model is necessary. This model is trained using features that are extracted or generated from the data. In essence, anything can be defined as a feature. From the length of a document to the number of times a certain word appears. What makes a feature strong, is that it divides the data into clear, distinct groups. The features in this study are divided into two groups, namely simple and complex lexical features. The distinction between these two feature groups is made as follows: if a feature requires multiple computations to be obtained, it is a complex lexical feature. If it can be obtained with little to no computation, it is deemed a simple lexical feature.
For this research, the following features are defined:

- Length of the document
- Presence of certain words
- Sentiment score of the entire document
- Bag of words
- Named entities in the document
- Standard deviation of sentence lengths
- Automated readability index
- Standard deviation of word lengths
- Type-token ratio
- Amount of proper nouns
- Percentage of sentences written in passive voice

Of all these features, the last 6 were taken from the paper by Goel et al. (2010) in which they were proven to be useful for detecting corruption. Even though this is not something that is explicitly present in the Shell Papers, it is searched for in the Shell Papers since there are suspicions of this. These features will not be explained in depth, but it will be stated to which category they belong.

### 4.1.1 Simple lexical features

The length of the document is chosen as a feature since there could always be a relation between a certain storyline and document length, or logistics in the case of the Enron dataset. Generating this feature is simple, as only a length function needs to be called the document.

The presence of certain words is a more elaborate feature, as this is different for every dataset that the method is used on. In the Enron dataset, words were used that seemed to point to emails that contained information about logistics directly related to the company. These were “Meeting”, “Plane”, “Expense report”, “Voicemail”, “Email”, “Weeks”, “Schedule”, “Time”, “Week”, “Invite” and “Call”. For the Shell papers dataset, words were used to seemed to point to possible storylines. Examples of these words are “Assen” and “NAM”. The feature is computed by taking the sum of all occurrences of each of the words in the text.

From the paper by Goel et al. (2010), the features standard deviation of sentence lengths, automated readability index, standard deviation of word lengths and type-token ratio are considered simple lexical features.

### 4.1.2 Complex lexical features

Sentiment score is chosen as it seemed like an interesting way to capture an aspect that is unique to human language, namely emotional value. One of the limitations that was experienced in extracting the sentiment from the documents, is that there were no prior labels for sentiment. This meant that the model would be less accurate in generating sentiment values of texts in the dataset, as it was not trained on this dataset. This would not suffice in a study that solely focused on the sentiment values of the text. However, since the values are simply used as a feature for the final model, it is sufficient since each value just captures part of the information contained within the text. The library that was used for extracting sentiment values is

called HuggingFace (Wolf et al., 2020). This library implements transformers, which are currently the best models for performing tasks that involve natural language, in an easy-to-use way in Python. Since there are no sentiment labels available in either of the datasets, a pre-trained model was used. Since the Shell papers dataset contains documents written only in Dutch, and the Enron dataset contains documents only written in English, two different pre-trained models are used. Each of the models was trained on either specifically the Dutch language or the English language. The model chosen for the Dutch documents is "RobBERT" (Delobelle, 2021), which is based on the BERT model architecture (Devlin, Chang, Lee, & Toutanova, 2019). RobBERT has shown state-of-the-art performance for tasks such as NER and sentiment analysis and therefore it was chosen. For the English documents the "distilibert-base-uncesd-finetuned" model within Huggingface was used, as it provides very good performance without being pre-trained on a dataset at first (Sanh, Debut, Chaumond, & Wolf, 2019).

Bag of words is chosen as a feature since it is a proven way to capture information from a document (H. K. Kim et al., 2017). With bag of words, the entire vocabulary of a set of documents is first extracted. Depending on the number of documents in a dataset, this vocabulary can be very large. For every instance in the dataset, it is determined how many times each word of the vocabulary appears in the instance. This is the basic version of bag of words that can lead to a too-large array for simple models. Therefore, a slightly enhanced and more compact bag of words is used. This enhanced bag of words incorporates TF-IDF and feature hashing.

The final feature that is chosen is based on the idea of NER. With NER, as described in section 2.2.2, the main idea is to "locate and categorize important nouns and proper nouns in a text" (Mohit, 2014, p.221). By using this information as a feature, types of nouns that might only appear in texts part of a storyline or texts that point to company logistics will be captured. To extract this information for the Dutch documents, again the RobBERT (Delobelle, 2021) model is used in combination with HuggingFace (Wolf et al., 2020). This pre-trained model in combination with the HuggingFace transformer architecture provides a relatively easy way of obtaining these values. For the English documents, the most used pre-trained model for NER within Huggingface is used. This model is based on the "xlm-roberta-base" (Conneau et al., 2019).
Both models generate the following markers for named entities:

- **I-MIS**: Miscellaneous entity
- **B-MIS**: Beginning of a miscellaneous entity right after another miscellaneous entity
- **I-Per**: Person's name
- **I-ORG**: Organization
- **B-ORG**: Beginning of an organization right after another organization
- **I-LOC**: Location
- **B-LOC**: Beginning of a location right after another location

An example of a sentence with these markers can be seen below in Fig .7


Figure 7: A sentence with NER markers (Conneau et al., 2019).

The number of occurrences of each of these markers was counted and added as a feature to the dataset.

From the paper by Goel et al. (2010), the number of proper nouns in a text and the percentage of sentences written in passive voice are considered complex features. This is because to calculate these features, first something else needs to be computed, namely whether a sentence is written in passive voice or whether a word is a proper noun.

Note that all the code that was used for extracting the features can be found in appendix D.2.

## 4.2 Modeling

To be able to obtain useful insights from the features generated from the data, AL will be performed in combination with supervised machine learning. How AL is applied and which supervised machine learning models are used along with which querying strategies is explained in this subsection.

### 4.2.1 ASReview

To apply AL to the extracted features, ASReview is used. ASReview, which stands for Active learning for Systematic Reviews, is an open-source non-profit initiative by Rens van de Schoot and Jonathan de Bruin. The initiative was started for creating a tool that can help systemically review scientific articles (van de Schoot et al., 2020). The tool, which is still actively developed, consists of a GUI built on Python that allows easy use of AL on datasets of scientific documents. However, within this research, it is used to train models to classify documents that could either point to a certain storyline or company logistics.

As stated in section 2.5, ASReview includes a wide range of pre-installed supervised machine learning models and feature extraction methods. The supervised machine learning models are mostly based on the models from the Sklearn Python library (Pedregosa et al., 2011), except for the neural network with 2 hidden layers which is based on the Keras library (Chollet, 2015). The feature extraction methods are based on the Gensim library (Rehurek & Sojka, 2011), Keras library (Chollet, 2015) and Huggingface library (Wolf et al., 2020). While these feature extraction methods are useful, they cannot be combined and are rather basic on their own. Therefore, a custom feature extraction method, which consists of the features defined in section 4.1 is used together with a predefined model and querying strategy. The feature extraction method is implemented into ASReview through the use of an extension method.

Not all features are actually generated with the extension method within ASReview for this research. This is since one of the features cannot be used in combination with a holdout set in the way that feature matrices are being generated within ASReview. This specific feature, which is the bag of words feature, is taken over the entire dataset, including the holdout set. The holdout set is a set that is not used in training. More about what a holdout set actually is and why these were deemed necessary can be found in section 4.5. This would mean that after this feature is generated, the part of this feature that belongs to the holdout set would need to be removed manually. While this would have been possible, some settings which ASReview generates from the dataset that is uploaded before the feature extraction prevent the proper functioning of ASReview if this is done. In more practical terms, this means that ASReview crashes. Therefore, another approach was chosen in which the bag of words feature is loaded from a separate file. This separate file is created outside ASReview with the use of Jupyter notebooks. By doing this the source code of ASReview does not need to be altered, meaning that the tool does not crash. All the relevant code through which this feature matrix is generated and the feature extraction method itself can be found both partly on Github[2] and fully in appendix D.2.

The choice of model and querying strategy is further elaborated upon below.

### 4.2.2 Choosing a model and querying strategy

To be able to find a pattern within the data a model is necessary, more specifically a supervised model as AL is performed. Many supervised models exist with wide ranges of applicability for a great variety of uses. There are lesser complex models such as logistic regression and naive bayes and very complex models such as neural networks and random forests. A general rule is, the more complex a model is, the more labeled data will be necessary to properly train the model. Since the assumption of AL is that labeled data is not abundant as all the data needed to be labeled manually, a simpler model is the right choice when starting the AL. Two common simpler models that were considered, were naive bayes and logistic regression. Since naive bayes assumes all features are conditionally independent, logistic regression is chosen.

With logistic regression, the model is trained in such a way that if the probability of the event happening exceeds a certain value, the outputted label is 1. If the probability is lower than this threshold, the label is always 0. The inputs of the logistic regression model are mostly matrices, except for the length of the document. To each of these matrices, a weight is applied and a bias is finally added to generate the final

[2]https://github.com/michagast/asreview_ftm

output. The mathematical representation of logistic regression is as follows (Kleinbaum, Klein, & Pryor, 2010):

$$f(z) = \frac{1}{1 + e^{-(\alpha + \sum \beta_i X_i)}} \tag{8}$$

In this equation $\alpha$ represents the bias, $\beta$ represents the weight and $X$ represents an input feature. Furthermore, $f$ represents the equation in full and $z$ is the output of the equation.

After the AL is performed, a labeled dataset is obtained. Since all instances are now labeled, more complex models can be utilized. Models that will be considered for this are recurrent neural networks and transformers, as they are considered the most appropriate for NLP tasks (Ghosh et al., 2016). By using this approach, the most value will be gotten from the entire AL process and the best possible final classification model will be obtained.

To determine which querying strategy should be used a test run will be performed with the Enron dataset. In this test run, which can be found in section 5, other models besides logistic regression and the influence of oversampling will also be tested.

### 4.2.3 Active Learning

To be able to start the AL process, a small slice of the data needs to be labeled first. This is done by an expert on the dataset if these labels are unavailable. Of all the labeled data, a baseline of 70% is used for performing AL and the remaining 30% is used for testing the model that results from the AL. This could be changed depending on how many preliminary labels are available and how much time is available for performing the AL.

To perform the AL part, ASReview is used. First, the data is loaded into ASReview. Next, the custom feature extraction method is chosen, along with the querying. Next, the model is trained on the labeled instances, and after it is trained the querying starts. For this to first happen, the trained model predicts a label on all queried instances, and then by using the selection criteria of this metric, an instance is picked out. This instance was then presented to the human oracle, after which it is labeled and added to the labeled set. This continues until an accuracy that is the same as the passively trained model is achieved, no data is left to label or time has run out. A visual guide on how to use ASReview in any AL project can be found on the ASReview website[3].

## 4.3 Feature selection

Since all the features described in section 4.1 were based on literature and not on how well they work with the datasets used in this research, a round of feature selection was also performed. In this section, the feature selection will be discussed in the order that it was performed in the research. All feature selection was performed with a logistic regression model, which was chosen as the to-be-used model in section 4.2.2.

### 4.3.1 Enron

As discussed in section 4.1, both simple and complex lexical features were extracted from the text. To see which features contained the most information, histograms were created that contained the average value of each feature for both the positive and negative labels. Next to this, coefficients were collected from the previously discussed logistic regression model, which contained all features.

The first feature that was chosen was the length of the document, which is a simple lexical feature. The average value for the length per label can be found in Figure 8.

[3]https //asreview.readthedocs.io/en/latest/project_create.html

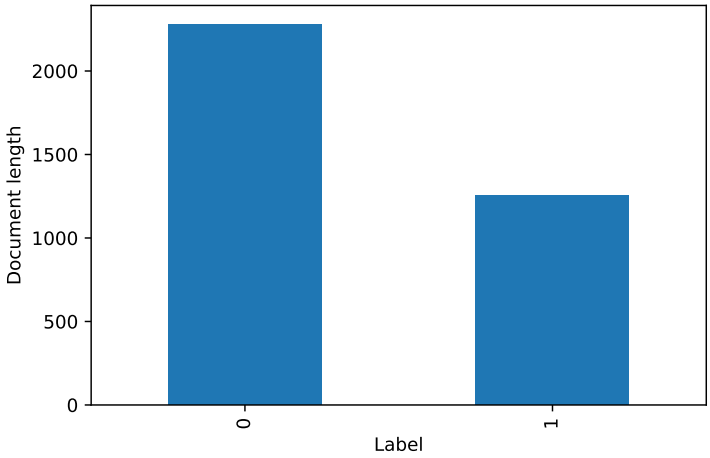



Figure 8: Mean value of the length per document for the logistics label.

The length of the document was chosen as a feature since there is a large difference of almost 1 000 between labels for the mean value of length. While the coefficient for this label is rather low at -0.000454 when it is used the results of the model significantly improve.

The second feature that was chosen is the percentage of passive voice sentences within a document, which is a complex lexical feature. The average value for this feature per label can be found in Figure 9.

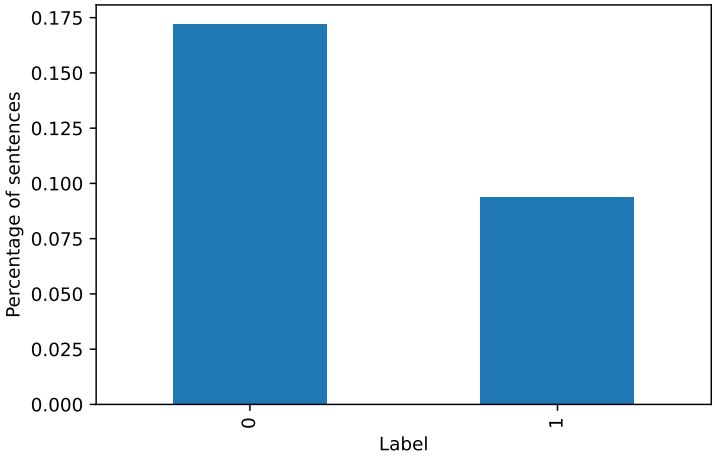



Figure 9: Mean value of the percentage of sentences written in passive voice in a document for the logistics label.

While the absolute difference between the labels for this feature is much smaller at just 0.07, the relative difference is still clearly visible since this feature always has a value between zero and one. Moreover, the coefficient of this feature is the second highest when not looking at the bag of words coefficients at -0.8. Therefore, it is considered an effective feature for finding the positive cases of the logistics label.

The third feature that was chosen from the complete feature set was the type-token ratio of a document. The average value for this feature per label can be found in Figure 10.

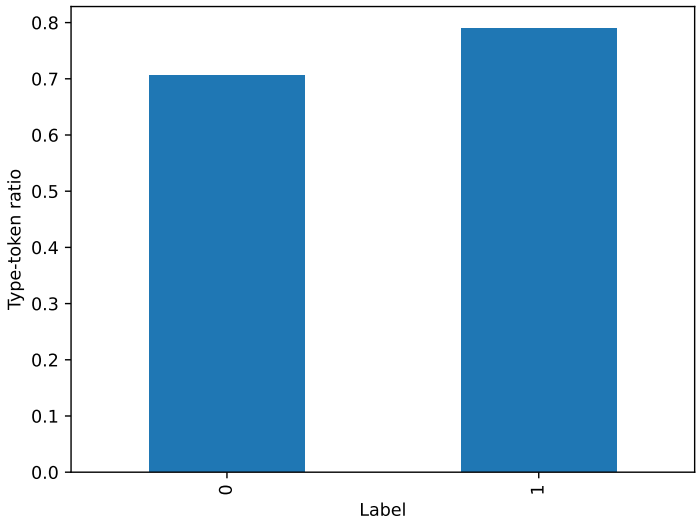



Figure 10: Mean value of the type-token ratio found in a document for the logistics label

While the bar chart of this feature is not impressive, its coefficient is. This feature has the highest of all non-bag of words coefficients at 1.42. While the bar chart shows a very minor difference in mean average value between the two labels, its coefficient is in the end a more valuable piece of information.

The final feature that was chosen was the bag of words feature. While it is not useful to plot the values of the exact bag of words in the way that the previous features have been, it is possible to instead look at a part of it that can be visualized easily. These are the TF-IDF scores.

Table 1: The ten terms with the highest TF-IDF values for the Enron documents that contained information about logistics(left) and the documents that did not contain information about logistics(right).

| **Label: 1** | |
|---|---|
| **Term** | **TF-IDF Value** |
| photograph | 5.584 |
| cancel | 4.117 |
| stauffach | 3.341 |
| rsvpd | 3.341 |
| tel | 3.248 |
| speak | 2.731 |
| pritchard | 2.227 |
| reilli | 2.227 |
| krth | 2.227 |
| moodi | 2.227 |

| **Label: 0** | |
|---|---|
| **Term** | **TF-IDF Value** |
| koothrapp | 3.341 |
| www.vaionlineit | 3.341 |
| wwwssrncom | 3.341 |
| love | 1.910 |
| testimoni | 1.882 |
| fine | 1.742 |
| randi | 1.659 |
| joseph | 1.649 |
| wwwstanfordedu | 1.630 |
| wwwhousingfinanceorg | 1.569 |

In Table 1 the terms with the highest TF-IDF scores are shown for both labels. Note that these are the highest scores among all documents and not a single document. What becomes apparent immediately is that not a single word in either top 10 matches, and neither do the values. This shows that the words that are most unique to each label are very different.

Table 2: The TF-IDF values for each label for the top 10 words of the other label.

| Label: 0 | | Label: 1 | |
|---|---|---|---|
| **Term** | **TF-IDF Value** | **Term** | **TF-IDF Value** |
| photograph | 0.005 | koothrapp | NA |
| cancel | 0.061 | wwwvaionlineit | NA |
| stauffach | NA | wwwssrncom | NA |
| rsvpd | NA | love | 0.039 |
| tel | 0.098 | testimoni | 0.072 |
| speak | NA | fine | 0.043 |
| pritchard | NA | randi | 0.055 |
| reilli | NA | joseph | 0.168 |
| krth | NA | wwwstanfordedu | 0.349 |
| moodi | NA | wwwhousingfinanceorg | NA |

In Table 2 slightly different information can be found as opposed to that found in Table 1. In this table, the top 10 terms for each label are shown, but now the TF-IDF values of the opposite label are shown. If no value was available, an NA was put in. Since no value is close to the values shown in Table 1, it can be concluded that TF-IDF values contain much information. The difference in value for the words per label will mean that the model will most likely be able to pick up on a certain label easier. Knowing these values and the research done and the widespread use of bag of words, it can be assumed that the bag of words feature is useful. The size of the final bag of words array is 1001x1.

All the features combined consisted of an array of 1 004 numbers per email.

### 4.3.2 Shell papers

With the Shell papers dataset, the feature selection was less extensive since the labels for this dataset were received late. Instead of having the time to take a look at the histograms for every separate feature, only the feature coefficients were used, which can be found in Table 3. The feature coefficients were generated by using the small subset of labeled data that was obtained from FTM. The same settings for the model were used as with the Enron feature selection. Three things need to be noted here. First, some libraries that were used for features did not have an option for Dutch. This was the case for the feature that contained the number of proper nouns within a text. While a custom library could have been written, this was deemed not necessary. Second of all, the field of NLP is very revolved around the English language. This means that even if models exist that work for Dutch, it is highly unlikely that they will be of the same quality as their English counterparts. Third and finally, since the bag of words is a really large feature, it is taken out of Table 3, but do note that its coefficients are far ahead in terms of value than those of the rest. Therefore it was selected. With this being said, the coefficients for the other features are as follows:

Table 3: The feature coefficient scores for the preliminary labels of Shell papers dataset

| **Coefficient** | **Feature name** |
|---|---|
| 0.1359 | Sentiment value |
| 0.0029 | Standard deviation of sentence length |
| 0.00002 | Text length |
| -0.0100 | Amount of proper nouns |
| -0.0120 | Readability index |
| -0.1449 | Standard deviation of word length |
| -0.2697 | Type-token ratio |

The comparison in Table 3 shows that the features sentiment value, the standard deviation of word length and type-token ratio are the most important besides bag-of-words. Therefore, these features are also selected. Besides these three features, text length, even though it has the lowest feature coefficient, was chosen. Since it was shown to increase performance substantially. Also, note that the features “presence of specific words”

and NER are not in Table 3 since they were already quite explicitly present within the bag words feature. Besides, compared to the NER feature, the bag of words feature is much easier to generate.

All the features combined again consist of an array of 1 004 numbers per document.

## 4.4 Evaluation

While a general evaluation method has been proposed by (Kottke, Calma, Huseljic, Krempl, & Sick, 2017), this method is more focused on evaluating multiple AL methods with each other. This is not the goal of this research, but the evaluation method and the rest of the paper do have two useful pointers on how to evaluate AL models properly. First, learning curves should be shown to see if there is a noticeable difference between the passive and AL models. Second, a clearly defined stopping criterion should be used. Or enough samples should be drawn for the model to converge. A learning curve comparison, in this case, can be made by seeing how accuracy improves in both passive and AL scenarios. The stopping criterion, in this case, is clearly defined in section 4.2.3.

### 4.4.1 Performing the evaluation

To evaluate the generated model in this case, it will be used to predict new instances, as the evaluation is focused on determining if the model can generalize well over new instances. These instances need to be labeled because without a ground truth performance metrics can not be computed. The performance metrics used in this research are accuracy and F1-score. Accuracy is chosen as it is a general measure for quantifying model performance. F1-score is chosen since the labels in this research are unbalanced. Learning curves are also used when testing and evaluating the method on the Enron dataset and when evaluating the method on the Shell dataset if enough instances are labeled manually. Area Under the Curve (AUC) could have also been an interesting metric. However, since all instances would need to be manually labeled for this metric, this is not feasible and the metric is not used on the Shell data. While during the Enron dataset test all labels will be available, the learning curve of the F1-score will be used instead here, since this shows well if the minority label is discovered properly by the model.

The more technical description of the way that the evaluation is performed with the data is as follows. Every iteration, iteration meaning the labeling of an instance, a trio of variables will be stored locally. These variables are the indices of the train set, the currently trained model and the entire feature matrix. The indices are used to remove the training data from the full feature matrix. The full feature matrix is extracted from ASReview since this saves the time of having to manually generate all features within a Jupyter notebook. While this is done every iteration for the simplicity of the code, it is not necessary to do this. As the feature matrix gets generated before performing any model training/querying, meaning it won't change once the actual AL has started. If storage space is scarce, one could opt for slightly more complex code to save the feature matrix just once. Going back to the evaluation process, by using the saved model from each iteration, the accuracy, precision, recall and F1-score are calculated for each test set. The test set is determined by removing the training indices from the full feature matrix. These scores are calculated for each iteration, which are then used to generate graphs. All the code for saving the variables from ASReview can be found in appendix D and the code for the creation of the graphs can be found in appendix D.3.

## 4.5 Holdout sets

In this research, the test set is defined as the set which is used in ASReview but not yet for AL. This means that the test set constantly changes. To make sure the evaluation is not affected by an unrealistic test set, a holdout set will also be used to perform the evaluation on. A holdout set is a subset of data that is not used when training or testing the model. This ensures that it never changes in terms of the data that is in it. When creating a holdout set the following should be taken in mind, If one knows the underlying distribution of labels present within a dataset, then the holdout set could reflect this. If this distribution is unknown, random sampling might work best.

### 4.5.1 Performing the evaluation on the holdout set

From a technical perspective, the following changes had to be made to the process described in the previous section to achieve a visualization of the results of the method on the holdout set. Feature matrices were created outside ASReview as these could not be retrieved from ASReview. Both matrices use the same model variables.

# 5 Configurations

Since the actual execution of this research can only be performed once, the method of this research was tested on the Enron emails dataset. A part of this dataset was labeled on specific attributes regarding the textual context. Therefore it is a good way to evaluate and test the performance of the method that is later applied to the dataset from Follow the Money. The label that the method is tested on, is whether an email contains information about company logistics. This label is unbalanced with roughly 1/3 of the samples belonging to the positive class.

The testing of the method went as follows. First, a feature set was created that had a good distinction between the positive and negative class of the specific label. Second, this feature set was tested with a wide range of configurations that could be found within ASReview. Finally, results were gathered and evaluated to see what did and did not work. The second and third step are expanded upon in the following sections.

## 5.1 Possibilities within ASReview

As discussed before, the tool that is used for performing the actual Active Learning (AL) within this research is ASReview. Within ASReview, many options exist for querying, feature extraction, modeling and sampling for training the model. Since in this research a feature extraction method was designed specifically for the goal of classifying documents on specific attributes, the built-in feature extractors are ignored. The model that is used primarily in this research is logistic regression. However, other models that come with ASReview will also be tested. This leaves three possible querying strategies, three sampling techniques and seven models to choose from for a total of 126 possible configurations. Three querying strategies are selected in terms of AL: uncertainty query, maximum query and cluster query. Uncertainty query has the priority, as it has been covered in other papers (Kumar & Gupta, 2020), however maximum query and cluster query will also be tested just to see if they give better or worse results. Maximum query tries to find the instance of which the model is most certain it belongs to the positive class. With cluster query, K-means-clustering is first performed using the available features. Then the highest probabilities, or the instances which are most likely to belong to the positive class, within random clusters are sampled. The three sampling techniques that ASReview implemented are double resampling, normal sampling and undersampling. With double resampling, oversampling is performed, meaning that if the positive label is underrepresented in the dataset some positive instances get sampled multiple times to get an even amount of positive and negative instances to train the model on. Undersampling tries to achieve the same thing but then by leaving out negative instances. The 7 possible models have been discussed in section 2.5. Of these models, 5 were tested and 4 generated usable results. Models were excluded if they took too long to run or were built for other purposes than the one in this research.

Almost all of the remaining previous configurations were tested. The most interesting ones are discussed in the next section. These configurations were deemed the most interesting as they were either the best performing or most complex. The standard configuration of ASReview is also included.

## 5.2 Configuration test results

This section contains three figures. In each figure, AL performance is compared to Passive Learning (PL) performance. In each graph, the AL performance is abbreviated with AL and the PL performance is abbreviated with PL. All configurations were kickstarted with 15 labeled instances of which 10 were negative and 5 were positive.

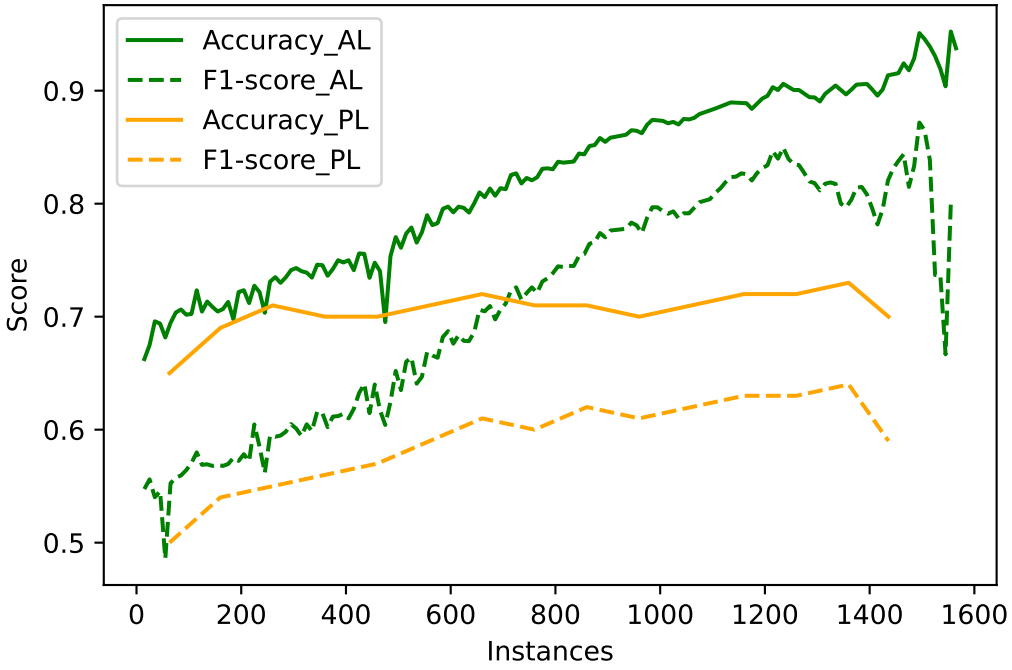



Figure 11: AL and PL using logistic regression with L2 regularization, uncertainty query and double resampling.

When looking at Figure 11, it becomes clear that the AL is performing almost as expected. In the papers by Das Bhattacharjee et al. (2017) and Lafferty et al. (2001) a 70-90% decrease in the number of labels necessary for achieving the same scores as PL was observed and this is nearly seen in this figure. While the maximum accuracy is achieved within these percentages, it is important to consider that this is unbalanced data. Therefore the F1-score also needs to be closely examined. The maximum F1-score of PL is at around 1380 samples. This score is achieved at around 500 samples in the AL scenario. While this is not precisely the reduction of 70%, it comes very close. What is furthermore interesting to see is that the final accuracy achieved with AL is almost 20% higher than that of PL.

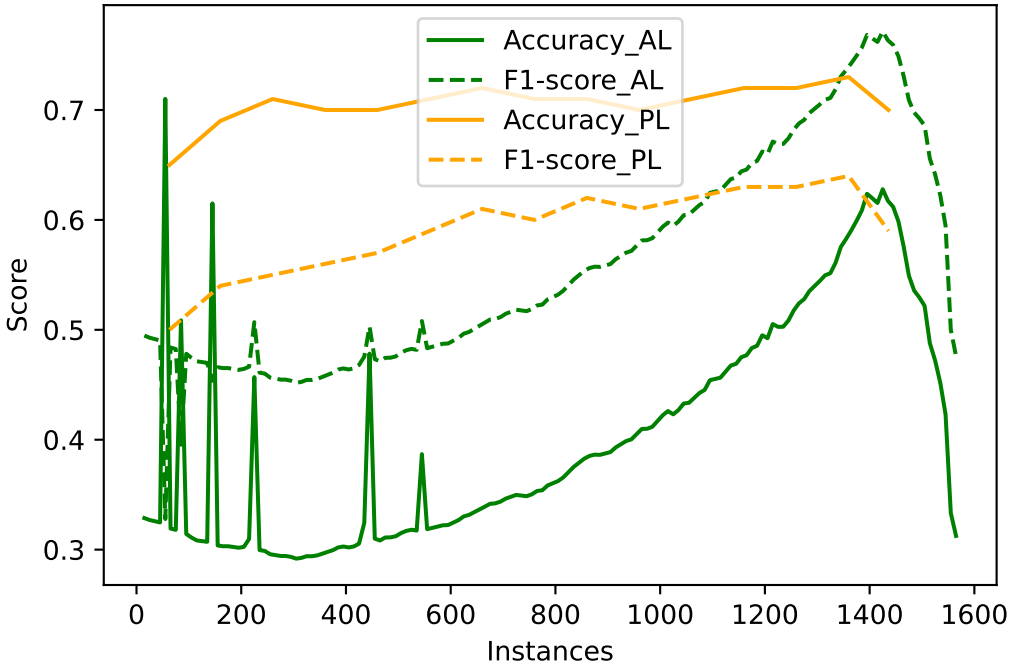



Figure 12: AL using a neural network with two dense layers, uncertainty query and double resampling and PL using logistic regression with L2 regularization.

When looking at Figure 12, worse results compared to logistic regression are found. In this test, a neural network with two dense layers that ASReview comes with was used. Note that the PL results are still generated from a logistic regression model. A much more complex model compared to logistic regression was used. Since complex models need more data to reach the same results as simpler models, the model performs much worse with fewer instances. In the end, the model manages to reach a performance that is a little bit better than standard PL. However, this performance increase is much less compared to that which is found with logistic regression.

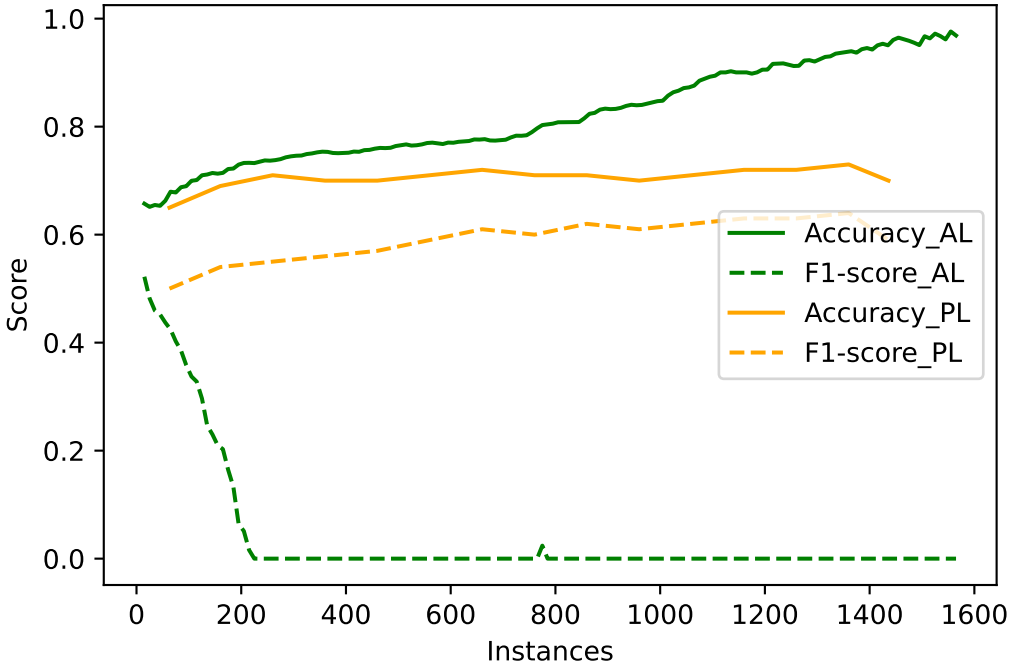



Figure 13: AL and PL using logistic regression with L2 regularization, maximum query and normal sampling.

Finally, when looking at Figure 13, the worst configuration is found. In this test, the default query setting from ASReview was used, along with normal sampling and the custom feature extractor. While this query setting works great for academic systematic reviews, it is incredibly bad at generating a model that is good at generalizing over multiple instances. While the accuracy is decent, the F1-scores are very bad. This could be due to maximum query finding all the easy-to-identify positive cases quickly and then leaving the hard-to-identify cases in the test set.

To conclude, the best configuration after multiple tests turned out to be a combination of logistic regression with uncertainty query and double resampling. This is also the combination that will be used when applying the method to the Shell data.

# 6 Results

In this section, the results of the Active Learning (AL) run on the Shell papers, including a simulation using the preliminary labels, will be discussed. In this section, AL_HO stands for Active Learning Holdout meaning the results come from the Active Learning model on the holdout set. PL_HO stands for Passive Learning Holdout.

## 6.1 Results of simulation run

As specified in section 2.1 a small amount of labeled data is necessary to perform AL. This small part of labeled data was used to perform a simulation run within ASReview. The results of this simulation run can be seen in Figure 14.

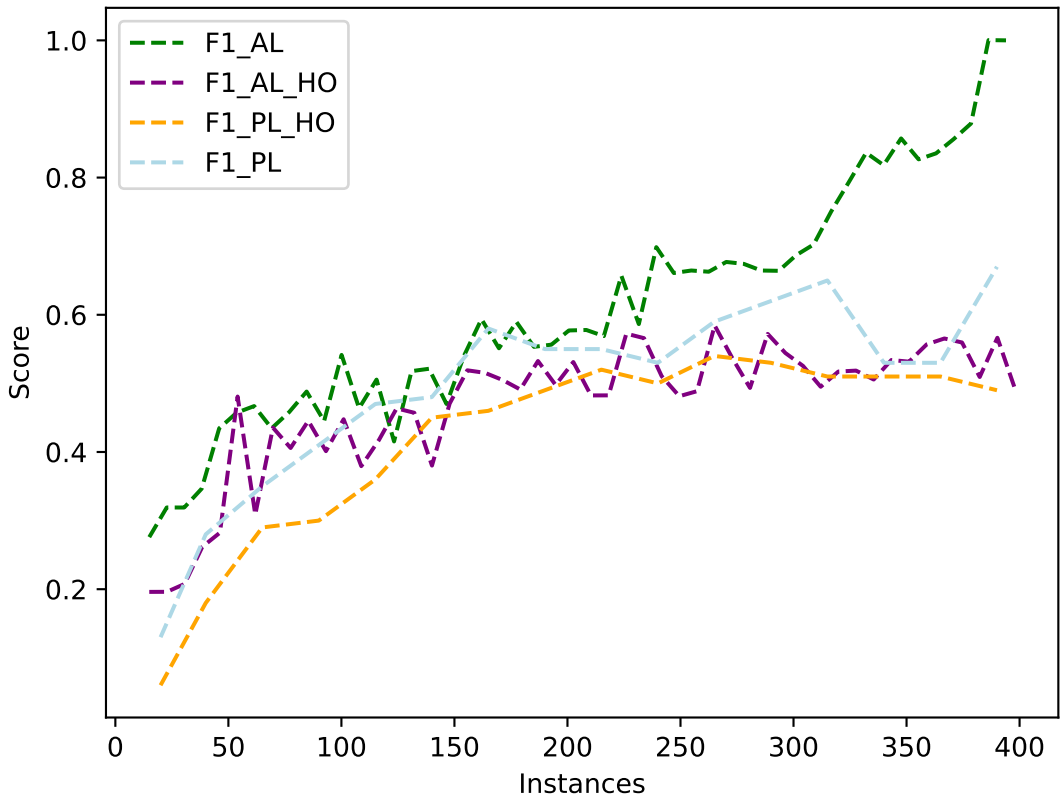



Figure 14: Comparison of F1-scores of both Active and Passive Learning on the preliminary Shell papers labels

In this image, the F1-scores of the AL and Passive Learning (PL) are compared on both the train set and the holdout set. This holdout set contained 200 instances. The green and blue lines in the image represent the F1-scores of the test sets of both the Active and PL. What becomes clear when looking at these lines is that the AL reaches a much higher F1-score than the PL. This is in line with what was found in the previous section on configurations. Also interesting in this figure are the lines for the holdout sets. The purple line, which is the F1-score for the AL holdout set, starts a little higher than the orange line which is the F1-score for the PL holdout set. This is due to the fact that the starting sets were not exactly the same. The purple line seems a lot noisier, but it has more data points than the orange line. As the data points from the orange line had to be gathered manually since no script was written to calculate the numbers per iteration. Besides

this, the orange line stays a bit lower than the purple line and mostly in the end the orange line seems to go down more than the purple line. The question is of course, where will the F1-score and accuracy for the holdout set go?

## 6.2 Real Active Learning run Shell data

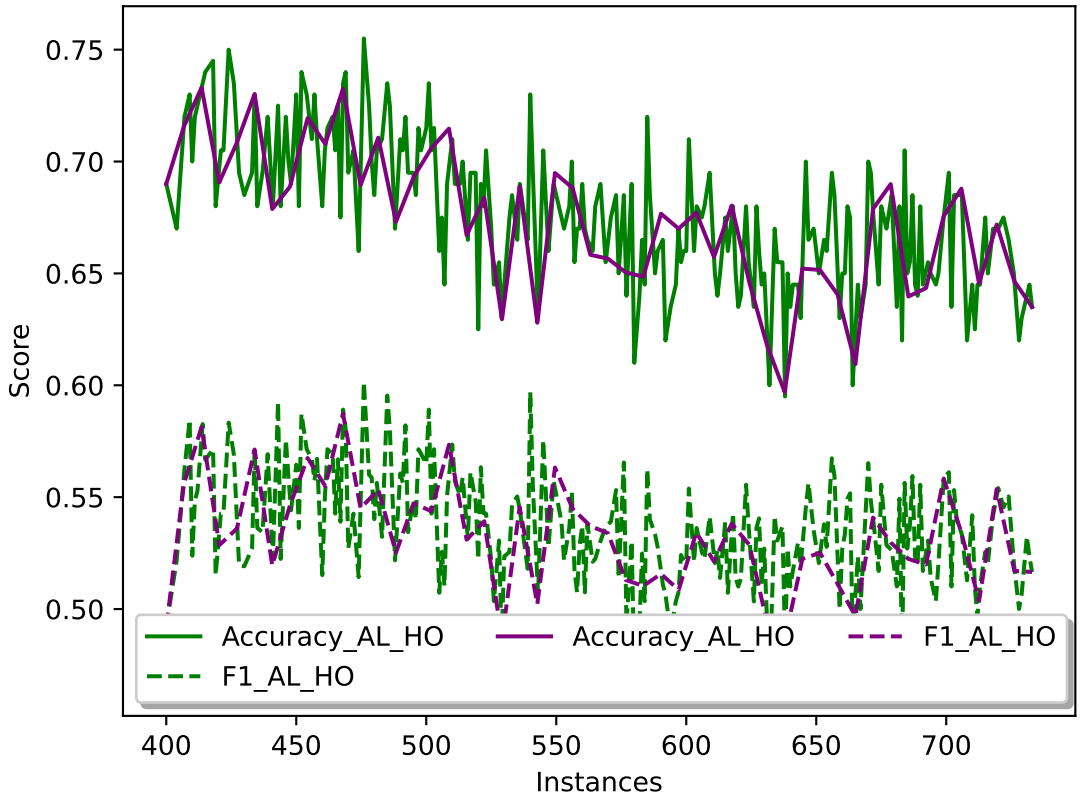



Figure 15: The accuracy and F1-score on the holdout set for the AL run on the Shell papers data.

In Figure 15 the answer to the previous question can be found. This image contains the result of the AL run with FTM in which in total 333 instances were labeled manually by the oracle. The same holdout set of Figure 14 is used to display the results. While it is not a direct continuation of the graph shown in Figure 14, since in Figure 15 the model was completely trained in an AL way and this model was passively trained for the first 400 instances, it is still very relevant. To get to this image, 399 instances were used to begin the AL. This is also why the graph starts at 400 since this is the start of the first actively learned labeled instance. The purple line represents the exact same data as the green line, but instead of featuring all data points, it features about 20% of the number of data points in the green line. The line goes up and down a bit, with a maximum accuracy achieved of 75.5% with an F1-score of 0.6 after 76 instances of AL. The ending result of the AL, which in this experiment happened after 333 instances, is lower than the starting point with an accuracy of 64% and an F1-score of 0.52.

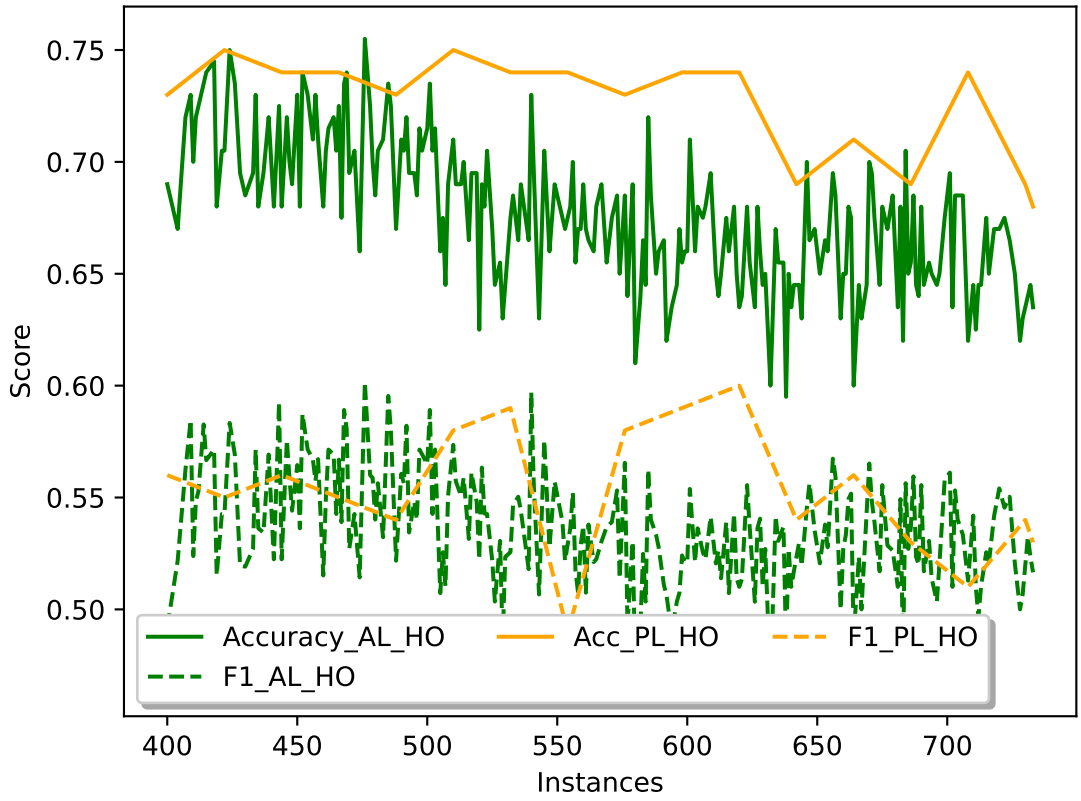



Figure 16: The accuracy and F1-score on the holdout set for the AL run on the Shell papers data together with the performance of PL.

Finally, in Figure 16 the PL performance is compared to the AL performance. While the accuracy of PL is a small bit higher, the F1-score is around the same level and often even lower than that of AL. Interestingly, the highest point of accuracy belongs to AL and the highest point for F1-score belongs to PL. For a further discussion of the results found in Figure 15 and Figure 16 please refer to section 7.1 of the discussion.

As specified in the method, with the now obtained model the rest of the dataset can be labeled. Using this fully labeled dataset a final logistic regression and neural network can be trained. The results of this can be seen in Table 4.

Table 4: The results of a logistic regression and neural network model trained on the fully labeled dataset.

| | Precision | Recall | F1-score | Accuracy |
|---|---|---|---|---|
| Logisic regression | 0.83 | 0.92 | 0.87 | 0.91 |
| Neural network | 0.59 | 0.38 | 0.46 | 0.68 |

For the models, the data was normalized to a 0-1 scale since the neural network required this. This resulted in an impressive accuracy score of 91% for the logistic regression model and an accuracy of only 68% for the neural network. When looking at the F1-score, the logistic regression model is almost twice as high. Note that the precision and recall are those of the positive label. A more extensive version of the table can be found in Appendix B.

# 7 Discussion

This discussion is structured as follows. First, an in-depth analysis of the results and what they exactly mean is given. After this, six technical subthemes are discussed revolving around the execution of this research and what decisions were made and what could have been done differently. After this, two non-technical subthemes are discussed which end with what the general implications of this research are regarding the real world and previous Active Learning (AL) research.

## 7.1 Discussion of results

In this research, the AL process using ASReview was first tested on the Enron data. In this test, it turned out that a combination of logistic regression together with uncertainty query was the best configuration that ASReview at this time had to offer for performing AL to train a classifier on detecting emails that were about logistics. It was interesting that this turned out to be the best option, since it was completely different from the ASReview default settings. This configuration was then applied with a Dutch variant of the feature-set on the Shell papers, since the Shell papers documents were written in Dutch. More about the concessions and consequences of using the method on a different language can be found in section 7.7. In the simulation run that was first performed with 400 instances in the train set and 200 in the holdout set, it seemed that the AL worked as was expected when compared to the other research executed on the topic and the Enron test run. In these simulations, AL usually learned at a more efficient manner than Passive Learning (PL). Besides simulations on the Enron and Shell datasets, the method was also evaluated on a dataset on Dutch news. The results of this simulation are roughly the same as those of the other simulations and can be found in Appendix A.

In Figure 15 a different result was found however. Instead of steadily going up, mostly the accuracy seemed to steadily go down. The fall of the F1-score was less rampant, but this also ended lower than it started. What could be the explanation for these results?

First, only a very small percentage of the data was actually labeled within the AL run. At the end of the run, only 5.4% of the total dataset had been labeled. This was much less than the 10-30% of data that was deemed necessary, as found by other researchers (Das Bhattacharjee et al., 2017; Nguyen & Patrick, 2014). It could therefore be that the spike in AL performance is going to happen much later than this graph. Second of all, it could be that the holdout set represented the subset of the 400 labeled instances very well, but not the other part of the dataset. Therefore the usage of the holdout set as a way to determine the performance of the AL model could have been incorrect. Third and finally, while unlikely, the labels given by the expert could have been wrong. The expert could have been distracted by the functioning of the tool or could have looked at the wrong things. More about this can be read in section 7.9.

When comparing the AL to PL, of which the graph can be seen in Figure 16, it turned out that the PL was a little more accurate while the AL was able to distinguish the minority label better. It is always contextually dependent which is more important, but since this is a problem where the dataset is probably very unbalanced, F1-score most likely matters more. Therefore, one could say that the AL outperformed the PL.

Finally, the completely labeled dataset was used to train both a logistic regression model and a neural network. The logistic regression model outperformed the neural network by a lot. This could be due to the fact that the neural network did not have the correct settings, as neural networks allow for almost infinite fine-tuning. It might also be that the neural network simply needs more data to reach a high accuracy and F1-score. However, considering the difference in performance it might be that logistic regression is better in this use case. The enormous difference in logistic regression performance between the final model from the fully labeled Shell dataset and the simulation can be attributed to the normalization of the features. As the logistic regression model might have overreacted to some extreme values in the length feature. All in all, the final logistic regression model could possibly be used in practice.

## 7.2 Feature reduction

As stated in the previous section, the model that was used in this research was logistic regression. With logistic regression, one can also apply regularization to the feature coefficients that are computed. Regularization essentially makes the model less sensitive to the features that are inputted, by sacrificing a higher bias

for a lower variance. This is very effective at preventing overfitting. For logistic regression, three types of regularization exist, but only 2 work with the standard solver. These are L1 and L2 regularization. Both were tested and l2 regularization turned out to be slightly better. L2 regression, or ridge regression, is a way of punishing higher coefficients. The way this is done is as follows:

$$J(w) = \frac{1}{m}\sum_{j=1}^{m} Cost(h(x^{(i)}), y^{(i)}) + \frac{\lambda}{2m}\sum_{j=1}^{n} w2_j \tag{9}$$

In equation 9 the cost-function of logistic regression with an L2-penalty is shown. The cost function can essentially be seen as the difference between the predicted label and the actual label. In the equation, $Cost(h(x^{(i)}), y^{(i)})$ is this difference for instance $x^{(i)}$ with ground truth label $y^{(i)}$. Since with logistic regression, a label of 1 or 0 is outputted depending on if the result of the logistic regression is higher than some threshold, this cost function will hold the value of the difference between the true label and the precise outcome of logistic regression. Since the best prediction might lead to very high coefficients, the L2-penalty is added. This is the latter part of the equation that gives a penalty depending on how high the coefficient was. In the end, this leads to a worse prediction on the train set but a better prediction on the test set.

Besides leading to a better performance, the use of a penalty also results in an implicit feature selection. For example, in the case of the Shell papers dataset with all features, when not applying this penalty, the amount of coefficients that are less than 0.01 from 0 is 24. When the penalty is applied, this amount rises to 98. The feature importance of the non-bag of words features also changes:

Table 5: The feature values with no L2 regularization(top) and L2 regularization(bottom) for the Shell data.

**No L2**

| Coefficient | Feature |
|---|---|
| -0.004312 | Proper nouns |
| -0.001143 | Readability index |
| -0.000125 | Standard deviation of word lengths |
| 0.000012 | Length |
| 0.000073 | Sentiment score |
| 0.000105 | Type-token ratio |
| 0.000325 | Standard deviation of sentence lengths |

**L2**

| Coefficient | Feature |
|---|---|
| -0.215934 | Standard deviation of word lengths |
| 0.002189 | Proper nouns |
| 0.0000002 | Length |
| 0.000241 | Standard deviation of sentence lengths |
| 0.001815 | Readability index |
| 0.224586 | Type-token ratio |
| 0.426069 | Sentiment score |

Table 5 shows the previously stated implicit feature selection that is performed with the use of a penalty, in this case L2-regularization. Apparently, the top 4 features obtained with L2-regularization were different from the top 4 features without L2-regularization. If l1 regularization were to be used, this feature importance order would again be different. In the end, it is best to look at which type of regularization gives the best model performance, which in this research is equivalent to which model has the highest accuracy and F1-score.

## 7.3 Influence of Active Learning on model learning

Besides the influence of the regularization within logistic regression on the learning process, on a higher level, there is also an implicit influence caused by the AL itself. Since the AL process includes having the model select the data from which it learns, and this type of generalization does not happen with PL, this could be

an influence on the model learning. This influence could be seen as negative, as it decreases the explainability of the method. As it might be unknown why a certain document could be complicated for the model, which would be the case when using uncertainty query. As explainability is something that should always be strived for in machine learning, this is a downside of the premise of AL. However, this research and other research (Das Bhattacharjee et al., 2017; Nguyen & Patrick, 2014; Tran et al., 2017; Waltl et al., 2017) has shown that this implicit influence on the model learning is a good thing. As none of these papers and this research would have been able to report such positive results of the effects of AL if it was not. Therefore in the context of this research and the other mentioned papers, this seemingly negative influence is actually seen as positive.

## 7.4 Consequences of splitting data

As specified in section 3.4.1, during the preprocessing of the Shell data multiple documents that contained more than one email or document were split. This was done mostly due to the fact that these documents were far above the average length of all documents. The splitting of these documents has multiple positive and negative consequences. These will be discussed in this subsection of the discussion.

### 7.4.1 Positive consequences

The biggest positive consequence is that splitting the documents made the number of prior labels higher, which lead to a better start of the AL process. This also allowed the researcher to run a more valuable simulation and perform a premature feature selection. With the number of labels that were originally in the dataset, both of these tasks would have been either unreliable in results or too hard to execute. Without this premature feature selection, another decision would have had to be made regarding the to-be-used features. A few possible options could have been chosen from:

- A first option could have been that the same feature selection method as the one used currently was utilized. As said before, this probably would not have been the best option in this case.
- A second option could have been that no feature selection takes place and that all features described in section 4.1 were used. While this is not an ideal option either, it is probably better than performing feature selection with very few labels. As having very little labeled data will result in a very biased selection of features.
- A third and final option could be to have the oracle label extra data first before performing the AL. While this option will most likely yield the best results, it is also the most unlikely to happen in a real-world scenario. Since the time of an oracle is rather scarce and one would better be focused on having as much AL executed as to having the best possible hyperparameters. Note that of course an endless amount of options exists for feature selection, but it is out of the scope of this research to discuss every single one of them.

Another positive consequence of splitting the data is that, theoretically, the oracle should have less difficulty in determining whether a document belongs to a certain label or not. This is because the document should be less cluttered with unnecessary information that could otherwise distract the oracle.

A third and final positive consequence of splitting the data is that the model might be trained better at detecting documents adhering to specific labels. Since, just like the last point, a document is less cluttered and therefore a model should be able to pinpoint better which specific parts of a document lead to a specific label.

### 7.4.2 Negative consequences

Besides positive consequences, there is also a negative consequence to splitting large documents into smaller documents. This consequence was that some documents would either have become so small or would only contain irrelevant information that it was practically impossible to label them properly. This consequence was also one of the bigger concerns during the FTM AL run, as both the expert, researcher and supervisors were afraid that splitting the documents would result in unwanted situations regarding the labeling. In the end, it was decided for this consequence that the benefits of splitting the data outweighed the drawbacks. What one could do to combat the creation of irrelevant documents during the splitting of the documents, is to eliminate documents that are very short, since it may be unlikely that these will contain useful information.

## 7.5 Multi-label classifiers

One of the topics that has not been discussed at all so far in this research, but is very useful in many occasions regarding document analysis, is multi-label classification. Up to this point, all related work and classification scenarios were focused on binary label classification. Binary label classification, as previously discussed, is only concerned with a single label that either has a zero or one (or true or false) as possible values. All labels used within this research, are binary labels. Multi-label or non-binary classification is different from this. Instead of just saying whether a document belongs to a storyline or an email is about logistics, the classifier will now still give just a single output but more than two outputs will be possible. The following paragraphs explain what multi-label classifiers could mean for the data analyses used in this research.

### 7.5.1 Consequences of using a multi-label classifier

In the case of the Shell papers dataset, a label was chosen for this research about which the expert had enough confidence that he would be able to label documents in a timely enough manner that would make running an AL run feasible. However, more than one storyline exists within the Shell papers dataset. And since the model that was trained with the AL in this scenario would only be able to detect whether a document belongs to the storyline it was trained on, it would mean that an AL run would have to be executed for every single known storyline which exists within the documents. If instead a classifier would be used that is capable of outputting more than two labels like a neural network, a lot of time could be saved in the bigger scheme of things. Since now, only a single AL run would be required to generate a classifier that can recognize all known storylines. As this almost sounds too good to be true, there are a few downsides to this approach.

The first downside is that the model inherently is going to be more complex than a model that can only output a binary labeling. Using a more complex model can lead to unwanted things. For example, the explainability of the model becomes worse, meaning that our model becomes more of a black box than just a simple decision-making algorithm. This is something that, if possible, should always be avoided. Besides losing explainability, a complex model is almost always going to need more data to reach a steady accuracy compared to that of a simpler model. While in the grand scheme of things less data is also necessary, since fewer AL runs need to be performed to reach the same outcome, it is really choosing between two types of bad in this case. Finally, besides making the model itself more complex, increasing the complexity of a model also leads to a more complex evaluation. Since inherently the results that come out of the model are dependent on more factors, meaning that more variables need to be accounted for in the evaluation.

A second downside is concerned with the oracle. In the setup with a binary label classifier, the oracle simply has to determine whether a document belongs to a single category or not. With a non-binary label classifier, however, the oracle will now have to make a much more specific decision that could be tough. Having to make a tougher decision could lead to more bias in labeling and/or a potentially worse labeling.

A final downside to using a more complex model has to do with ASReview itself. If it were possible with the current tool, the researcher would have opted for a multi-label classifier. However, in its current form, ASReview does not offer this functionality. This is because the purpose that ASReview was built for, namely to accelerate the systematic reviews of academic documents, does not require multi-labeling. While it should certainly be possible on the technical side, it will require a lot more complexity from the tool, as there need will need to be more possibilities for making a decision about a document. If there would be 50 possibilities for a document label, one could imagine that either 50 buttons or a dropdown list with 50 items would need to be added to give the oracle the opportunity to assign any possible label within the scope. Instead of adding this many buttons, there could also be opted for open questions that could treat the decision of deciding on a certain label as a decision tree problem. This has been researched before by R. Hu (2011) and has been deemed to be effective. This would add another layer of complexity to ASReview and the entire AL process in general, which could be unwanted.

## 7.6 Other querying methods

Besides multi-label classifiers, there is another part of the AL process that is broader than how it was applied in this research. This part of the process is concerned with querying. As stated in the related works section, there exist many ways to perform this task within AL. To refresh the memory a bit, the task of querying consists of selecting an instance of the unlabeled pool that has just been labeled by the model that will

help the model most in increasing its performance. This part of the AL is arguably one of the most crucial parts, as it is the main difference between passive and AL. In this research, the querying methods within ASReview were tested on the Enron dataset. Of the six querying methods found within ASReview two did not work which were the mixed query approaches. These approaches combined maximum query with random query and maximum query with uncertainty query. Of the remaining four querying methods, only three were different from normal PL. Of these three querying methods, uncertainty query turned out to be the most useful for training a model on being able to generalize over a set of new documents. This is rather interesting as the default setting uses maximum query and not uncertainty query. It could be possible that maximum query, which takes the instance of which the model is most certain it belongs to the positive class, is very good at finding positive instances within a single dataset but very bad at generalizing over new data. The only querying method that has not been mentioned so far is cluster query, but this did not yield any more interesting results than the two previous querying methods.

It has to be noted that ASReview only includes a small portion of all the querying methods that exist. In the paper by Kumar and Gupta (2020) alone, 19 querying strategies are presented, of which 18 are not available in ASReview. Of these 18 methods, 12 are suited for the problem in this research. Since it is not the goal of this research to compare AL approaches, it is not interesting and worthwhile to discuss the effects of every single querying strategy found in just this paper on the outcomes of this research. Therefore, the only other querying strategy whose effects and consequences on this research are discussed is query by committee. This is the most interesting and promising querying strategy besides uncertainty query that appears within both the paper by Kumar and Gupta (2020) and the other papers which mention AL querying strategies that have been mentioned in this research (R. Hu, 2011; Settles, 2009, 2012; Settles & Craven, 2008). Query-by-committee can technically be seen as a form of query-bagging, since it uses the results of more than one model to settle the choice of what instance is going to give the highest gain in model accuracy. This has been deemed to be a very effective way of performing querying in a way that is still relatively easy to explain and understand. Implementing this querying method could therefore lead to much better results.

When looking at the practical implications of performing this study with the query-by-committee method, there are relatively few major downsides. This is because this is mostly a change in the inner workings of AL and not a change that affects any part of the process that is either linked to the oracle or to model complexity. Of course, the query by committee strategy is a more complex strategy than uncertainty query. But compared to much more complex strategies, such as expected error reduction and variance reduction, it is still relatively simple and easier to explain than possible other strategies. The biggest challenge in implementing this querying strategy would be on the software side.

In this research, it was actually attempted to implement query-by-committee into ASReview. However, due to other challenges which were deemed more important and the level of coding knowledge of the researcher, this attempt was not successful. It should be noted that one of the developers of ASReview did say it should technically be more than possible to implement this querying strategy into ASReview. The question always remains if the developers feel the need to do this as their current standard setup, which consists of a combination of naive bayes, maximum query and double resampling (oversampling) works well enough for ASReview's current intended use case. But since ASReview is an open-source tool, anyone could write the code for implementing this querying method.

## 7.7 The current state of NLP

As specified in section 4.3.2 the field of NLP is very revolved around the English language. Because of this, pre-trained models for the English language, which in this study were used for NER and sentiment analysis, are better than the models for other languages. Even though none of the features that require pre-trained models were used in the Enron test run, in the Shell data run the sentiment analysis feature was used. Due to the fact that this feature was based on a pre-trained model based on the Dutch language, the performance might have been worse than intended. However, as specified in section 4.3.2, since this study is not about how accurate the sentiment values are, this is not a major problem. The same goes for the non-existence of a Dutch passive voice detector. However, since this simply currently does not exist for Python, it is not even known what influence this could have had on the results.

This also brings up the point of languages being inherently different in how certain things are described in

them. The features used in this research might have very different values for different languages, even though the definition of the feature is exactly the same. This issue was combated on a feature level by performing feature selection for both datasets separately. The only place where this issue might still exist is with the configuration testing on the Enron dataset. However, since not enough labels were available to properly do this with the Shell dataset, this was the second-best choice.

## 7.8 Third option within ASReview

Now that the technical side of the research has been highlighted, the focus will now switch to the less technical side of the research. This less technical side has mostly to do with the risks and rewards of using humans in combination with technology, and possible biases that could have occurred that are not directly technically related. This part will start with a discussion of what kind of consequences adding a third labeling option within ASReview would have.

While this discussion subject could also have been in the technical part of this discussion, the non-technical part of this problem and/or opportunity is much more interesting. This subject arose during the execution of the AL run, of which a detailed account can be found in appendix C. When the expert was labeling the Shell papers documents, he sometimes found himself in a situation where he did not know with enough certainty what the proper label of a document should be. This had two reasons. First, some documents just were not incredibly informative on their own. Second, due to the splitting of the documents, some documents that were previously deemed informative enough to label with certainty were now split up into multiple documents which did not all have this quality anymore. While it is entirely dependent on the dataset whether original or split-up documents contain enough information to be able to label them on certain things, it does sprout whether it would be a good idea to add a third option for labeling within ASReview, namely "I don't know".

Adding this option to ASReview would make the reviewing process for the expert easier, as he or she would now be able to just simply click the "I don't know" button if they are not sure about the labeling of a document. It could also result in better labels, as only the documents of which the expert is most certain they belong to the positive class will get labeled in such a way.

However, to get to the stage of adding an "I don't know" option and using it to compare AL with PL, research would need to be performed. This research would focus on how adding such an option affects the AL process in general. Normally, the order in which documents are shown to an expert is determined by the querying strategy that simply looks at the certainty of the model on the labels. However, if an "I don't know" option were added, this process would get undermined. Since now, it could be that document number 5 instead of document number 1 will get labeled. Instead of an "I don't know" option, a slider could also be implemented that gives the expert a more fluent way of showing how certain they are of the decision of a labeling.

In the end, adding more flexibility to the labeling options will most likely result in better labels. But if one wants to use it in a real-world scenario, the implications of it would first need to be properly tested using some reference dataset.

## 7.9 Bias in labeling

Within this research, humans had a very prominent role, as they needed to label the data which ultimately determines how and how well the model will function. The introduction of humans into research introduces validity threats into the equation, as humans tend to exhibit behavior that cannot be controlled very well. This is generally why in research that solely revolves around humans, a control group is used to make sure that the results of the research are actually the results of the variables that were being observed and not something else. While a technical control group was also used in this research, namely in the form of comparing PL with AL, a more social-science-oriented control group was not used. This may have resulted in labels that are not true to the ground truth that was deemed to be known. A more specific example of this can be found in appendix C, in which the AL run on the Shell papers data was described in depth. In this AL run, it was observed that the expert quite quickly was basing his judgment of a document that he had been presented with not solely on the document that he had been presented with but also implicitly used documents that he had previously been presented with. He also tried to "help" the model by pushing it off of certain "irrelevant" topics that he recognized within the documents that he had been presented with previously. This was unexpected behavior and may have led to incorrect labels and/or results. One could write a very detailed set of instructions to

make sure that the expert is executing the task as expected, but even then one would not have removed all validity threats concerning the human in the loop. It may very well be that the expert inherently was wrong, as he may have been biased because of his knowledge or other motives. This could be combated by using more than one expert, however then the question still arises of what the right judgment actually is, as the right judgment may turn out to be factual but at a first glance it might be completely opinionated. This threat to validity is one that will almost always remain, and a researcher will have to accept that what an expert claims to be enough knowledge to make decisions around labeling is actually enough knowledge.

## 7.10 General implication of the results found

Having discussed all the previous subthemes, it can be concluded that this research touches upon a lot of subjects. It connects different fields of research, which makes it complex but also allows for many applications.

When looking at the model used in this research, it turned out that simple models can outperform more complex models.

When looking at the features, it turned out that the length of a document, the emotional value of the document, the standard deviation of the word lengths within the documents and the different amount of words that appear within a document are best at identifying the storyline on the relationship between the city of Assen and the NAM. Of course, the bag of words-feature told the most about the storyline, but this feature is more of a compact representation than something specific. The specific features that pointed to logistics in emails are less interesting, as the Enron dataset was mostly used for testing the method.

Furthermore, the results show that AL is a very promising method for performing machine learning research in a way that is more explainable and effective than PL. This is where this research has extended current research, as such a general and explicit comparison has not been made in recent research. This research also has extended the field of AL in general. As previous research was mostly very focused on one specific application, whereas the setup of this research allows it to be used in any context. This second extension should enrich the field of AL.

The final topic of this discussion is deployment. In this research, the deployment of the method was almost fully executed. The method was deployed through the use of ASReview on the computer of the researcher in a virtual environment. This partial deployment was necessary to get the results to answer the research questions. It is not a full deployment of the method, however. With this current form of deployment, the researcher has to physically be with the expert to execute the AL. This is not ideal and could be combated by putting the ASReview environment either in a docker container or by hosting the interface on a web domain. The analysis of the results could also be done in a smoother way that does not require the results to be saved, stored and retrieved in the ad-hoc way it currently is. This is of less importance than fully deploying ASReview, however, and will therefore be not discussed any further.

# 8 Conclusion

In this study, the Enron and Shell papers datasets were examined and used to build machine learning models. First, the datasets were properly cleaned and features were extracted that could represent the meaning of the documents. After this, feature selection was performed, which resulted in a strong feature set. These features were fed through an Active Learning pipeline, using ASReview, after which accuracy and F1-scores were compared and a final model was made. When analyzing the scores and final models, the following can be said about the research questions. The first sub-research question was "**Is it possible to identify documents that adhere to complex criteria using Natural Language Processing in combination with Active Learning?**". It turned out that it was possible to do this, as the models generated from both the Enron and the Shell papers datasets showed good accuracy and F1-scores. Especially the final model that was generated from the fully labeled Shell papers dataset showed impressive accuracy and F1-scores. Also, when comparing the passive to the Active Learning, the latter usually showed the best performance. For the second sub-research question "**What type of lexical features are more useful for training an Active Learning model to help identify storylines or information about company logistics in business documents?**" the following could be said. For the final Enron model, two simple and three complex lexical features were used. For the Shell model, two simple and two complex lexical features were used. These results would suggest that there is no clear answer on what type of lexical feature is more useful. When looking at the coefficients of the logistic regression models, however, the complex lexical features clearly proved to be more useful than the simple lexical features. Therefore, complex lexical features can be deemed more useful for training an Active Learning model for identifying storylines or documents about logistics. For the third and final sub-research question "**What are the aspects of the datasets that allow for efficient training of the model?**" the following can be said. When comparing all three simulations, the two involving Dutch text surpassed the Passive Learning F1-score much slower than their English counterparts. As stated in the discussion, this could be attributed to the field of NLP being focused on the English language. Not only because the English models for some features might be better, but also since some features cannot be generated easily currently for Dutch text. Another aspect that could influence the training efficiency is the clearness of the label the model is trained on. Company logistics might have been easier to identify than a storyline or a type of news.

Having answered all sub-research questions, the main research question "**To what extent can Active Learning in combination with Natural Language Processing be used to help identify possible cases of corruption and company logistics in business documents?**", could be answered in the following way. By using Active Learning, as opposed to Passive Learning, documents that belong to complex categories such as storylines and logistics can be identified properly and often more quickly than with Passive Learning. Considering, as stated in the introduction, that labeling instances is often costly and time-consuming, Active Learning is an improvement over Passive Learning in real-world scenarios where labels are scarce. Therefore, the processes of investigations into possible corruption and the identification of documents belonging to complex categories, can be aided by Active Learning in combination with NLP. As labels are scarce in these fields and labeling is both time-intensive and costly. While the setup proposed in this research was not the ideal one for aiding investigations into possible corruption, it is more than possible to aid such investigations with this current setup. To ensure efficient Active Learning one should pay special attention to the language of the dataset and the general consensus around the label.

To conclude, Active Learning is a very promising technique that can be used for identifying documents that belong to specific complex categories. This research has shown that with the right setup, Active Learning can easily be applied to any domain with documents as long there is sufficient textual data and at least one expert to aid the Active Learning process.

## 8.1 Future research

If future research were to be performed on the same subject the following topics would be most interesting to investigate. First, it would be interesting to see what kind of changes query-by-committee would have on the results. As this is one of the most praised querying methods, it might have a very positive impact on the results. Second, multi-label classifiers could be an interesting topic to investigate. While it will require major changes in the workings of ASReview, it could possibly make the overall Active Learning pipeline in complex classification problems much more efficient. Finally, one of the main suspected reasons for the Active

Learning run on the Shell papers dataset being less impressive than expected, was the small percentage of data that was labeled during the run. While 333 instances might seem like a lot, this was only around 5% of the total dataset. To be able to properly compare this research to other Active Learning research a larger Active Learning run would need to be performed in which at least 10-30% of data is labeled. This could be achieved by either having more time for labeling or having multiple experts label the same dataset at the same time.

## 8.2 Acknowledgements

The researcher's appreciation goes out to the supervisors for their guidance on the work of this research. Besides the supervisors, the researcher would like to thank Bas van Beek from Follow the Money for providing the Shell papers dataset in a usable format and agreeing on being the expert in the Shell papers Active Learning run. Finally, appreciation also goes out to Willem Meints who is an AI architect at Info Support and helped with thinking about the Active Learning pipeline and assisted in parts of the feature refinement.

# References


Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A., & Arshad, H. (2018, November). State-of-the-art in artificial neural network applications: A survey. *Heliyon*, *4*(11), e00938. doi:10.1016/j.heliyon.2018.e00938

Baum, E. B., & Lang, K. (1992). Query learning can work poorly when a human oracle is used. In *International joint conference on neural networks* (Vol. 8, p. 8).

Brouwer, M. E., Williams, A. D., Kennis, M., Fu, Z., Klein, N. S., Cuijpers, P., & Bockting, C. L. (2019). Psychological theories of depressive relapse and recurrence: A systematic review and meta-analysis of prospective studies. *Clinical Psychology Review*, *74*, 101773. doi:10.1016/j.cpr.2019.101773

Carcillo, F., Borgne, Y.-A. L., Caelen, O., & Bontempi, G. (2017). An assessment of streaming active learning strategies for real-life credit card fraud detection. In *2017 ieee international conference on data science and advanced analytics (dsaa)* (p. 631-639). doi:10.1109/DSAA.2017.10

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T. P., Shearer, C., & Wirth, R. (2000). Crisp-dm 1.0: Step-by-step data mining guide..

Chollet, F. (2015). *Keras.* `https://github.com/fchollet/keras`. GitHub.

Chowdhury, K. R. (2020). Natural language processing. In *Fundamentals of artificial intelligence* (pp. 603–649). New Delhi: Springer India. doi:10.1007/978-81-322-3972-7_19

Cohn, D., Atlas, L., & Ladner, R. (1994, May 01). Improving generalization with active learning. *Machine Learning*, *15*(2), 201-221. doi:10.1007/BF00993277

Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., ... Stoyanov, V. (2019). Unsupervised cross-lingual representation learning at scale. *CoRR*, *abs/1911.02116*. Retrieved from `http://arxiv.org/abs/1911.02116`

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., & Salakhutdinov, R. (2019). Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. *arXiv*. doi:10.48550/ARXIV.1901.02860

Das Bhattacharjee, S., Talukder, A., & Balantrapu, B. (2017, 12). Active learning based news veracity detection with feature weighting and deep-shallow fusion.. doi:10.1109/BigData.2017.8257971

Delobelle, P. (2021). *Robbert.* `https://github.com/iPieter/RobBERT`. GitHub.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019, May). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*. Retrieved 2022-04-14, from `http://arxiv.org/abs/1810.04805`

Fang, M., Li, Y., & Cohn, T. (2017). Learning how to active learn: A deep reinforcement learning approach. *CoRR*, *abs/1708.02383*.

Feldman, R. (2013, apr). Techniques and applications for sentiment analysis. *Commun. ACM*, *56*(4), 82–89. doi:10.1145/2436256.2436274

FTM. (2019, 7). *Shell Papers.* Follow the Money. Retrieved 2022-04-06, from `https://www.ftm.nl/dossier/shell-papers#over`

FTM. (2020, May). *Over follow the money.* Follow the Money. Retrieved from `https://www.ftm.nl/over-ftm`

Ghosh, S., Vinyals, O., Strope, B., Roy, S., Dean, T., & Heck, L. (2016). Contextual LSTM (CLSTM) models for Large scale NLP tasks. *CoRR*. (Publisher: arXiv Version Number: 2) doi:10.48550/ARXIV.1602.06291

Goel, S., Gangolly, J., Faerman, S. R., & Uzuner, O. (2010, January). Can Linguistic Predictors Detect Fraudulent Financial Filings? *Journal of Emerging Technologies in Accounting*, *7*(1), 25–46. doi:10.2308/jeta.2010.7.1.25

Goldberg, Y. (2017). *Neural network methods for natural language processing* (No. #37). San Rafael, Calif.: Morgan & Claypool Publishers. doi:10.2200/S00762ED1V01Y201703HLT037

Harmsen, W., de Groot, J., Harkema, A., van Dusseldorp, I., De Bruin, J., Van den Brand, S., & Van de Schoot, R. (2021). Artificial intelligence supports literature screening in medical guideline development: towards up-to-date medical guidelines. *Narcis*.

Hashimoto, K., Kontonatsios, G., Miwa, M., & Ananiadou, S. (2016, August). Topic detection using paragraph vectors to support active learning in systematic reviews. *Journal of Biomedical Informatics*, *62*, 59–65. doi:10.1016/j.jbi.2016.06.001

He, K., Zhang, X., Ren, S., & Sun, J. (2016, June). Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 770–778). Las Vegas, NV, USA: IEEE. doi:10.1109/CVPR.2016.90

Hu, P., Lipton, Z. C., Anandkumar, A., & Ramanan, D. (2018). Active learning with partial feedback. *arXiv*

*preprint arXiv:1802.07427*.

Hu, R. (2011). Active Learning for Text Classification. *TU Dublin*. (Publisher: Dublin Institute of Technology) doi:10.21427/D70K5Z

Khedr, A. E., & Yaseen, N. (2017). Predicting stock market behavior using data mining technique and news sentiment analysis. *International Journal of Intelligent Systems and Applications*, *9*(7), 22.

Kim, H. K., Kim, H., & Cho, S. (2017, November). Bag-of-concepts: Comprehending document representation through clustering words in distributed representation. *Neurocomputing*, *266*, 336–352. doi:10.1016/j.neucom.2017.05.046

Kim, S., Mai, T.-D., Khanh, T. N. D., Han, S., Park, S., Singh, K., & Cha, M. (2020). Active learning for human-in-the-loop customs inspection. *IEEE*.

Kleinbaum, D. G., Klein, M., & Pryor, E. R. (2010). *Logistic regression: a self-learning text* (3rd ed ed.). New York: Springer.

Klimt, B., & Yang, Y. (2004). The Enron Corpus: A New Dataset for Email Classification Research. In D. Hutchison et al. (Eds.), *Machine Learning: ECML 2004* (Vol. 3201, pp. 217–226). Berlin, Heidelberg: Springer Berlin Heidelberg. (Series Title: Lecture Notes in Computer Science) doi:10.1007/978-3-540-30115-8_22

Kottke, D., Calma, A., Huseljic, D., Krempl, G., & Sick, B. (2017). Challenges of Reliable, Realistic and Comparable Active Learning Evaluation. In *Proceedings of the Workshop and Tutorial on Interactive Adaptive Learning* (pp. 2–14). Skopje, Macedonia: NARCIS.

Kumar, P., & Gupta, A. (2020, July). Active Learning Query Strategies for Classification, Regression, and Clustering: A Survey. *Journal of Computer Science and Technology*, *35*(4), 913–945. doi:10.1007/s11390-020-9487-4

Lafferty, J., McCallum, A., & Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *ACM*.

Laws, F., & Schätze, H. (2008). Stopping criteria for active learning of named entity recognition. In *Proceedings of the 22nd international conference on computational linguistics - volume 1* (p. 465–472). USA: Association for Computational Linguistics.

Liu, B. (2012). Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, *5*(1), 1–167.

Liu, B. (2020). *Sentiment analysis* (2nd ed.). Cambridge, England: Cambridge University Press.

Luhn, H. P. (1958, April). The Automatic Creation of Literature Abstracts. *IBM J. Res. & Dev.*, *2*(2), 159–165. doi:10.1147/rd.22.0159

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, *18*(2), 203-226. doi:10.1016/0004-3702(82)90040-6

Mittal, A., & Goel, A. (2012). Stock prediction using twitter sentiment analysis. *Standford University, CS229 (2011 http://cs229. stanford. edu/proj2011/GoelMittal-StockMarketPredictionUsingTwitterSentimentAnalysis. pdf)*, *15*, 2352.

Mohit, B. (2014). Named entity recognition. In *Natural language processing of semitic languages* (pp. 221–245). Springer.

Muslea, I., Minton, S., & Knoblock, C. A. (2003). Active learning with strong and weak views: A case study on wrapper induction. In *Ijcai* (Vol. 3, pp. 415–420).

Muslea, I., Minton, S., & Knoblock, C. A. (2006, October). Active Learning with Multiple Views. *jair*, *27*, 203–233. Retrieved 2022-06-01, from `https://jair.org/index.php/jair/article/view/10470` doi:10.1613/jair.2005

Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011, 09). Natural language processing: an introduction. *Journal of the American Medical Informatics Association*, *18*(5), 544-551. doi:10.1136/amiajnl-2011-000464

Nakano, F. K., Cerri, R., & Vens, C. (2020, September). Active learning for hierarchical multi-label classification. *Data Min Knowl Disc*, *34*(5), 1496–1530. doi:10.1007/s10618-020-00704-w

Nguyen, D. H. M., & Patrick, J. D. (2014, 05). Supervised machine learning and active learning in classification of radiology reports. *Journal of the American Medical Informatics Association*, *21*(5), 893-901. doi:10.1136/amiajnl-2013-002516

Novak, B., Mladenič, D., & Grobelnik, M. (2006). Text Classification with Active Learning. In M. Spiliopoulou,

R. Kruse, C. Borgelt, A. Nürnberger, & W. Gaul (Eds.), *From Data and Information Analysis to Knowledge Engineering* (pp. 398–405). Berlin/Heidelberg: Springer-Verlag. Retrieved 2022-06-10, from http://link.springer.com/10.1007/3-540-31314-1_48 (Series Title: Studies in Classification, Data Analysis, and Knowledge Organization) doi:10.1007/3-540-31314-1_48

Olsson, F. (2009). A literature survey of active machine learning in the context of natural language processing. *DiVa*.

Pagolu, V. S., Reddy, K. N., Panda, G., & Majhi, B. (2016). Sentiment analysis of twitter data for predicting stock market movements. In *2016 international conference on signal processing, communication, power and embedded system (scopes)* (pp. 1345–1350).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Rabuzin, K., & Modrušan, N. (2019). Prediction of Public Procurement Corruption Indices using Machine Learning Methods:. In *Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management* (pp. 333–340). Vienna, Austria: SCITEPRESS - Science and Technology Publications. doi:10.5220/0008353603330340

Rao, T., & Srivastava, S. (2012). Analyzing stock market movements using twitter sentiment analysis. *IEEE*.

Raviv, E. (2020, November). *Why complex models are data-hungy?* [Blog]. Retrieved 2022-07-06, from https://eranraviv.com/complex-models-data-hungry/

Rehurek, R., & Sojka, P. (2011). Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, *3*(2).

*Requesting information from the government (WOO request)* [Government]. (2022, January). Retrieved 2022-01-06, from https://business.gov.nl/regulation/freedom-of-information/

Reyes, O., Morell, C., & Ventura, S. (2018, January). Effective active learning strategy for multi-label learning. *Neurocomputing*, *273*, 494–508. doi:10.1016/j.neucom.2017.08.001

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, *abs/1910.01108*.

Schraagen, M. P., Brinkhuis, M. J. S., & Bex, F. J. (2017). Evaluation of Named Entity Recognition in Dutch online criminal complaints. *Computational Linguistics in The Netherlands journal*, *7*, 3–16. Retrieved 2022-11-16, from https://dspace.library.uu.nl/handle/1874/356185 (Accepted: 2017-11-03T17:27:02Z)

Settles, B. (2009). Active learning literature survey. *MindsUW*.

Settles, B. (2012). *Active learning*. Morgan & Claypool.

Settles, B., & Craven, M. (2008). An Analysis of Active Learning Strategies for Sequence Labeling Tasks. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing* (pp. 1070–1079). Honolulu: Association for Computational Linguistics.

Sharma, S., Sharma, S., & Athaiya, A. (2020, April). Activation functions in neural networks. *IJEAST*, *4*(12), 310–316.

Sparck Jones, K. (1972, January). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, *28*(1), 11–21. doi:10.1108/eb026526

Tokmetzis, D. (2021, 11). *Follow the money, asr.* https://github.com/ftmnl/asr. GitHub.

Tran, V. C., Nguyen, N. T., Fujita, H., Hoang, D. T., & Hwang, D. (2017). A combination of active learning and self-learning for named entity recognition on twitter using conditional random fields. *Knowledge-Based Systems*, *132*, 179-187. Retrieved from 10.1016/j.knosys.2017.06.023

*UC Berkeley Enron Email Analysis* [Educational]. (2020, July). Retrieved 2022-05-02, from https://bailando.berkeley.edu/enron_email.html

van den Brand, S., & van de Schoot, R. (2022, August). A Systematic Review on the Implementation of AI-aided Systematic Reviews in Clinical Guideline Development. (Publisher: Open Science Framework) doi:10.17605/OSF.IO/ZSM9B

van de Schoot, R., de Bruin, J., Schram, R., Zahedi, P., de Boer, J., Weijdema, F., ... Oberski, D. L. (2020). Asreview: Open source software for efficient and transparent active learning for systematic reviews. *Nature*, *abs/2006.12166*. doi:10.1038/s42256-020-00287-7

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In I. Guyon et al. (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc.

Vlachos, A. (2008, July). A stopping criterion for active learning. *Computer Speech & Language*, *22*(3),

295–312. doi:10.1016/j.csl.2007.12.001

Wallace, B. C., Trikalinos, T. A., Lau, J., Brodley, C., & Schmid, C. H. (2010, December). Semi-automated screening of biomedical citations for systematic reviews. *BMC Bioinformatics*, *11*(1), 55. doi:10.1186/1471-2105-11-55

Waltl, B., Muhr, J., Glaser, I., Bonczek, G., Scepankova, E., & Matthes, F. (2017). Classifying legal norms with active machine learning. In *Jurix* (pp. 11–20).

Wang, M., Feng, T., Shan, Z., & Min, F. (2022, January). Attribute and label distribution driven multi-label active learning. *Appl Intell*. doi:10.1007/s10489-021-03086-8

Warsaw, B., Wouters, T., & Baxter, A. (2007). *Email parser.* `https://github.com/python/cpython/blob/3.10/Lib/email/parser.py`. GitHub.

Weinberger, K., Dasgupta, A., Langford, J., Smola, A., & Attenberg, J. (2009). Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09* (pp. 1–8). Montreal, Quebec, Canada: ACM Press. Retrieved 2022-07-26, from `http://portal.acm.org/citation.cfm?doid=1553374.1553516` doi:10.1145/1553374.1553516

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... Brew, J. (2019). Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, *abs/1910.03771*.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... Rush, A. M. (2020, 10). Transformers: State-of-the-art natural language processing. In (pp. 38–45). Association for Computational Linguistics.

Zhu, J., Wang, H., Hovy, E., & Ma, M. (2010, April). Confidence-based stopping criteria for active learning for data annotation. *ACM Trans. Speech Lang. Process.*, *6*(3), 1–24. doi:10.1145/1753783.1753784

# Appendix

## A Simulation run on Dutch news dataset

Since the configuration test run in this research was ran on an English dataset, and the actual Active Learning (AL) run was on a Dutch dataset, an extra simulation run was performed on a dataset about Dutch news after the FTM AL run. In this appendix, this run will be discussed.

### A.1 Data

The dataset for this run was taken from Kaggle and contains 244 414 entries with 5 columns. These columns contain the date at which a news article was published, the title of the article, the content of the article, the category of the article and the URL of the article. Only the abstract and category columns were used for this simulation. To generate a binary label, which could be used with ASReview, the category column was utilized. From this column, which contains 11 possible categories, the category "politics" was taken. With this category, a new column was created named "label" which contained a 1 if the article was about politics and a 0 if it was not. In total 22 087 articles were about politics and 222 327 were not. Minimal cleaning was performed since this test was more about how AL would perform and not about how well the model would perform on generalizing over new articles. To make this simulation fair, a subset of 2000 instances was used for training and 250 for the holdout set. These subsets have the same positive-negative ratio as the original dataset which is about 10:100.

Of all the features describes in section 4.1, readability index, standard deviation of word length, type-token ratio and bag of words were selected. This was done using the same method as the Shell papers feature selection.

### A.2 Simulations

With the Dutch news dataset, only the logistic regression model was utilized. Furthermore, the querying methods uncertainty and maximum query were tested. The results of these simulation runs can be found below.

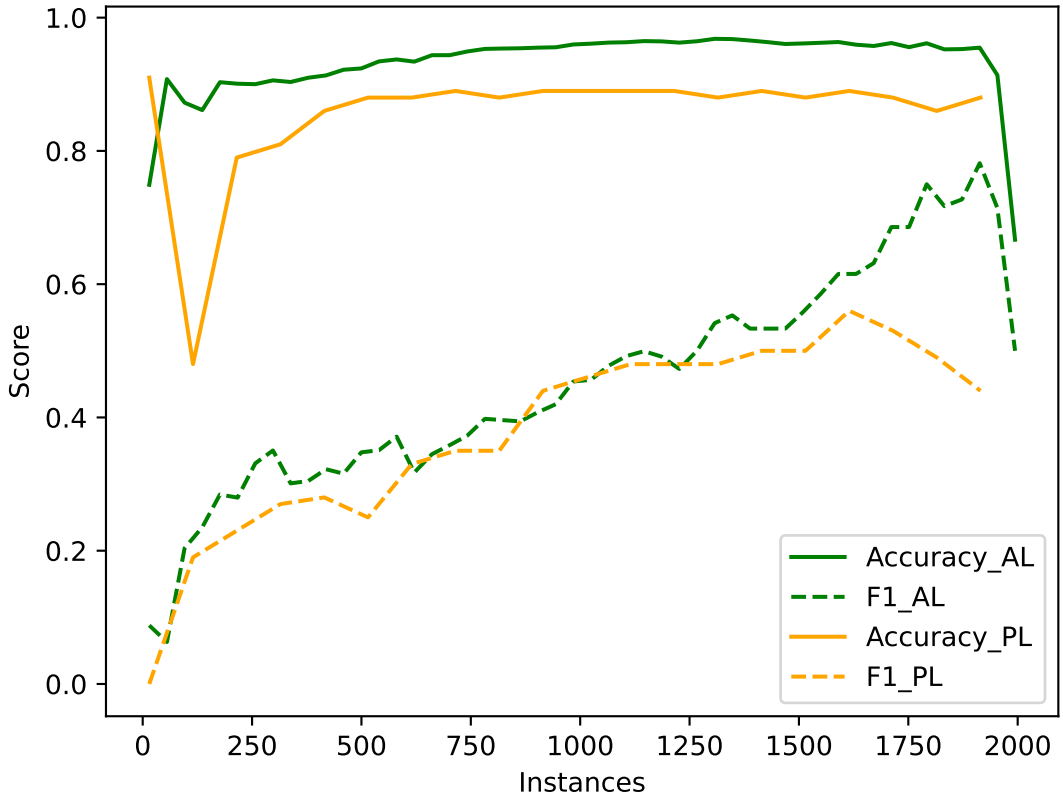



Figure 17: Comparison of F1-scores of both Active and Passive Learning on the test set using uncertainty query.

In Figure 17 the results of a simulation run on the Dutch news dataset with uncertainty query can be found. In this graph, the accuracy and F1-score of the test are plotted. Similar results to the Enron simulation run with the same ASReview settings are found. The only big difference is that the accuracy of the AL is not much higher than that of the Passive Learning (PL). However, this makes sense, as the PL accuracy is already very high so there is not that much to improve upon.

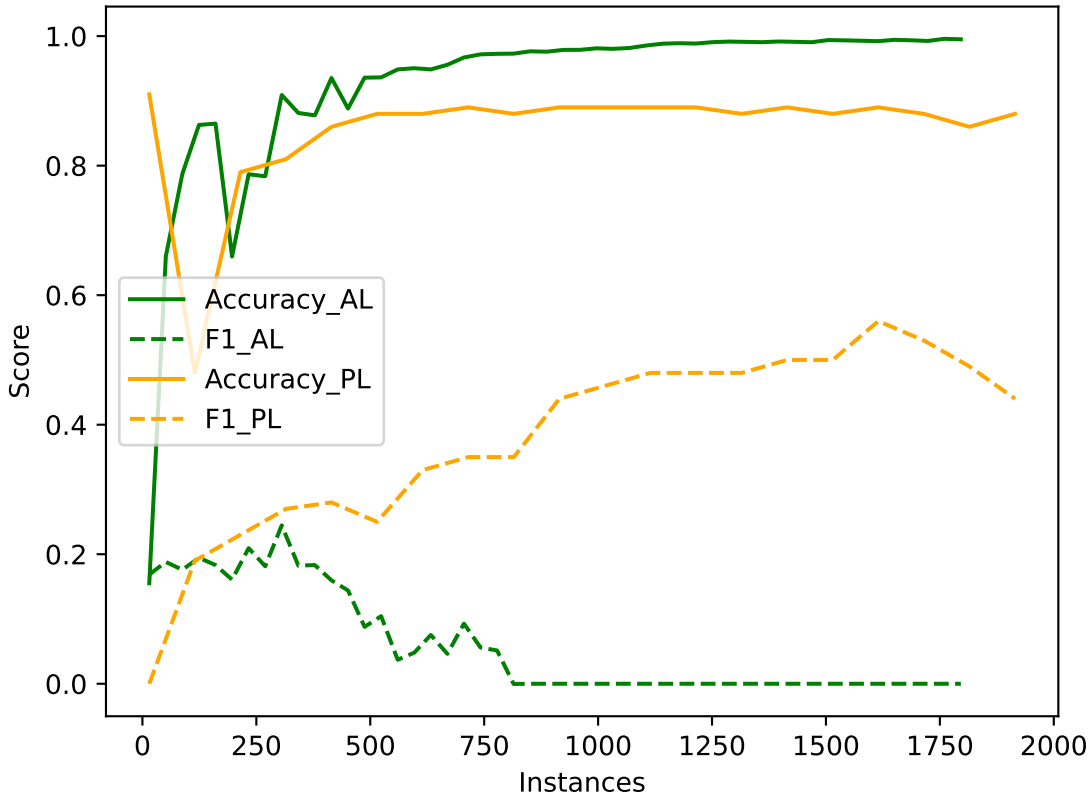



Figure 18: Comparison of F1-scores of both AL and PL on the test set using maximum query.

Similar results to the Enron maximum query simulation run are found in Figure 18. The Accuracy is at the same level while the F1-score is very low. While one could say that a very low F1-score is expected, as the focus of maximum query is to find all positive instances as quickly as possible, this theory should suggest that all positive instances were found at around 800 instances. This is not the case, however, as only 78% of all positive instances have been taken out of the test set at this time. This means that around 800 instances the model labels everything as negative making it a very bad classifier.

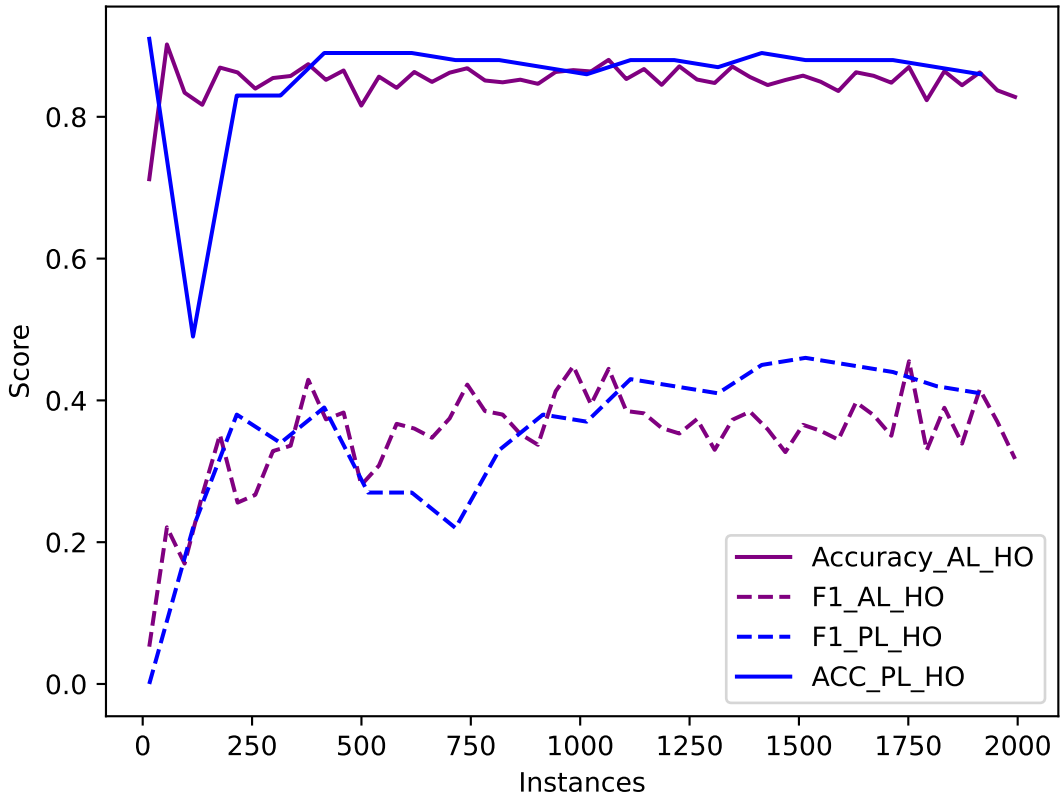



Figure 19: Comparison of F1-scores of both AL and PL on the holdout set using uncertainty query.

In Figure 19 the results of the holdout set with uncertainty query are presented. These results are worse than the test results but not much worse. Mostly the F1-score of AL is more stable than that of PL. Besides the big drop in accuracy for PL in the beginning the accuracy of AL and PL is practically the same.

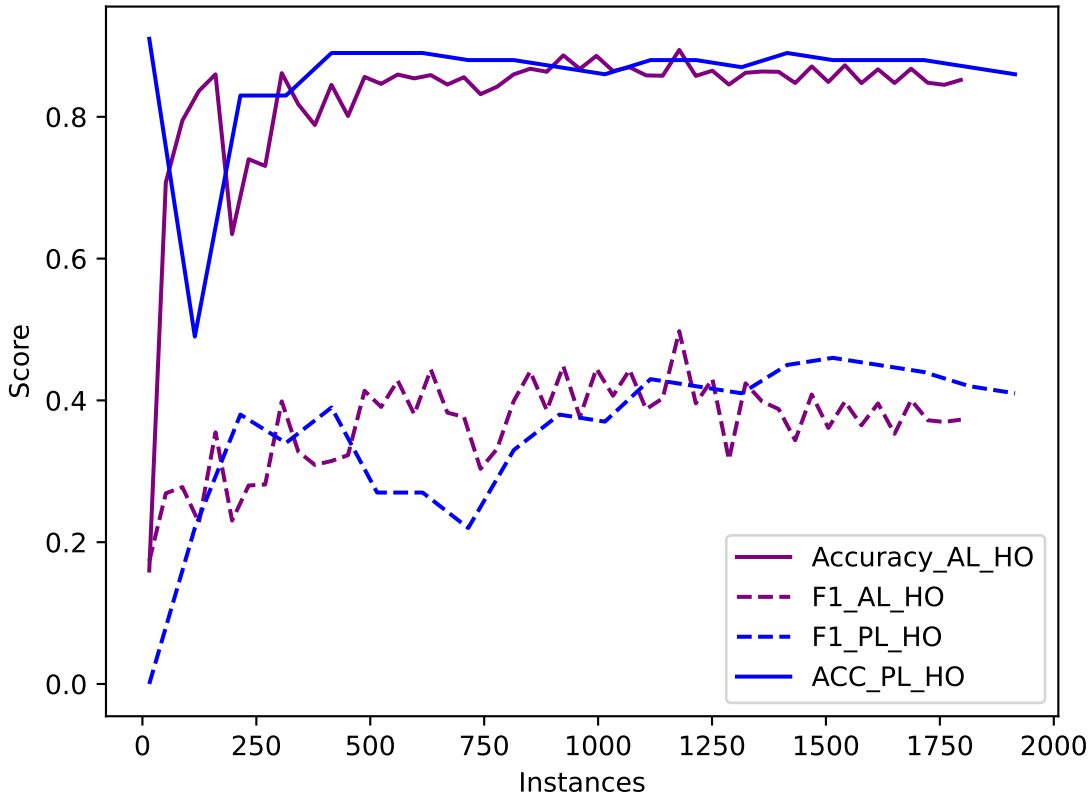



Figure 20: Comparison of F1-scores of both AL and PL on the holdout set using maximum query.

In the final graph, which can be seen in Figure 20, the results of the holdout set with maximum query can be found. Note that the starting points of the lines are not exactly the same as the ones in Figure 19 as the starting sets are not the same. The accuracy is pretty much the same as that of uncertainty query. And while the F1-score is slightly higher than that of uncertainty query, the F1-score is more unstable than that of uncertainty query.

When comparing this figure to the results of the test sets, it almost seems as if two completely different classifiers were used. As with the test set the F1-score at one point became zero whereas the holdout set F1-score did not show such behavior. What has to be taken into account is that minimal cleaning was performed on the dataset. This could have led to the remaining positive instances being outliers and therefore very hard to label properly.

## A.3 Simulation takeaways

It was interesting to see with this simulation that the results of the holdout sets were practically the same between uncertainty and maximum query. While the test results were much worse for maximum query, this could have been caused by bad data. Nonetheless, uncertainty query did not show this issue and therefore it could be considered as better for this context.

# B Extended table of final model performance

This appendix contains the more extended version of the table shown in the final part of the results. The table below includes all information that was given by Python about the model.

Table 6: The results of a logistic regression (top) and neural network (bottom) model trained on the fully labeled dataset.

**Logistic regression**

| | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| 0 | 0.96 | 0.9 | 0.93 | 910 |
| 1 | 0.83 | 0.92 | 0.87 | 450 |
| Accuracy | | | 0.91 | 1360 |
| Macro avg | 0.89 | 0.91 | 0.90 | 1360 |
| Weighted avg | 0.91 | 0.91 | 0.91 | 1360 |

**Neural network**

| | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| 0 | 0.71 | 0.85 | 0.77 | 884 |
| 1 | 0.59 | 0.38 | 0.46 | 496 |
| Accuracy | | | 0.68 | 1380 |
| Macro avg | 0.65 | 0.62 | 0.62 | 1380 |
| Weighted avg | 0.67 | 0.68 | 0.66 | 1380 |

In Table 6, besides the single precision, recall, F1-score and accuracy values other values are present. It not only shows the previously mentioned scores for the negative label, but also the macro and weighted average. The macro average is an unweighted average for the precision, recall or F1-score for both labels that does not take label imbalance into account. The weighted average does take this label imbalance into account and will be higher than the macro average if the model recognizes the majority label better than the minority label. This difference can be caused due to a large difference between the size of the majority and minority label. In the case of the logistic regression, there is barely any difference between these two scores whereas with the neural network the difference is much more profound.

# C Detailed description of Active Learning run FTM

While in the results and discussion section parts of the Active Learning run on the Shell data were discussed, these were all brief reports of this run. In this appendix, a detailed description of the Active Learning run along with all the things mentioned by the expert and other observations are discussed.

The Active Learning run on the Shell data was performed at the Follow the Money office in Amsterdam. Before the run began, the researcher had set up the complete method on his laptop which consisted of a single ASReview project file that had been created in advance with the use of the feature extractor that can be found on GitHub. To test the method, the researcher first asked the expert to label a single document and then wait to see whether the variables necessary for the evaluation were saved correctly. When the process was confirmed to be working, the researcher let the expert label documents at his own pace, with the only requirement that the expert had to wait 5 seconds after labeling each instance since that was the amount of time ASReview needed to train the model.

During the process, which lasted about 3 hours, multiple comments were made by the expert on the Active Learning process. For example, due to the functioning of ASReview, the layout of all the documents needed to be removed. This made the documents harder to read and the expert was slightly annoyed by this. Furthermore, the expert was also slightly annoyed that he had to wait 5 seconds each time with labeling a new instance. Next to this, the expert suggested the need for a "I don't know button".

Besides comments on ASReview itself the expert also made comments on how the data had been prepared. Due to a low amount of labels present in the first part of the dataset, it was decided that this part was to be combined with the second part of the dataset. However, because of this combined dataset, some documents that were completely irrelevant to the storyline on which the Active Learning was performed were included. Furthermore, the expert commented that due to the combination of the first and second dataset the label had become too big.

The expert also had some very interesting comments about he perceived the Active Learning. The expert treated the Active Learning model as if it was a human. What is meant by this, is that the expert tried to understand what the model precisely was learning and tried to adapt its judgments based on this in order to "help" the model in the right direction. What the expert did not know was that his judgment on what he thought the model was doing was incorrect. To account for this the researcher iterated that the decision on whether a document belonged to the storyline should be made based on just the document in front of him and not any other documents that the expert had seen so far. This comment was iterated multiple times to the expert.

A final question that was asked to the expert was how he perceived the process of Active Learning in general, to which the expert jokingly answered: "As necessary".

# D Code

In this appendix, 3 Jupyter notebooks are presented that were used to perform this research. This is not all code that was used, and the rest can be found on Github[4]

## D.1 Shell preprocessing

[4]https://github.com/michagast/Master_thesis

# Preprocess_final_overleaf

October 21, 2022

## 1 Notebook that is used for preprocessing the data that belongs to FollowTheMoney. Part of the code in this notebook was taken from https://github.com/asreview-ftm-hackathon/Data

## 2 Package Imports

[20]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import pandas as pd
from bs4 import BeautifulSoup
import string
import re
import datetime
import numpy as np
import warnings
```

## 3 Import Data

[2]:
```
process = pd.read_csv('https://github.com/ftmnl/asr/raw/main/data/allExport.
↪csv', sep='|')     # Used for the first part of the dataset that is on Github
```

[14]:
```
process_new = pd.read_csv(r'C:
↪\Users\MichaG\Documents\Scriptie\Data-main\export_dataset2.csv') # Used for␣
↪the second part of the dataset that is not on
                                                                              ␣
↪               # Github
```

[3]:
```
#remove noise
process = process.dropna()

process = process.rename(columns = {"file_name_sort": "title", "content":␣
↪"abstract"})

remove = ['.DS_Store', 'NaN', 'Readme.md']
process = process[~process.title.isin(remove)]
```

## 4 Transform Content

[4]:
```
#abstract html to string

translate_table = dict((ord(char), None) for char in string.punctuation)

def prettify(text):
    text = BeautifulSoup(text, 'html.parser').get_text()
    text = text.replace("\r", "")
    text = text.replace('\n', '')
    #text = text.translate(translate_table)
    return str(text)

process.abstract = process.abstract.apply(prettify)
```

## 5 Search formats

[5]:
```
#extract id, type and date from title

import re

numberlist = []
typelist = []
datelist = []

for title in process.title:
    id = re.search('^[0-9\.]+', title)
    if id == None:
        numberlist.append(None)
    else:
        numberlist.append(id.group(0))

    processtype = re.search('(?<=[0-9\_]\_)[a-z A-Z]+(?=\_)', title)
    if processtype != None:
        typelist.append(processtype.group(0))
    else:
        typelist.append("Onbekend")

    date = re.search("[0-9-?]+(?=.pdf$)", title)
    if date == None:
        datelist.append(None)
    else:
        try: datelist.append(datetime.strptime(date.group(0), '%d-%m-%Y'))
        except:
            try: datelist.append(datetime.strptime(date.group(0), '%-d-%-m-%Y'))
            except: datelist.append(date.group(0))
```

```
process["id"] = numberlist
process["type"] = typelist
process["date"] = datelist
```

# 6 Remove uninteresting files

[6]:
```
process.abstract = process.abstract.apply(lambda x: str(x))

for index, row in process.iterrows():
    if re.search(r'\bposter\b', process.loc[index].title): # Remove poster␣
 ↪documents as they most likely do not contain useful info
        process.drop([index], axis = 0, inplace = True)
    elif re.search(r'checksum', process.loc[index].abstract): # Remove any file␣
 ↪that still contains the word checksum
        process.drop([index], axis = 0, inplace = True)
process.abstract = process.abstract.apply(lambda x: re.sub(r'S?s?ubject',␣
 ↪'Onderwerp', x)) # Change the word "subject" to "onderwerp" for every file␣
 ↪to make the data cleaning work for every file.
```

## 6.1 Improve date column

[7]:
```
#Convert normal date column to column with type datetime
def tryconvertdate(date):
    try:
        return pd.to_datetime(date, infer_datetime_format=True)
    except:
        pass

actualdatelist = process.date.apply(tryconvertdate)
actualdatelist = actualdatelist.tolist()
for i in range(1, len(process)):
    if pd.isnull(actualdatelist[i]):
        actualdatelist[i] = process.date.iloc[i]
```

[9]:
```
#Handle edge cases so the entire column becomes of date time format
for i in range(1, len(actualdatelist)):
    if not isinstance(actualdatelist[i], datetime.datetime):
        if actualdatelist[i] == None:
            actualdatelist[i] = pd.Timestamp(None)
        elif not re.search("([?])", actualdatelist[i]) == None:## check if␣
 ↪entry contains question marks
            actualdatelist[i] = pd.to_datetime(actualdatelist[i], errors =␣
 ↪'coerce')
        elif not re.search("[0-9]{4}[-][0-9]{4}", actualdatelist[i]) == None:
            actualdatelist[i] = pd.to_datetime(actualdatelist[i][0:4])
process["date"] = actualdatelist
```

[10]:
```
#Make types uniform
for val in process.index.values:
    if process.type.loc[val] == 'correspondentie':
        process.loc[val] = 'Correspondentie'
```

## 7 Clean Data

[11]:
```
# Clean the title
def cleanTitle(title):
    title = re.sub('^[0-9\.]+_+[a-z A-Z]+_', '', title)
    title = re.sub('[0-9\-]+.pdf$', '', title)
    title = re.sub('.msg_', ' ', title)
    return title

process.title = process.title.apply(cleanTitle)
```

[12]:
```
# Remove email-adresses and links from abstract
process.abstract = process.abstract.apply(lambda x: re.sub(r'\S*@\S*\s?', '',␣
 ↪x))
process.abstract = process.abstract.apply(lambda x: re.sub(r'(http|www)\S+',␣
 ↪'', x))
```

## 8 Properly split documents and extract texts

[13]:
```
def splitdocuments(category):
    ''' Function that cleans and splits documents based on their contents.
    This Method does two things. First of all it tries to extract only the␣
 ↪relevant text from a document.
    If the function notices a document consists of more then one document, it␣
 ↪will try to split the document while adding
    them to the dataframe and removing the old one.
    '''
    for index, row in category.iterrows():                                  ␣
 ↪                                                                  # We␣
 ↪iterate over all rows in the input
        occurences = [(m.start(0), m.end(0)) for m in re.
 ↪finditer(r'\bOnderwerp\:? ?\b',row.abstract)]                             ␣
 ↪               # Find all documents in the entire document by scanning the␣
 ↪amount of times 'Onderwerp' is mentioned
        if len(occurences) > 1:                                             ␣
 ↪                                                                  # Enter␣
 ↪if more than one document in the document was found
            maillist = []                                                   ␣
 ↪                                                                  # Create␣
 ↪an empty list to store emails in if 'Onderwerp' is found more than once
```

```
            rowdf = pd.DataFrame(category[0:0])
            full_mail = ''# Create empty dataframe with correct column names to␣
↪store emails in temporarily
            for i in range(0,len(occurences)):                                  ␣
↪                                                                   # For␣
↪every document found within the document
                rowdf = rowdf.append(row)                                       ␣
↪                                                                   # Since␣
↪only the abstracts of each row need to be changed according to the amount of␣
↪documents found within a document, each time the iteration is entered a new␣
↪copy of the row is added to the datafram
               # Extract the single documents based on indices. By using regex,␣
↪each document is scanned for the end(by using either Groet or Hoogachend).␣
↪Then depending on which is found earlier. The starting index of that word is␣
↪used.
                index1, index2 = None,None                                      ␣
↪                                                                   # Create␣
↪empty variables to store indexes in
                mail = row.abstract[occurences[i][1]:]                          ␣
↪                                                                   # Find␣
↪beginning of document
                if re.search(r'\bg?G?roete?n?\:? ?\b', mail):                   ␣
↪                                                                   # Find␣
↪possible end of document
                    index1 = re.search(r'\bg?G?roete?n?\:? ?\b', mail).span()
                if re.search(r'\b.?h?H?oogachtend\:? ?,?\b', mail):              ␣
↪                                                                   # Find␣
↪possible end of document
                    index2 = re.search(r'\b.?h?H?oogachtend\:? ?,?\b', mail).
↪span()

                if index1 is not None and index2 is not None and index1[0] <␣
↪index2[0]:                                                           # If␣
↪both ends were found in a single document, get the end with the lowest␣
↪beginning index value
                    maillist.append(mail[:index1[1]])
                elif index1 is not None and index2 is not None and index1[0] >␣
↪index2[0]:
                    maillist.append(mail[:index2[1]])
                elif index1 is not None:
                    maillist.append(mail[:index1[1]])
                elif index2 is not None:
                    maillist.append(mail[:index2[1]])
                else:                                                           ␣
↪                                                                   # If␣
↪none of the identifiers were found, simply append the document
```

```
                    maillist.append(mail)
            if maillist:
                                                                                # Append all seperate changed abstract to a new list which then is added to the main inputted document. The original document is also removed.
                #category = category[category['id'] != row.id]
                for j in range(0, len(maillist)):
                    full_mail += maillist[j]
                #full_mail = rowdf.iloc[j,1]
                category.loc[index, 'abstract'] = full_mail
                #category = pd.concat([category, rowdf])
        else:
                                                                                # Only reached if document only consists of a single document, or if the document contains some edge case that was not defined. Does the rest as the for loop above.
            index5, index6, final_abstract = None, None, None
            temp_abstract = row.abstract
            if re.search(r'\bOnderwerp\:? ?\b', temp_abstract):
                temp_abstract = temp_abstract[re.search(r'\bOnderwerp\:? ?\b', temp_abstract).span()[1]:]
            if re.search(r'\bg?G?roete?n?\:? ?\b', temp_abstract):
                index5 = re.search(r'\bg?G?roete?n?\:? ?\b', temp_abstract).span()
            if re.search(r'\b.?h?H?oogachtend,?\b', temp_abstract):
                index6 = re.search(r'\b.?h?H?oogachtend,?\b', temp_abstract).span()

            if index5 is not None and index6 is not None and index5[0] < index6[0]:
                final_abstract = temp_abstract[:index5[1]]
            if index5 is not None and index6 is not None and index5[0] > index6[0]:
                final_abstract = temp_abstract[:index6[1]]
            elif index5 is not None:
                final_abstract = temp_abstract[:index5[1]]
            elif index6 is not None:
                final_abstract = temp_abstract[:index6[1]]

            if not re.search(r'\bOnderwerp\:? ?\b', row.abstract):
                                                                                # If none of the identifiers are found simply append the document
                final_abstract = temp_abstract
            if final_abstract is not None:
                category.loc[index, 'abstract'] = final_abstract
```

```
    return category
```

### 8.1 Clean subsets

[21]:
```
# Apply the previously created function to the relevant subsets
correspondentie = process[process['type'] == 'Correspondentie'].copy()
edited_correspondentie = splitdocuments(correspondentie)

mail = process[process['type'] == 'Mail'].copy()
edited_mail = splitdocuments(mail)

document = process[process['type'] == 'Document'].copy()
document.drop([1925], axis=0, inplace=True) # Column dropped due to very long␣
 ↪abstract
for val in document.index.values:
    if re.search(r'Offerte', document.abstract.loc[val]): # Offertes are not␣
 ↪interesting for text analysis so the abstracts containing this term are␣
 ↪dropped
        document.drop([val], axis=0, inplace=True)

edited_document = splitdocuments(document)
```

## 9 Concatenate and extract

[67]:
```
# Concatenate the previously created subsets and extract them so that they can␣
 ↪easily be loaded in into another python script
finaldf = pd.concat([edited_correspondentie, edited_mail, edited_document])
finaldf['abstract_length' ] = finaldf.abstract.apply(lambda x: len(x))
finaldf.abstract = finaldf.abstract.apply(lambda x: re.sub(r'\d+', "",x))␣
 ↪#remove digits from abstract
finaldf.abstract =  finaldf.abstract.apply(lambda x: re.sub(r'(?:^| )\w(?:$|␣
 ↪)', ' ', x).strip()) #remove single letters surrounded by spaces
finaldf = finaldf.reset_index()
finaldf[['id','type','title','abstract', 'abstract_length']].
 ↪to_excel(r'preprocessed_final_nosplit.xlsx')
```

## D.2 Feature Generation

# Enron_unbalanced_overleaf

October 21, 2022

## 1 In this notebook the feature generation and passive learning for the Enron dataset is performed

[ ]:
```
import pandas as pd                    #For data science purposes
import matplotlib.pyplot as plt        #For plotting
import os
from email.parser import Parser        #For parsing enron emails
from tqdm import tqdm, tqdm_pandas     #For loading bars
import re                              #For performing regex
import numpy as np
import torch                           #For running models with cuda
import string
from transformers import AutoTokenizer, AutoModelForTokenClassification,␣
 ↪AutoModelForSequenceClassification, pipeline
import pickle                          #For importing saved variables
from nltk import data, pos_tag
import nltk
from PassivePySrc import PassivePy     #For detecting passive voice sentences
import enchant                         #For checking english words
from gensim.parsing.preprocessing import remove_stopwords #For removing␣
 ↪stopwords
import spacy
import mmh3
from tensorflow.keras.preprocessing.text import hashing_trick

spacy.prefer_gpu()

tqdm.pandas()

# Folder Path
path = r"C:
 ↪\Users\MichaG\Documents\Scriptie\Data-main\Enron_with_categories\enron_with_categories"

# Change the directory
os.chdir(path)

emaillist = []
```

```
# Read text File

# Extract email from a txt file
def extract_text_from_file(file_path):
    with open(file_path, 'r') as f:
        if file_path.endswith(".txt"):
            data = f.read()
            email = Parser().parsestr(data)
            body = email.get_payload()
            return body
# Extract labels from .cats file
def extract_labels(file_path):
    with open(file_path, 'r') as f:
        if file_path.endswith(".cats"):
            return(f.read().splitlines())

# By using the above two methods, create a dataframe and extract labels and␣
 ↪text from emails. Then concatenate these into one dataframe.
column_names = ["abstract", "label", "length"]
enron_df = pd.DataFrame(columns = column_names)
for root, dirs, files in os.walk(r"C:
 ↪\Users\MichaG\Documents\Scriptie\Data-main\Enron_with_categories\enron_with_categories"):
 ↪
    for file in files:
        numericlabel, label, email, length, filename = None,None,None,None,None␣
 ↪ # Create empty variables for storing results
        if file.endswith('.txt'):
            email = extract_text_from_file(os.path.join(root,file))
            email = ' '.join(email.split()[:len(email.split())])
            length = len(email)
            filename = file.rpartition('.')[0]
            for catfile in files:
                if catfile.endswith('.cats') and catfile.rpartition('.')[0] ==␣
 ↪file.rpartition('.')[0]:
                    numericlabel = extract_labels(os.path.join(root,catfile))
                    for labelset in numericlabel:
                        if '1,4' in labelset: #This can be changed to import␣
 ↪whatever type of label that one would like
                            label = 1
                            break
                        elif not label == 1:
                            label = 0
            if length < 15000 and length > 15: # Remove very long and short␣
 ↪emails
```

```
                enron_df = pd.concat([pd.DataFrame.from_records([{'abstract':␣
 ↪email, 'label': label, 'length' : length, 'title' : filename}]), enron_df],␣
 ↪ignore_index=True)

enron_df = enron_df.astype({'label': 'int64'})
```

[ ]:
```
# Import and download NLTK packages
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from collections import Counter
from num2words import num2words
from nltk.corpus import stopwords

# Download required NLTK packages
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\MichaG\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\MichaG\AppData\Roaming\nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\MichaG\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[ ]:
```
True
```

# 2 Feature generation

## 2.1 Sentiment values

[107]:
```
# Import model and tokenizer for sentiment analysis
model = AutoModelForSequenceClassification.
 ↪from_pretrained("distilbert-base-uncased-finetuned-sst-2-english" ) # Load␣
 ↪in model for sentiment analysis
model.eval() # Make sure model is not in training mode anymore
tokenizernlp = AutoTokenizer.
 ↪from_pretrained("distilbert-base-uncased-finetuned-sst-2-english",␣
 ↪model_max_len=512) # Load in tokenizer and set max_len to 512 as graphics␣
 ↪card can't handle more than that
```

[108]:
```
# Perform sentiment analysis on all rows
tokenizer_kwargs = {'padding':True,'truncation':True,'max_length':512}
```

```
sentiment_analysis = pipeline("sentiment-analysis" , model=model, tokenizer =␣
 ↪tokenizernlp, device = 0) # Create pipeline for performing sentiment analysis
def generatesentimentvalues(text):
    sentiment_result = sentiment_analysis(text, **tokenizer_kwargs) # perform␣
 ↪sentiment analysis on text
    if sentiment_result[0]['label'] == 'NEGATIVE': # As we can only use numbers␣
 ↪for the final model, we need to transform the results from numbers with a␣
 ↪label to purely numbers
        result = 0-sentiment_result[0]['score']
    else:
        result = sentiment_result[0]['score']
    return result
enron_df['sentiment_score'] = enron_df.abstract.progress_apply(lambda x:␣
 ↪generatesentimentvalues(x)) #Apply sentiment analysis on dataframe
```

```
  1%|          | 11/1597 [00:00<01:27,
18.13it/s]C:\Users\MichaG\Anaconda3\lib\site-
packages\transformers\pipelines\base.py:997: UserWarning: You seem to be using
the pipelines sequentially on GPU. In order to maximize efficiency please use a
dataset
  warnings.warn(
100%|      | 1597/1597 [01:28<00:00, 18.13it/s]
```

### 2.2 Named Entity Recognition

[ ]:
```
# Import model and tokenizer for named entity recognition
modelner = AutoModelForTokenClassification.
 ↪from_pretrained('xlm-roberta-large-finetuned-conll03-english',␣
 ↪return_dict=True)
modelner.eval()
#modelner = modelner.to("cuda:0")

tokenizerner = AutoTokenizer.
 ↪from_pretrained('xlm-roberta-large-finetuned-conll03-english')
```

[66]:
```
alphabets= "([A-Za-z])"
prefixes = "(Mr|St|Mrs|Ms|Dr)[.]"
suffixes = "(Inc|Ltd|Jr|Sr|Co)"
starters =␣
 ↪"(Mr|Mrs|Ms|Dr|He\s|She\s|It\s|They\s|Their\s|Our\s|We\s|But\s|However\s|That\s|This\s|Wher
acronyms = "([A-Z][.][A-Z][.](?:[A-Z][.])?)"
websites = "[.](com|net|org|io|gov)"

def split_into_sentences(text):
    text = " " + text + "  "
    text = text.replace("\n"," ")
    text = re.sub(prefixes,"\\1<prd>",text)
```

```
    text = re.sub(websites,"<prd>\\1",text)
    if "Ph.D" in text: text = text.replace("Ph.D.","Ph<prd>D<prd>")
    text = re.sub("\s" + alphabets + "[.] "," \\1<prd> ",text)
    text = re.sub(acronyms+" "+starters,"\\1<stop> \\2",text)
    text = re.sub(alphabets + "[.]" + alphabets + "[.]" + alphabets + "[.
↪]","\\1<prd>\\2<prd>\\3<prd>",text)
    text = re.sub(alphabets + "[.]" + alphabets + "[.]","\\1<prd>\\2<prd>",text)
    text = re.sub(" "+suffixes+"[.] "+starters," \\1<stop> \\2",text)
    text = re.sub(" "+suffixes+"[.]"," \\1<prd>",text)
    text = re.sub(" " + alphabets + "[.]"," \\1<prd>",text)
    if """ in text: text = text.replace(."","".")
    if "\"" in text: text = text.replace(.\"","\".")
    if "!" in text: text = text.replace("!\"","\"!")
    if "?" in text: text = text.replace("?\"","\"?")
    text = text.replace(".",".<stop>")
    text = text.replace("?","?<stop>")
    text = text.replace("!","!<stop>")
    text = text.replace("<prd>",".")
    sentences = text.split("<stop>")
    sentences = sentences[:-1]
    sentences = [s.strip() for s in sentences]
    return sentences

def generate_named_entities(text):
    ''' Function that generates named entity values for the inputted text.
    This Method does a few things. First it splits the text into single␣
↪sentences(split_into_senteces). The short sentences are then removed based␣
↪on the average length of the sentences in the text(remove_short_tokens)
    The tokens are then fed into a tokenizer and the generated tokens are fed␣
↪into a model that generates named entities based on the tokens. The result␣
↪of this is returned as a dict which can then be appended to the dataframe.
    them to the dataframe and removing the old one.
    '''
    tokens = [x for x in split_into_sentences(text) if not any(y in x for y in␣
↪['/','+'])] # split text into sentences and remove any sentence that␣
↪contains / or + as a character
    tokens = remove_short_tokens(tokens)
    if tokens:
        inputs = tokenizerner.batch_encode_plus(tokens, return_tensors="pt",␣
↪padding = True, max_length = 512, truncation = True) # tokenize sentences,␣
↪max_length is 512 for if cuda is enabled to speed the model up
        with torch.no_grad():
            results = modelner(**inputs)
            for i, input in enumerate(inputs['input_ids']):
                namedentities = [modelner.config.id2label[item.item()] for item␣
↪in results.logits[i].argmax(axis=1)] #for every probability for a named␣
↪entity for a word, turn the probabilities into their associated labels
```

```
        entitynumberlist = generate_entity_list(namedentities) #Based on the␣
 ↪array of entity names that is generated, count each entity and make a dict␣
 ↪of this
    else:
        entitynumberlist = {'B-LOC': 0, 'B-MISC': 0, 'B-ORG' : 0, 'I-LOC' : 0,␣
 ↪'I-MISC': 0, 'I-ORG': 0, 'I-PER': 0}
    return entitynumberlist

def remove_short_tokens(tokens):
    average = 0
    for token in tokens:
        average += len(token)
    try:
        average = average/len(tokens)
        return([x for x in tokens if len(x) >= average])
    except:
        return(tokens)

def generate_entity_list(entities):
    B_LOC, B_MISC, B_ORG, I_LOC, I_MISC, I_ORG, I_PER = 0,0,0,0,0,0,0
    for entity in entities:
        if entity == 'B-LOC':
            B_LOC += 1
        elif entity == 'B-MISC':
            B_MISC += 1
        elif entity == 'B-ORG':
            B_ORG += 1
        elif entity == 'I-LOC':
            I_LOC += 1
        elif entity == 'I-MISC':
            I_MISC += 1
        elif entity == 'I-ORG':
            I_ORG += 1
        elif entity == 'I-PER':
            I_PER += 1
    return({'B-LOC': B_LOC, 'B-MISC': B_MISC, 'B-ORG': B_ORG, 'I-LOC': I_LOC,␣
 ↪'I-MISC': I_MISC, 'I-ORG':I_ORG, 'I-PER':I_PER})
```

[ ]:
```
#Gnerate seperate named entity recognition dataframe that contains all entities␣
 ↪found in the texts
col_names_entity_df1 =  ["B-LOC", "B-MISC", "B-ORG", "I-LOC", "I-MISC",␣
 ↪"I-ORG", "I-PER"]
entity_df1 = pd.DataFrame(columns = col_names_entity_df1)
entity_df1 = entity_df1.append({'B-LOC': 0, 'B-MISC': 0, 'B-ORG' : 0, 'I-LOC' :␣
 ↪0, 'I-MISC': 0, 'I-ORG': 0, 'I-PER': 0},ignore_index = True)
#entity_df1 = entity_df1.append(enron_df.abstract.progress_apply(lambda x:␣
 ↪generate_named_entities(x)), ignore_index = True)
```

```
entity_df1.head()
```

[ ]:
```
#Since the structure of the previous dataframe went wrong, extract the␣
 ↪information of the dataframe and put it into a proper dataframe
col_names_entity_df2 =  ["B-LOC", "B-MISC", "B-ORG", "I-LOC", "I-MISC",␣
 ↪"I-ORG", "I-PER"]
entity_df2 = pd.DataFrame(columns = col_names_entity_df2)
for column in entity_df1:
    entity_df2 = entity_df2.append(entity_df1.iloc[0][column],␣
 ↪ignore_index=True)

entity_df2['filename'] = enron_df.filename
enron_df = pd.merge(enron_df, entity_df2, on='filename', how='inner')
enron_df[1:20]
```

[224]:
```
# Since the named entity recognition is very slow on the cpu, and is too big␣
 ↪for the gpu, a saved version of the variable is loaded in using pickle and␣
 ↪appended to the original dataframe.
with open(r'C:\Users\MichaG\Documents\Scriptie\Data-main\NERframe.txt', 'rb')␣
 ↪as f:
    entity_df = pickle.load(f)

enron_df = pd.merge(enron_df, entity_df, left_on='title', right_on='filename',␣
 ↪how='inner')
```

## 2.3 Other features

### 2.3.1 Specific words

[99]:
```
def specific_words_check(text):
    ''' Function that searches for specific words and sums the total occurences
    '''
    amount_of_words = len(re.findall(r'(\b[Mm]+eet?(ing)?s?
 ↪\b|\b[Pp]+lane\b|\bexpense report\b|\b[Cc]+all\b|\b[Vv]+oicemail\b|\b[Ee]+?
 ↪[Mm]+ail(ing)?\b|\b[Ww]+eeks\b|\b([Ss]+chedul(e)?(ing)?
 ↪)|\b[Tt]+ime|\b[Ww]+eek\b|\b[Ii]+nvite?d?(ing)?\b|\b([0-1]?[0-9]):
 ↪[0-5][0-9]\b)', text))
    if amount_of_words:
        return(amount_of_words)
    else:
        return 0
```

[100]:
```
enron_df['specific_words_counter'] = enron_df.abstract.apply(lambda x:␣
 ↪specific_words_check(x)) #apply the previous function using a lambda␣
 ↪statement to create a new column
```

### 2.3.2 Standard deviation of sentence lengths

[109]:
```
def standard_dev_sentence_length(text):
    ''' Function that calulates the standard deviation of the length of all the␣
 ↪sentences in a text.
    '''
    sentences = nltk.tokenize.sent_tokenize(text)
    sentence_length = []
    for item in sentences:
        sentence_length.append(len(item))
    return(np.std(sentence_length))
```

[110]:
```
enron_df['stdev_sentence_length'] = enron_df.abstract.progress_apply(lambda x:␣
 ↪standard_dev_sentence_length(x))
```

```
100%|      | 1597/1597 [00:00<00:00, 2860.47it/s]
```

### 2.3.3 Automated readability Index

[111]:
```
def readability_index(text):
    ''' Function that calculates the automated readability index of a text.
    '''
    sentences = nltk.tokenize.sent_tokenize(text)
    words = text.count(' ')
    characters = len(text) - words
    try:
        return(4.71*(characters/words) + 0.5*(words/len(sentences)) -21.43)
    except:
        return(0)
```

[112]:
```
enron_df['readability_index'] = enron_df.abstract.progress_apply(lambda x:␣
 ↪readability_index(x))
```

```
100%|      | 1597/1597 [00:00<00:00, 3406.37it/s]
```

### 2.3.4 Standard deviation of word lengths

[114]:
```
def standard_dev_word_length(text):
    ''' Function that calculates the standard deviation of the word lengths in␣
 ↪a text.
    '''
    words = text.split()
    words_length = []
    for word in words:
        words_length.append(len(word))
    return(np.std(words_length))
```

[115]:
```
enron_df['stdev_word_length'] = enron_df.abstract.progress_apply(lambda x:␣
↪standard_dev_word_length(x))
```

```
100%|      | 1597/1597 [00:00<00:00, 10765.24it/s]
```

### 2.3.5 Type Token ratio

[9]:
```
def type_token_ratio(text):
    ''' Function that calculates the type token ratio of the text. The type␣
↪token ratio is the ratio between the toal amount of words and the amount of␣
↪unique words in a text.
    '''
    unique = set(text.split())
    return len(unique)/ len(text.split())
```

[10]:
```
enron_df['type_token_ratio'] = enron_df.abstract.progress_apply(lambda x:
↪type_token_ratio(x))
```

```
100%|      | 1597/1597 [00:00<00:00, 12163.69it/s]
```

### 2.3.6 Proper nouns

[ ]:
```
def proper_nouns(text):
    ''' Function that caclulates how many proper nouns there are in a text.␣
↪This is done with the help of the NLTK perceptron tagger.
    '''
    tagged_sent = pos_tag(text.split())
    propernouns = [word for word,pos in tagged_sent if pos == 'NNP']
    return len(propernouns)
```

[ ]:
```
enron_df['proper_nouns'] = enron_df.abstract.progress_apply(lambda x:␣
↪proper_nouns(x))
```

```
100%|      | 1597/1597 [00:25<00:00, 61.86it/s]
```

### 2.3.7 Passive voice sentences

[6]:
```
spacy_model = "en_core_web_lg"
passivepy = PassivePy.PassivePyAnalyzer(spacy_model)
```

```
C:\Users\MichaG\Anaconda3\lib\site-packages\spacy\util.py:837: UserWarning:
[W095] Model 'en_core_web_lg' (3.0.0) was trained with spaCy v3.0 and may not be
100% compatible with the current version (3.3.1). If you see errors or degraded
performance, download a newer compatible model or retrain your custom model with
the current spaCy version. For more details and available updates, run: python
-m spacy validate
  warnings.warn(warn_msg)
```

[7]:
```
def percentage_passive_voice(text):
    ''' Function that caclulates what percentage of sentences in a text are␣
 ↪written in passive voice. This is done using the passivePy package.
    '''
    passive_amount = passivepy.match_text(text, full_passive=True,␣
 ↪truncated_passive=True).passive_count.iloc[0]
    sentence_amount = nltk.tokenize.sent_tokenize(text)
    return passive_amount/len(sentence_amount)
enron_df['passive_voice_%'] = enron_df.abstract.progress_apply(lambda x:␣
 ↪percentage_passive_voice(x))
```

```
100%|      | 1597/1597 [02:35<00:00, 10.26it/s]
```

### 2.3.8 Active voice sentences

[234]:
```
def percentage_active_voice(text):
    ''' Function that caclulates what percentage of sentences in a text are␣
 ↪written in active voice. This is done using the passivePy package.
    '''
    return(1-percentage_passive_voice(text))
enron_df['active_voice_%'] = enron_df.abstract.progress_apply(lambda x:␣
 ↪percentage_active_voice(x))
```

```
100%|      | 1596/1596 [02:03<00:00, 12.94it/s]
```

### 2.3.9 Bag of words

### 2.3.10 preprocessing tfidf

[11]:
```
def convert_lower_case(text):
    return np.char.lower(text)
def remove_email_adresses(text):
    text = re.sub(r'\S*@\S*\s?', '', str(text))
    return text
```

[12]:
```
def remove_stop_words(data):
    stop_words = stopwords.words('english')
    words = word_tokenize(str(data))
    new_text = ""
    for w in words:
        if w not in stop_words and len(w) > 1:
            new_text = new_text + " " + w
    return new_text
```

[13]:
```
def remove_punctuation(data):
    symbols = "!\"#$%&()*+-./:;<=>?@[\]^_`{|}~\n"
    for i in range(len(symbols)):
        data = np.char.replace(data, symbols[i], ' ')
```

```
        data = np.char.replace(data, "  ", " ")
    data = np.char.replace(data, ',', '')
    return data
```

[14]:
```
def remove_apostrophe(data):
    return np.char.replace(data, "'", "")
```

[15]:
```
def stemming(data):
    stemmer= PorterStemmer()

    tokens = word_tokenize(str(data))
    new_text = ""
    for w in tokens:
        new_text = new_text + " " + stemmer.stem(w)
    return new_text
def convert_numbers(data):
    tokens = word_tokenize(str(data))
    new_text = ""
    for w in tokens:
        try:
            w = num2words(int(w))
        except:
            a = 0
        new_text = new_text + " " + w
    new_text = np.char.replace(new_text, "-", " ")
    return new_text
def remove_random_words(data):
    data = str(data)
    data = re.sub(r'\b(USL)?(usl)?\b', '', data)
    data = re.sub(r'\b(\.((DOC)?(doc)?))\b', '', data)
    data = re.sub(r'\b(e-mail)\b', '', data)
    data = re.sub(r'(\<div\>)', '', data)
    data = re.sub(r'(\<br\>)', '', data)
    data = re.sub(r'(.*?\.[\w:]+)', '', data)
    data = re.sub(r'\[image\]', '', data)
    #data = re.sub(r'\b3d\b', '', data)
    return data
```

[16]:
```
def remove_numbers_phonenumbers(text):
    ''' Function that remove specific sequences of numbers from a text so that␣
 ↪they are not seen as words by the BagofWords feature.
    '''
    text = str(text)
    text = re.sub(r'\b([0-9]{3}-[0-9]{3}-[0-9]{4})\b', '', text) #Remove US␣
 ↪phone numbers
    text = re.sub(r'\b([0-1][0-9]\/[0-3][0-9]\/[0-9]{4})\b', '', text) #Removes␣
 ↪dates
```

```
    text = re.sub(r'\b([0-1]?[0-9]):[0-5][0-9]\b', '', text) #Removes timestamps
    text = re.sub(r'\b\w*\d\w*\b', '', text) #Removes single whitespaces
    return(text)
```

[17]:
```
def preprocess(data):
    data = convert_lower_case(data)
    data = remove_email_adresses(data)
    data = remove_random_words(data)
    data = remove_punctuation(data) #remove comma seperately
    data = remove_apostrophe(data)
    data = remove_stop_words(data)
    data = remove_numbers_phonenumbers(data)
    data = stemming(data)
    data = remove_punctuation(data)
    data = remove_numbers_phonenumbers(data)
    data = stemming(data) #needed again as we need to stem the words
    data = remove_punctuation(data) #needed again as num2word is giving few␣
 ↪hypens and commas fourty-one
    data = remove_stop_words(data) #needed again as num2word is giving stop␣
 ↪words 101 - one hundred and one

    return data
```

[18]:
```
enron_df_bow = enron_df.copy() #Create copy so that we dont modify the actual␣
 ↪dataframe

enron_df_bow.abstract = enron_df.abstract.progress_apply(lambda x:␣
 ↪word_tokenize(str(preprocess(x))))
```

```
100%|      | 1597/1597 [01:36<00:00, 16.55it/s]
```

[19]:
```
DF = {}

for i in range(0,len(enron_df_bow)):
    tokens = enron_df_bow.abstract.iloc[i]
    for w in tokens:
        try:
            DF[w].add(i)
        except:
            DF[w] = {i}
for i in DF:
    DF[i] = len(DF[i])
```

[20]:
```
def doc_freq(word):
    c = 0
    try:
        c = DF[word]
```

```
    except:
        pass
    return c
```

[21]:
```
N = len(enron_df_bow)
doc = 0

tf_idf = {}
df_tfidf = pd.DataFrame({'Doc': pd.Series(dtype='str'),
                   'term': pd.Series(dtype='str'),
                   'value': pd.Series(dtype='float'),
                   'label': pd.Series(dtype='int')})
for i in tqdm(range(0,N)):
    tokens = enron_df_bow.abstract.iloc[i]

    counter = Counter(tokens)
    words_count = len(tokens)

    for token in np.unique(tokens):

        tf = counter[token]/words_count
        df = doc_freq(token)
        idf = np.log((N+1)/(df+1))

        tf_idf[doc, token] = tf*idf
        #df_tfidf.loc[len(df_tfidf)] = [doc,token,tf*idf, enron_df.loc[doc].
 ↪label]

    doc += 1
```

```
100%|      | 1597/1597 [00:00<00:00, 1923.99it/s]
```

[ ]:
```
df_tfidf = pd.DataFrame({'Doc': pd.Series(dtype='str'),
                   'term': pd.Series(dtype='str'),
                   'value': pd.Series(dtype='float')})
for key in tf_idf:
    new_row = pd.DataFrame({'Doc':[key[0]], 'term':[key[1]], 'value':␣
 ↪[tf_idf[key]]})
    df_tfidf = pd.concat([new_row,df_tfidf.loc[:]]).reset_index(drop=True)
```

### 2.3.11 Bag of words with hashes

### 2.3.12 preprocessing bow

[22]:
```
enron_df_bow = enron_df.copy()
enron_df_bow['abstract'] = enron_df_bow.abstract.progress_apply(lambda x:␣
 ↪preprocess(x))
```

```
100%|      | 1597/1597 [01:30<00:00, 17.59it/s]
```

[23]:
```
def Hashed_BOW(text, length):
    '''With this function hashes of texts can be generated
    '''
    bow_array = [0]*length
    hashes = hashing_trick(
        text,
        length,
        filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',
        lower=True,
        split=' ',
        analyzer=None)
    for hash_ in hashes:
        bow_array[hash_] = bow_array[hash_]+1

    return bow_array[0:length]
```

[24]:
```
# In this cell a dataframe of all bag of words values is created
BOW_df = pd.DataFrame(columns=[str(i) for i in range(0,1001)])

for i in range(len(enron_df_bow)):
    BOW_values = Hashed_BOW(enron_df_bow.abstract.loc[i], 1001)
    BOW_df.loc[i] = BOW_values
```

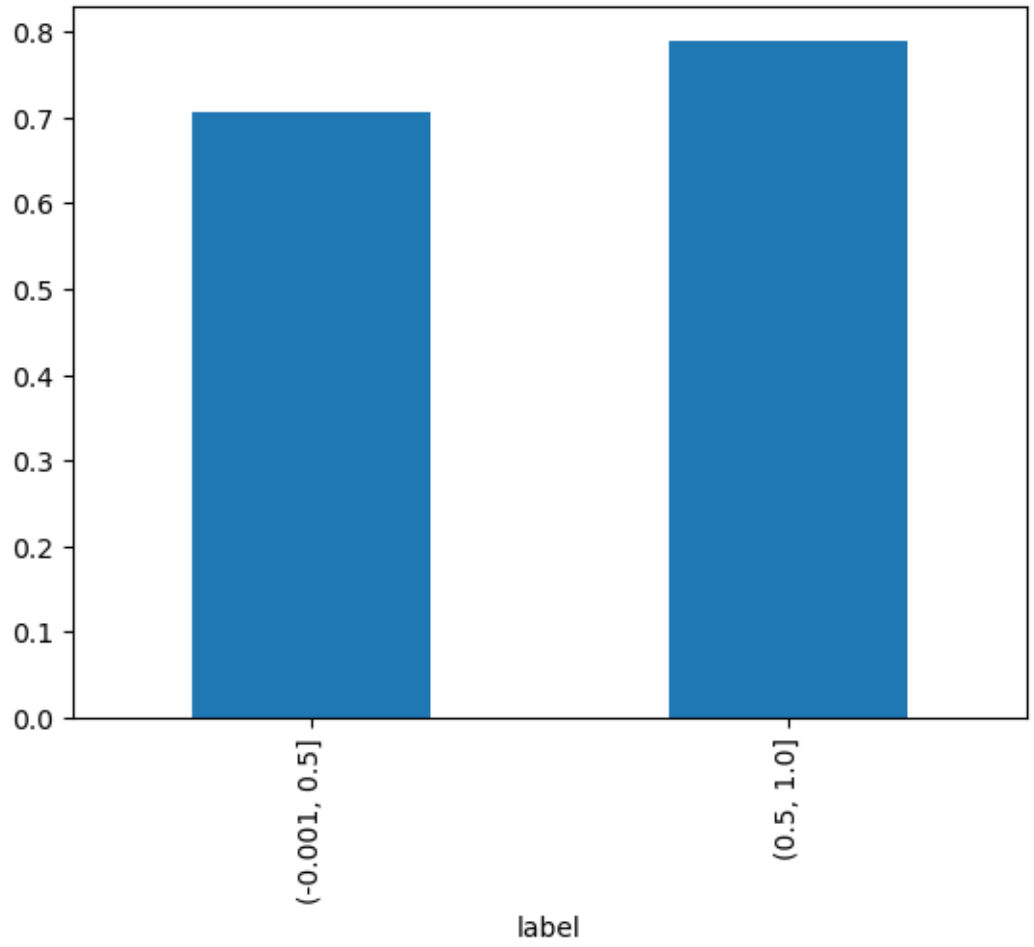
[25]:
```
# In this cell the tf-idf values are applied to the bag of words values
for key in tf_idf:
    BOW_df[str(hashing_trick(key[1], 1001, lower=True)[0])].iloc[key[0]] =␣
 ↪BOW_df[str(hashing_trick(key[1], 1001, lower=True)[0])].
 ↪iloc[key[0]]*tf_idf[key]
```

```
C:\Users\MichaG\AppData\Local\Temp\ipykernel_8072\2187400621.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  BOW_df[str(hashing_trick(key[1], 1001, lower=True)[0])].iloc[key[0]] =
BOW_df[str(hashing_trick(key[1], 1001, lower=True)[0])].iloc[key[0]]*tf_idf[key]
```

[ ]:
```
enron_df_final = pd.concat((enron_df, BOW_df), axis = 1) #Concatenate the␣
 ↪BagOfWords dataframe with the other dataframe
```

[18]:
```
enron_df_final = enron_df.copy() # If no bag of words exist, make the␣
 ↪enron_df_final variable a copy of the frame that contained the other features
```

[194]:
```
# With this cell feature histograms for feature selection can be generated
plot = enron_df_final.groupby(pd.cut(enron_df_final.label, bins =␣
 ↪2))['type_token_ratio'].mean().plot.bar()
plot.figure.savefig(r'C:\Users\MichaG\Pictures\Scriptie\typetokenratio.pdf')
```




### 2.4 Creating subset for testing asreview

[560]:
```
#By using train test split with the setting stratify we can create a small␣
 ↪labeled set in the dataset and a larger unlabeled set which can then be used␣
 ↪for active learning in ASReview
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(enron_df_final.
 ↪drop('label',axis=1),
                                                     enron_df_final['label'],␣
 ↪stratify= enron_df_final['label'], test_size=217, random_state = 29)

enron_df_subset = X_train.copy()
enron_df_subset['label'] = y_train
enron_df_subset1 = X_test.copy()
enron_df_subset1['label'] = np.nan
enron_test_export = pd.concat([enron_df_subset,enron_df_subset1])
enron_test_export = enron_test_export.drop(['length'], axis = 1)
```

```
enron_test_export[['abstract', 'label','title']].to_excel(r'C:
 ↪\Users\MichaG\Documents\Scriptie\Data-main\enron_logistic_df_unbalanced_with_title_small.
 ↪xlsx')
#enron_df.to_csv(r"C:\Users\MichaG\Documents\Scriptie\Data-main\enron.csv")
```

[ ]:
```
#Train test split for when using neural networks. Currently not using this
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(enron_df_final.
 ↪drop('label',axis=1),
                                                    enron_df_final['label'],␣
 ↪stratify= enron_df_final['label'], test_size=0.1)
enron_train_df = pd.concat([X_train.copy(), y_train.copy()], axis = 1)
enron_train_df.drop(['abstract', 'title', 'filename','B-LOC', 'B-MISC',␣
 ↪'B-ORG'], axis=1, inplace = True)
enron_test_df = pd.concat([X_test.copy(), y_test.copy()], axis = 1)
enron_test_df.drop(['abstract', 'title', 'filename','B-LOC', 'B-MISC',␣
 ↪'B-ORG'], axis=1, inplace = True)
```

### 2.5 Simulation subset creation

[135]:
```
enron_df_final[['abstract', 'label', 'title']].to_excel(r'C:
 ↪\Users\MichaG\Documents\Scriptie\Data-main\enron_simulation_data_bussines.
 ↪xlsx')
```

## 3 Testing model

[76]:
```
#Performing testing of the model. By using stratify we can simulate training␣
 ↪the model using a much small dataset and we can make sure that the ratio of␣
 ↪labels in the train and test set is the same as in the orignal dataset
from imblearn.over_sampling import RandomOverSampler, SMOTE
from sklearn.model_selection import train_test_split  #For oversampling

#enron_df_lr = enron_df_final[top50]
#enron_df_lr = enron_df_final.drop(['abstract', 'title', 'filename','B-LOC',␣
 ↪'B-MISC', 'B-ORG', 'length'], axis=1) #Create copy so we don't modify the␣
 ↪original dataframe
enron_df_lr = enron_df_final.drop(['abstract', 'title'], axis=1) #Create copy␣
 ↪so we don't modify the original dataframe


enron_df_small = None
X_train, X_test, y_train, y_test = train_test_split(enron_df_lr.
 ↪drop(['label'],axis=1),
                                                    enron_df_lr['label'],␣
 ↪test_size= 200, random_state = 29)
```

```
ros = RandomOverSampler(random_state=0)
sm = SMOTE(random_state = 0)
X_train_resampled, y_train_resampled = sm.fit_resample(X_train, y_train)␣
 ↪#Oversmampling train test set

enron_df_small = X_train.copy()
enron_df_small['label'] = y_train

#When we want to simulate training with smaller labeled set we can utilise␣
 ↪these variables.
X_train_small, X_test_small, y_train_small, y_test_small =␣
 ↪train_test_split(enron_df_small.drop('label',axis=1),
                                                    enron_df_small['label'],␣
 ↪stratify = enron_df_small['label'],test_size=17, random_state = 29)
X_train_small
```

[76]:

| | length | passive_voice_% | type_token_ratio | 0 | 1 | 2 | \ |
|---|---|---|---|---|---|---|---|
| 1247 | 1941 | 0.214286 | 0.639610 | 0 | 0.000000 | 0.000000 | |
| 457 | 3697 | 0.153846 | 0.562832 | 0 | 0.000000 | 0.000000 | |
| 1447 | 1602 | 0.083333 | 0.614286 | 0 | 0.000000 | 0.000000 | |
| 138 | 938 | 0.000000 | 0.734694 | 0 | 0.000000 | 0.000000 | |
| 320 | 708 | 0.000000 | 0.938144 | 0 | 0.000000 | 0.000000 | |
| … | … | … | … .. | … | … | | |
| 978 | 6557 | 0.351351 | 0.456670 | 0 | 0.000000 | 0.000000 | |
| 899 | 561 | 0.500000 | 0.825000 | 0 | 0.000000 | 0.000000 | |
| 9 | 682 | 0.500000 | 0.784091 | 0 | 0.000000 | 0.000000 | |
| 859 | 9992 | 0.120690 | 0.551597 | 0 | 0.000059 | 0.000553 | |
| 1117 | 3503 | 0.000000 | 0.628366 | 0 | 0.000000 | 0.000000 | |

| | 3 | 4 | 5 | 6 | … | 991 | 992 | 993 | 994 | \ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1247 | 0.000000 | 0.0 | 0.000000 | 0.000000 | … | 0.000000 | 0.0 | 0.0 | 0.0 | |
| 457 | 0.043002 | 0.0 | 0.000000 | 0.000000 | … | 0.000000 | 0.0 | 0.0 | 0.0 | |
| 1447 | 0.000000 | 0.0 | 0.001713 | 1.674147 | … | 0.000000 | 0.0 | 0.0 | 0.0 | |
| 138 | 0.010930 | 0.0 | 0.000000 | 0.000000 | … | 0.000000 | 0.0 | 0.0 | 0.0 | |
| 320 | 0.009876 | 0.0 | 0.000000 | 0.000000 | … | 0.000000 | 0.0 | 0.0 | 0.0 | |
| … | … | … | … | … | … | … | … | … | … | |
| 978 | 0.000000 | 0.0 | 0.000000 | 0.000000 | … | 0.000003 | 0.0 | 0.0 | 0.0 | |
| 899 | 0.000000 | 0.0 | 0.000000 | 0.000000 | … | 0.000000 | 0.0 | 0.0 | 0.0 | |
| 9 | 0.000000 | 0.0 | 0.000000 | 0.000000 | … | 0.000000 | 0.0 | 0.0 | 0.0 | |
| 859 | 0.000000 | 0.0 | 0.000000 | 0.000000 | … | 0.000056 | 0.0 | 0.0 | 0.0 | |
| 1117 | 0.000000 | 0.0 | 0.000000 | 0.017558 | … | 0.000000 | 0.0 | 0.0 | 0.0 | |

| | 995 | 996 | 997 | 998 | 999 | 1000 |
|---|---|---|---|---|---|---|
| 1247 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 457 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.014896 | 0.000000 |
| 1447 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 138 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |

```
320   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
…          …         …         …         …         …         …
978   0.000059  0.006986  0.004146  0.000000  0.000000  0.009414
899   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
9     0.000000  0.000000  0.000000  0.000000  0.202526  0.000000
859   0.002509  0.000000  0.000029  0.005539  0.000191  0.000000
1117  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000

[1380 rows x 1004 columns]
```

[78]:
```
#Perform grid search to find optimal configuration. Takes a long time to do␣
 ↪since there are 200+ variables
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# define models and parameters
logmodel = LogisticRegression(max_iter = 1700, class_weight = 'balanced')
solvers = ['liblinear']
penalty = ['l2']
c_values = [1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.05, 0.03, 0.
 ↪025, 0.02, 0.01]
# define grid search
grid = dict(solver=solvers,penalty=penalty,C=c_values)
cv = RepeatedStratifiedKFold(n_splits=20, n_repeats=5, random_state=1)
grid_search = GridSearchCV(estimator=logmodel, param_grid=grid, n_jobs=-1,␣
 ↪cv=cv, scoring='f1_weighted',error_score=0)
grid_result = grid_search.fit(X_train_small, y_train_small)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
print(classification_report(y_test_small, grid_result.best_estimator_.
 ↪predict(X_test_small)))
```

```
Best: 0.748309 using {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.748309 (0.047330) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.747051 (0.047776) with: {'C': 0.9, 'penalty': 'l2', 'solver': 'liblinear'}
0.745007 (0.047902) with: {'C': 0.8, 'penalty': 'l2', 'solver': 'liblinear'}
0.742150 (0.047330) with: {'C': 0.7, 'penalty': 'l2', 'solver': 'liblinear'}
0.736698 (0.047882) with: {'C': 0.6, 'penalty': 'l2', 'solver': 'liblinear'}
0.731797 (0.046190) with: {'C': 0.5, 'penalty': 'l2', 'solver': 'liblinear'}
0.720024 (0.042338) with: {'C': 0.4, 'penalty': 'l2', 'solver': 'liblinear'}
```

```
0.703654 (0.042454) with: {'C': 0.3, 'penalty': 'l2', 'solver': 'liblinear'}
0.675589 (0.046153) with: {'C': 0.2, 'penalty': 'l2', 'solver': 'liblinear'}
0.649023 (0.048977) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
0.609885 (0.055952) with: {'C': 0.05, 'penalty': 'l2', 'solver': 'liblinear'}
0.599434 (0.060069) with: {'C': 0.03, 'penalty': 'l2', 'solver': 'liblinear'}
0.600348 (0.060443) with: {'C': 0.025, 'penalty': 'l2', 'solver': 'liblinear'}
0.596846 (0.060791) with: {'C': 0.02, 'penalty': 'l2', 'solver': 'liblinear'}
0.583926 (0.061941) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
              precision    recall  f1-score   support

           0       0.80      0.73      0.76        11
           1       0.57      0.67      0.62         6

    accuracy                           0.71        17
   macro avg       0.69      0.70      0.69        17
weighted avg       0.72      0.71      0.71        17
```

[ ]:
```
# With this cell the values of the coefficients of the features can be␣
 ↪presented in a visually appealing way
df_coef = pd.DataFrame(pd.Series(grid_result.best_estimator_.coef_[0]),␣
 ↪columns=['coef'])
df_coef['colnames'] = X_train.columns
# Remove bag of words feature from coefficients
for index, row in df_coef.iterrows():
    if re.search(r'[0-9]', row.colnames):
        df_coef.drop([index], inplace=True)
df_coef
```

[ ]:
```
       coef              colnames
0 -0.000454                length
1  0.007821          proper_nouns
2 -0.805656       passive_voice_%
3  0.104020       sentiment_score
4 -0.000312  stdev_sentence_length
5 -0.008922     readability_index
6  0.099499     stdev_word_length
7  1.419935      type_token_ratio
```

## D.3 Graph generation

# Enron_simulation_overleaf

October 20, 2022

## 1 This notebook serves as an example of how the graphs for each simulation and Active learning run were created. In this specific notebook the Enron dataset is used.

## 2 Data importing

[ ]:
```
# Load in packages
import pandas as pd                  #For data science purposes
import matplotlib.pyplot as plt      #For plotting
import os                            #For importing files directly from the disk
from email.parser import Parser      #For parsing enron emails
from tqdm import tqdm, tqdm_pandas   #For loading bars
import re                            #For performing regex
import numpy as np                   #For using numpy
import torch                         #For running models with cude
import string
from transformers import AutoTokenizer, AutoModelForTokenClassification,␣
 ↪AutoModelForSequenceClassification, pipeline
import pickle                        #For importing saved variables
from keras.models import load_model #For loading in keras models
tqdm.pandas()
```

[1]:
```
#Load in data

# Folder Path
path = r"C:
 ↪\Users\MichaG\Documents\Scriptie\Data-main\Enron_with_categories\enron_with_categories"

# Change the directory
os.chdir(path)

emaillist = []
# Read text File

# Extract email from a txt file
def extract_text_from_file(file_path):
```

```
    with open(file_path, 'r') as f:
        if file_path.endswith(".txt"):
            data = f.read()
            email = Parser().parsestr(data)
            body = email.get_payload()
            return body
# Extract labels from .cats file
def extract_labels(file_path):
    with open(file_path, 'r') as f:
        if file_path.endswith(".cats"):
            return(f.read().splitlines())

# By using the above two methods, create a dataframe and extract labels and␣
 ↪text from emails. Then concatenate these into one dataframe.
column_names = ["abstract", "label", "length"]
enron_df = pd.DataFrame(columns = column_names)
for root, dirs, files in os.walk(r"C:
 ↪\Users\MichaG\Documents\Scriptie\Data-main\Enron_with_categories\enron_with_categories"):
 ↪
    for file in files:
        numericlabel, label, email, length, filename = None,None,None,None,None␣
 ↪ # Create empty variables for storing results
        if file.endswith('.txt'):
            email = extract_text_from_file(os.path.join(root,file))
            email = ' '.join(email.split()[:len(email.split())])
            length = len(email)
            filename = file.rpartition('.')[0]
            for catfile in files:
                if catfile.endswith('.cats') and catfile.rpartition('.')[0] ==␣
 ↪file.rpartition('.')[0]:
                    numericlabel = extract_labels(os.path.join(root,catfile))
                    for labelset in numericlabel:
                        if '1,4' in labelset: #This can be changed to import␣
 ↪whatever type of label that one would like
                            label = 1
                            break
                        elif not label == 1:
                            label = 0
            if length < 15000 and length > 15: # Remove very long and short␣
 ↪emails
                enron_df = pd.concat([pd.DataFrame.from_records([{'abstract':␣
 ↪email, 'label': label, 'length' : length, 'title' : filename}]), enron_df],␣
 ↪ignore_index=True)

enron_df = enron_df.astype({'label': 'int64'})
```

[4]:
```
# The original datafile is necessary for getting the original labels
enron_df_holdout = pd.read_excel(r'C:
 ↪\Users\MichaG\Documents\Scriptie\Data-main\Enron python␣
 ↪data\enron_simulation_data_withholdout_new.xlsx', index_col=[0])
```

[5]:
```
# Predict using active learning model
from sklearn.metrics import precision_recall_fscore_support, accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# With the following function the scores can be generated from each iteration␣
 ↪in a more efficient manner
def generate_scores(model, y_true, X):
    predictions = []
    predictions = model.predict(X)
    accuracy = accuracy_score(y_true, predictions)
    scores = precision_recall_fscore_support(y_true, predictions)
    return [accuracy, scores[0][1], scores[1][1], scores[2][1]]
```

## 3 Data Importing

[6]:
```
# In this cell all relevant scores are generated from the variables that were␣
 ↪created within ASReview.
# First the starting point, ending point and results dataframe are created. The␣
 ↪starting point is the first iteration and the ending point is the last␣
 ↪iteration.
# In the results data frame the final results are stored
import pickle
starting_point = 15
ending_point = 1400
results = pd.DataFrame(columns=['Instance','Accuracy', 'Precision', 'Recall',␣
 ↪'F1', 'Size'])

# Now by using the starting and ending point each variable is loaded in for␣
 ↪every iteration and all scores are generated for every interation
for iteration in tqdm(range(starting_point, ending_point)):
    all_scores = None
    enron_df_subset = enron_df_holdout.copy()
    # open a file, where you stored the pickled data
    file = open(r'C:\Users\MichaG\Documents\Scriptie\ASReview\Simulations␣
 ↪runs\Uncertainty+oversampling+bowfixv4\uncertaintyoversamplingbowfixv4indices'␣
 ↪+str(iteration)+r'.pkl', 'rb')
    file1 = open(r'C:\Users\MichaG\Documents\Scriptie\ASReview\Simulations␣
 ↪runs\Uncertainty+oversampling+bowfixv4\uncertaintyoversamplingbowfixv4alldata'␣
 ↪+str(iteration)+r'.pkl', 'rb')
```

```
    modelfile = open(r'C:\Users\MichaG\Documents\Scriptie\ASReview\Simulations␣
 ↪runs\Uncertainty+oversampling+bowfixv4\uncertaintyoversamplingbowfixv4model'␣
 ↪+str(iteration)+r'.pkl', 'rb')

    # create variable using information in that file
    #model = load_model(r'C:
 ↪\Users\MichaG\Documents\Scriptie\ASReview\Simulations␣
 ↪runs\Uncertainty+oversampling+neural\uncertaintyoversamplingneuralmodel'␣
 ↪+str(iteration)+r'.hd5')
    indices = pickle.load(file)
    data = pickle.load(file1)
    data = np.delete(data, indices, axis = 0)
    model = pickle.load(modelfile)

    enron_df_subset.drop(indices, axis=0, inplace=True)
    labels = enron_df_subset.label

    all_scores = generate_scores(model, labels, data)
    results = pd.concat([results, pd.DataFrame.from_records([{'Instance':␣
 ↪iteration,'Accuracy': all_scores[0], 'Precision': all_scores[1], 'Recall' :␣
 ↪all_scores[2], 'F1' : all_scores[3], 'Size':len(data)}])])
results = results.astype({'Accuracy': 'float64', 'Instance': 'int32'})
results = results.reset_index()
```

```
100%|      | 1380/1385 [00:42<00:00,
41.58it/s]C:\Users\MichaG\Anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1327: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
100%|      | 1381/1385 [00:42<00:00, 32.12it/s]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [6], in <cell line: 7>()
     22     enron_df_subset.drop(indices, axis=0, inplace=True)
     23     labels = enron_df_subset.label
---> 25     all_scores = generate_scores(model, labels, data)
     26     results = pd.concat([results, pd.DataFrame.from_records([{'Instance :
 ↪ iteration,'Accuracy': all_scores[0], 'Precision': all_scores[1], 'Recall' :␣
 ↪all_scores[2], 'F1' : all_scores[3], 'Size':len(data)}])])
     27 results = results.astype({'Accuracy': 'float64', 'Instance': 'int32'})

Input In [5], in generate_scores(model, y_true, X)
      8 accuracy = accuracy_score(y_true, predictions)
      9 scores = precision_recall_fscore_support(y_true, predictions)
---> 10 return [accuracy, scores[0][1], scores[1][1], scores[2][1]]
```

```
IndexError: index 1 is out of bounds for axis 0 with size 1
```

### 3.1 Load holdout data

[10]:
```
# Load in the holdout data that was created outside of ASReview
holdout_set = pd.read_excel(r'C:
 ↪\Users\MichaG\Documents\Scriptie\Data-main\enron_holdoutdata.xlsx',␣
 ↪index_col=[0])
holdout_set_ytrue = holdout_set.label
holdout_set = holdout_set.drop(['label'], axis = 1)
holdout_set_np = holdout_set.to_numpy()
```

[11]:
```
# Generate the same scores as for the trainingdata but now for the holdoutdata
starting_point = 15
ending_point = 1400
results_holdout = pd.DataFrame(columns=['Instance','Accuracy', 'Precision',␣
 ↪'Recall', 'F1', 'Size'])

for iteration in tqdm(range(starting_point, ending_point)):
    all_scores_holdout = None
    modelfile = open(r'C:\Users\MichaG\Documents\Scriptie\ASReview\Simulations␣
 ↪runs\Uncertainty+oversampling+bowfixv4\uncertaintyoversamplingbowfixv4model'␣
 ↪+str(iteration)+r'.pkl', 'rb')
    file = open(r'C:\Users\MichaG\Documents\Scriptie\ASReview\Simulations␣
 ↪runs\Uncertainty+oversampling+bowfixv4\uncertaintyoversamplingbowfixv4indices'␣
 ↪+str(iteration)+r'.pkl', 'rb')
    indices = pickle.load(file)
    model = pickle.load(modelfile)
    all_scores_holdout = generate_scores(model, holdout_set_ytrue,␣
 ↪holdout_set_np)
    results_holdout = pd.concat([results_holdout, pd.DataFrame.
 ↪from_records([{'Instance': iteration,'Accuracy': all_scores_holdout[0],␣
 ↪'Precision': all_scores_holdout[1], 'Recall' : all_scores_holdout[2], 'F1' :␣
 ↪all_scores_holdout[3], 'Size':len(indices)}])])
results_holdout = results_holdout.astype({'Accuracy': 'float64', 'Instance':␣
 ↪'int32'})
results_holdout = results_holdout.reset_index()
```

```
100%|      | 1382/1385 [00:07<00:00, 182.38it/s]
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Input In [11], in <cell line: 5>()
      5 for iteration in tqdm(range(starting_point, ending_point)):
      6     all_scores_holdout = None
----> 7     modelfile =␣
 ↪open(r'C:\Users\MichaG\Documents\Scriptie\ASReview\Simulations runs\Uncertain y+oversampli
```

```
      8     file = open(r'C:
 ↪\Users\MichaG\Documents\Scriptie\ASReview\Simulations␣
 ↪runs\Uncertainty+oversampling+bowfixv4\uncertaintyoversamplingbowfixv4indices␣
 ↪+str(iteration)+r'.pkl', 'rb')
      9     indices = pickle.load(file)

FileNotFoundError: [Errno 2] No such file or directory: 'C:
 ↪\\Users\\MichaG\\Documents\\Scriptie\\ASReview\\Simulations␣
 ↪runs\\Uncertainty+oversampling+bowfixv4\\uncertaintyoversamplingbowfixv4model_397.
 ↪pkl'
```

[7]:
```
# Load in the reference data that was gathered from the passive learning run
data_reference = pd.read_csv(r'C:
 ↪\Users\MichaG\Documents\Scriptie\Data-main\Data voor␣
 ↪grafieken\Reference_run(Uncertainty)_simulation_set.csv', sep=';')
```

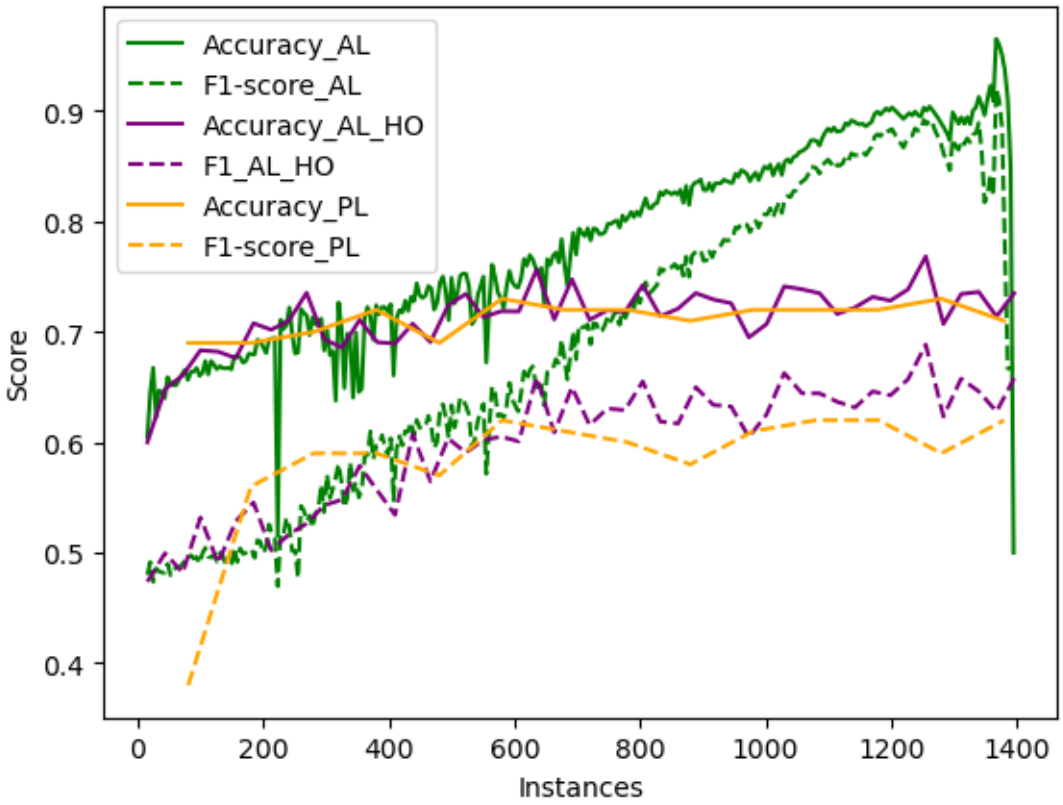
[7]:

| | Instances | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|---|
| 0 | 80 | 0.69 | 0.54 | 0.28 | 0.38 |
| 1 | 180 | 0.69 | 0.52 | 0.62 | 0.56 |
| 2 | 280 | 0.70 | 0.53 | 0.66 | 0.59 |
| 3 | 380 | 0.72 | 0.57 | 0.62 | 0.59 |
| 4 | 480 | 0.69 | 0.52 | 0.64 | 0.57 |
| 5 | 580 | 0.73 | 0.58 | 0.67 | 0.62 |
| 6 | 680 | 0.72 | 0.56 | 0.66 | 0.61 |
| 7 | 780 | 0.72 | 0.57 | 0.65 | 0.60 |
| 8 | 880 | 0.71 | 0.54 | 0.64 | 0.58 |
| 9 | 980 | 0.72 | 0.56 | 0.68 | 0.61 |
| 10 | 1080 | 0.72 | 0.55 | 0.73 | 0.62 |
| 11 | 1180 | 0.72 | 0.56 | 0.70 | 0.62 |
| 12 | 1280 | 0.73 | 0.57 | 0.61 | 0.59 |
| 13 | 1380 | 0.71 | 0.57 | 0.67 | 0.62 |

[14]:
```
# Perform smoothing on the ASReview data
from scipy.interpolate import make_interp_spline, BSpline
xnew_acc = np.linspace(results.Instance.min(), results.Instance.max(), 300)
spl = make_interp_spline(results.Instance, results.Accuracy, k=3)
acc_smooth = spl(xnew_acc)

xnew_F1 = np.linspace(results.Instance.min(), results.Instance.max(), 300)
spl_F1 = make_interp_spline(results.Instance, results['F1'], k=3)
F1_smooth = spl_F1(xnew_F1)

#Create smooth line for holdout
xnew_acc_ho = np.linspace(results_holdout.Instance.min(), results_holdout.
 ↪Instance.max(), 50)
spl_ho = make_interp_spline(results_holdout.Instance, results_holdout.Accuracy,␣
 ↪k=3)
```

```
acc_smooth_ho = spl_ho(xnew_acc_ho)

xnew_F1_ho = np.linspace(results_holdout.Instance.min(), results_holdout.
 ↪Instance.max(), 50)
spl_ho_F1 = make_interp_spline(results_holdout.Instance, results_holdout['F1'],␣
 ↪k=3)
F1_smooth_ho = spl_ho_F1(xnew_F1_ho)
```

[16]:
```
# plot lines
plt.plot(xnew_acc, acc_smooth, label = "Accuracy_AL", color = '#008000')
plt.plot(xnew_F1, F1_smooth, label = "F1-score_AL", linestyle='dashed',␣
 ↪color='#008000' )
plt.plot(xnew_acc_ho, acc_smooth_ho, label = 'Accuracy_AL_HO', color = 'purple')
plt.plot(xnew_F1_ho, F1_smooth_ho, label = 'F1_AL_HO', linestyle='dashed',␣
 ↪color = 'purple')
plt.plot(data_reference.Instances, data_reference.Accuracy, label =␣
 ↪"Accuracy_PL", color='#ffa500')
plt.plot(data_reference.Instances, data_reference['F1-score'], label =␣
 ↪"F1-score_PL", linestyle='dashed', color='#ffa500')
plt.legend()
plt.xlabel('Instances')
plt.ylabel('Score')
plt.show()
#plt.savefig(r'C:
 ↪\Users\MichaG\Pictures\Scriptie\simulation_withholdout(Uncertainty+oversampling).
 ↪pdf') # Save graph is this is necessary
```


Accuracy_AL
F1-score_AL
Accuracy_AL_HO
F1_AL_HO
Accuracy_PL
F1-score_PL
Score
0.9
0.8
0.7
0.6
0.5
0.4
0
200
400
600
800
1000
1200
1400
Instances