Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

Martin Valchev

April 27, 2021

Analysis and Application of Active Learning to Classify Malicious Twitter Posts

Project supervisor:
Oliver Bills

Second examiner:
David Millard

A project report submitted for the award of

BEng Software Engineering

Abstract

As new social networks continue to emerge, so do new mediums for sharing content. This often requires existing machine learning models to be re-trained on new data. Annotating large amounts of data is time-consuming, and expensive but necessary with novel social networks.

This project explores active learning as an alternative, more data-efficient approach to machine learning. The primary objective is to show that active learning can reduce annotation costs and expedite the design and development of models by leveraging the large amount of unlabeled data available.

To evaluate the benefits of active learning, a multi-stranded approach was employed. The learning process was decomposed into individual components that are known to have an impact on performance - dataset, features, classifier, and sampling strategy. All permutations of these components were explored in a passive scenario to obtain reference baselines. Similarly, iterative active learning was evaluated by recording the performance per annotation and comparing it against the baselines.

Results indicate that active learning manages to reduce the annotated data necessary to reach and often exceed the target baselines by more than 50%. This can be observed across most evaluated scenarios. It is evident that active learning is a sustainable approach for reducing annotation costs.

Scope

The project does not necessarily aim to produce a highly accurate model for tweet classification, but rather show annotation costs in relation to F1-scores of models trained using various machine learning techniques.

Project Goals

- To compile an appropriate dataset for machine learning, using Twitter's API, feature selection and feature engineering.
- To develop an application that interactively queries users to label data.
- To apply active learning in the context of social media classification, specifically to classify Twitter posts as malicious or not.
- To produce an analysis of the outcomes of active learning application.
- To compare active learning to passive supervised learning approaches in regards to data efficiency.

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

You must change the statements in the boxes if you do not agree with them.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

I have not used any resources produced by anyone else.

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did not involve human participants, their cells or data, or animals.

ECS Statement of Originality Template, updated August 2018, Alex Weddell aiofficer@ecs.soton.ac.uk

Contents

Definitions

Malicious tweets are defined as posts with negative sentiment, directed at a specific person or a large demographic with a malicious intent, i.e. containing radical, hateful, racist, sexist, phobic or excessively derogatory remarks, or such that could otherwise be described as cyber bullying. Furthermore, posts describing an event where a person is the target of such remarks, are also defined as part of the malicious class for the sake of more extensively capturing the context in which cyber bullying can manifest. Included also are attempts to de-platform public figures as well as online personas in the form of "cancelling".

In this paper, the words "model", "classifier" and "learner" are used synonymously, unless specified otherwise.

"Vectorizer" refers to a model trained on some text corpus to transform raw text into numeric features in some meaningful manner.

"ML" and "AL" are sometimes used in place of "Machine Learning" and "Active Learning" respectively.

1 Introduction

As new social networks continue to emerge, so do new mediums for sharing content. This often requires previously trained machine learning models to either be adapted to the new media using some form of transfer learning, or be re-trained on new data altogether. For example, a model that classifies Facebook posts as malicious or not may be extended to classify Twitter posts with little changes. However, if classifying TikTok shorts on the same classes was the goal, that task would call for a completely different classifier, with different training data. Annotating large amounts of data is time-consuming and expensive, but necessary with novel social networks, where transfer learning is not always applicable.

Active learning is a special case of machine learning, in which an algorithm can interactively query a user, sometimes referred to as an "oracle", to annotate data examples that benefit the model most and are therefore highly informative.

"The key idea behind active learning is that a machine learning algorithm can achieve greater accuracy with fewer training labels if it is allowed to choose the data from which it learns."[1]

This project explores active learning as an alternative, more data-efficient approach to machine learning.

2 Literature Review

While all papers that explore the topic of classification report their results with standardised metrics, a direct comparison is often sub-optimal for arriving at conclusions. This is primarily due to lack of consistency of the environment in which the results are produced. In other words, the specific dataset, features, approach to testing, as well as many other factors, all play a key role in the final evaluation of a machine learning model. As such, most findings in the following literature review are scrutinised and tested, and not accepted "as is".

2.1 Feature Selection and Feature Engineering

The set of available features that a model has access to for training, as well as their discriminative power are directly correlated to the performance the model can achieve. As such, it is important to review the state-of-the-art techniques for feature selection and engineering found in topical literature.

2.1.1 User Features

Dadvar M. et al. [2] propose the use of profile data as features, in addition to text vectorization for tweets, in order to further improve the performance of cyberbullying classification. That is, to derive the common behaviour of users based on patterns of offensive language in previous posts, average length of the comments, age, number

of pronouns, etc. The authors find that including user related information yields approximately 6.6% increase to F1-score in their most successful attempt. They conclude that "users' profile information is not always stated correctly", which attributes to the underwhelming result.

Al-Garadi et al. [3] perform an analysis over a set of features, including network, activity, user and textual information, to identify which features are most discriminative and hence provide the largest information gain in the context of cyberbullying detection. Four out of the top ten features are user related, which alludes to user data playing a significant part in the 42% AUC increase over their baseline result with textual features only, for a total of 0.943 micro F1-score. While their results are impressive, the difference in performance is marginal compared to other state-of-the-art literature that avoids the use of profile data. Again, due to a lack of consistency in dataset and evaluation process between different literature, it is hard to gauge how beneficial user profile information really is, therefore such data is collected and experimented with during prototyping.

Alternatively, one could opt to predict user behaviour with a separate classifier and use the resulting labels as a categorical feature for training the cyberbullying classification model. Balakrishnan V. et al. [4] are able to classify Twitter users as "Bully, Aggressor, Spammer and Normal" with a micro F1-score of ≈ 0.91 by utilising personality and sentiment data. The difference in approach to cyberbullying detection they've taken makes direct comparison to other literature not possible, but the results are promising. However, as this technique requires additional external data, it is in direct conflict of the project's goal to use less labeled data to achieve comparable F1-scores. Predicting user behaviour is therefore not taken into consideration for the implementation of this project.

2.1.2 Textual Features

Textual features are by far the most commonly used features in cyberbullying classification, likely due to the key role profanity plays in malicious texts. Many different techniques are proposed in topical literature, however, in most papers the following text transformation and feature extraction approaches are found to perform best:

- TF-IDF - measure of word importance in a text
- Bag-of-Words - histogram of the words in a text
- Named-entity Recognition - named entities in a text
- Sentiment Analysis - emotion/opinion of a text
- Word Embeddings - word similarity and associations

Rosa H. et al. [5] suggest that feature engineering beyond the use of a TF-IDF transformer is only marginally better and therefore not rewarding.

2.1.3 Statistical Features

Research by Al-Garadi et al. [3] indicates that other types of features can also be utilised successfully, but contribute less to the overall performance of a cyberbullying classification model. These features are less discriminative and are unlikely to benefit the project. Examples of such features include:

- Network Features - number of followers, followed users, following–followers ratio, verification status, etc.
- Activity Features - number of posts, mentions, tweets added to favorites, hashtag usage, etc.

2.1.4 Feature Selection

Feature selection is best analysed by Rosa H. et al. [5], who explore 22 different papers on the topic of automatic cyberbullying detection, as well as perform their own experiment. They found that the top three best-performing combinations of features types are:

- TF-IDF only
- TF-IDF + Word Embeddings
- TF-IDF + Personality Features + Word Embeddings

2.2 Classification Algorithms

Active learning with standard sampling strategies like uncertainty, by definition requires a probabilistic classifier that outputs a confidence measure for its predictions.

Neural networks are known for their inability to produce such measures accurately [6]. They are often overconfident in their labeling, which introduces bias during active learning sampling. Recent study proposes a novel loss function that can be applied to existing neural network architectures without any modification, which results in reliable predictions and is effective for active learning [7]. However, implementing a different loss function for a specific algorithm diverges from the idea of minimal variance during the evaluation of this project. Moreover, the novelty of the approach may well result in a poor implementation by this author and is therefore a project risk. Neural networks also require much more data to perform well in comparison to traditional machine learning algorithms. As such, active learning with neural networks is not considered in this project.

Precisely because of the importance of accurate confidence measures, naturally probabilistic classification models, i.e. models that predict a probability distribution over classes, are more suitable for the active learning task. Examples of such models are Logistic Regression, Naive Bayes, Random Forest, etc. SVM is a notable exception,

whose "convenient mathematical properties" make it "well-suited for active learning", as discussed by Kremer J. et al. [8].

Active learning for malicious text classification hasn't been explored yet in academic literature. However, the closely related topic of cyberbullying detection has been and is being highly studied in passive learning scenarios. The results of papers on the topic are therefore used as vague references for baseline passive learning performance.

One of the best-performing models has been proposed by Al-Garadi et al. [3]. In their paper, the performance of four classifiers - Naive Bayes, Linear SVM, Random Forest and K-Nearest Neighbors - is measured based on various features. Results suggest that the choice of classifier doesn't impact F1-scores as much as the choice of features and dataset balance do. For example, there is only 1.865% difference between their lowest-scoring Naive Bayes approach and their highest-scoring KNN model "under a basic setting". As such, this project focuses on exploring classifiers that are known to perform well in active learning environments, regardless of the classification problem, and not necessarily ones that have been shown to work well with cyberbullying detection specifically.

2.3 Active Learning Scenarios

When discussing active learning, a distinction can be made in regards to the way sampling queries are produced. Literature on the topic refers to this distinction as an "active learning scenario". Usually, a scenario is described as one of the following - membership query synthesis, stream-based selective sampling and pool-based sampling.

First, in the case of membership query synthesis, a model can request labels for both unlabeled data instances from a dataset, as well as ones that are generated by the machine [1]. This is clearly not a good approach for tasks that involve natural language, such as the one of malicious text classification. Maliciousness is not always apparent even in well-structured texts and chances are, a machine would not be able to generate sensible data that contains malicious traces either. Any labels given to nonsensical instances may well degrade the performance of the model on actual testing data. There are recent advances in the field of query synthesis for natural language that show promising results [9], however, more suitable approaches exist.

One such approach is stream-based selective sampling. Unlabeled data instances are drawn sequentially and the machine decides whether to label each one individually [1]. This would work well in practice, as one could fetch fresh tweets as a stream from Twitter's API and the machine would decide for each tweet whether it is worth labeling or should be discarded. This is however not applicable to this project in particular, because one of the requirements for evaluation is that a dataset used across different models must be consistent in order to produce relevant performance results.

This leads to the final and most suitable approach - pool-based sampling. In this

scenario, data instances are sampled from a pool, consisting of both labeled and unlabeled data, based on some sampling strategy [1]. This allows all models to share a single dataset, as well as for language models to be trained on the entire available text corpus at once.

2.4 Query Strategies

Another essential part of the active learning process is the strategy used to sample the available unlabeled data. In fact, the query strategy is almost solely responsible for the benefits of active learning.

For example, consider a random sampling strategy. Although it could use less data overall, it offers no guarantees that answering a query would benefit the model's performance in any way. It is passive learning in a sense.

Alternatively, a sampling strategy that is able to query the data instances a model is least certain about could be described as more optimal. Whether answering such queries would be beneficial is also undetermined, but the likelihood of a positive outcome is certainly higher. This strategy is referred to as "least confidence" sampling in literature [10].

Another candidate strategy for this project is margin sampling, initially introduced by Scheffer et al. [11]. Instead of sampling the data instances with least certainty, it samples those for which the difference between class predictions (margin) is smallest.

Entropy sampling is yet another uncertainty related strategy that is shown to work well in practice. It simply selects the samples displaying the greatest Shannon entropy [12].

Other sampling strategies exist, but are either too complex to implement, depend on assumptions of the data or scale badly due to high computational costs [13].

It is important to note that there is virtually no difference in the queries produced by each discussed strategy on a binary classification problem, as all heuristics coincide in the choice of samples. That being said, all strategies have been shown to perform better than passive learning and random sampling in multinomial classification. While literature refers to the family of these strategies as "uncertainty", this project uses that term in place of "least confidence".

2.5 Data Analysis

The following graphs visualise the relation between dataset size, balance ratio and F1-score in some of the highly influential works on cyberbullying detection:

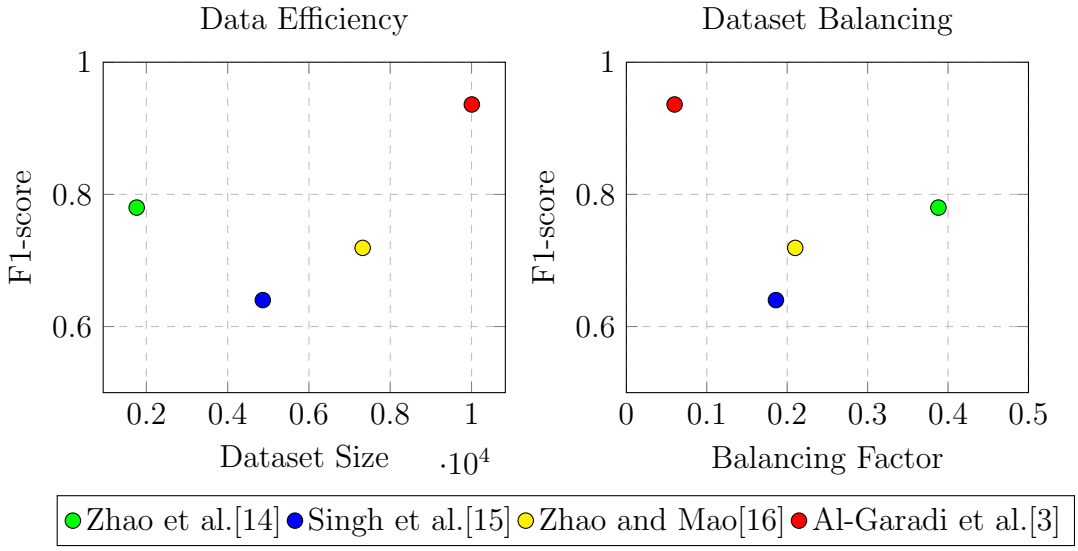


Figure 1: Relation between dataset size, balance ratio and F1-scores.

Analysing the graphs, one can extrapolate that there is direct link between dataset size, balance ratio, and classifier performance. Models with large amounts of labeled data perform well regardless of balancing, possibly because of a higher probability for that data to sample more edge-cases and therefore extract useful information that would otherwise remain unseen. Surprisingly, the smallest dataset [14] performed comparably well, considering it is just 17.61% of the size of the highest performing approach [3]. This can be attributed to the high balance between the number of cyberbullying and non-cyberbullying instances in the dataset. Naturally, there is a large difference between the features and techniques used by the reviewed papers, but looking strictly at sizes and balancing, two possible approaches become apparent - using either a small but balanced or large and not as balanced dataset. Considering the scope and time constraints of this project, a small but relatively balanced dataset seems appropriate.

2.6 Summary

1. User related features improve the overall performance of classification, but only marginally.
2. Features obtained via text vectorization account for the highest classification performance gains.
3. Feature engineering is not rewarding for cyberbullying classification.
4. Choice of machine learning model has low impact on passive learning performance in most cases.
5. Naturally probabilistic classification models are better suited for the project's goals.
6. Pool-based sampling is the most applicable scenario for this project.
7. Uncertainty sampling is easy to implement, yet works well in practice.
8. Small but balanced datasets perform comparably to large unbalanced sets, at a fraction of the annotation costs.

3 Analysis

3.1 Requirements

- The dataset must not contain references or links to Twitter users, i.e. complies with privacy laws and guidelines.
- All data must be labeled accurately.
- The dataset should sample a large enough space, accounting for edge cases whenever possible.
- A classification model should perform well on both training and testing data.
- Identifying malicious instances correctly (recall) should take precedence over not misclassifying non-malicious instances (precision).

3.2 Costs

- Monetary investment if data labeling is outsourced to an external party.
- Time investment if labeling done independently.

3.3 Benefits

- Using a single dataset allows for the training of multiple classification models asynchronously.
- Data efficiency, low data labeling costs.
- Stimulates research in the field of active learning, especially for practical applications.
- Minimal variance during technique comparison ensures accurate results and findings.
- Relatively easy implementation using open-source libraries.
- Accelerates startups interested in cyberbullying detection.
- Offers a robust methodology for rapid development of models on future novel social networks.

3.4 Constraints

- Limited time restricts the size of the dataset that can be acquired and labeled.
- Limited number of data annotators, resulting in lower initial performance.

3.5 Ethics

The dataset used to demonstrate proof of concept on a small scale is collected and labeled exclusively by the author, in a way that respects and adheres to all relevant ethical obligations and concerns. The dataset contains no sensitive data or anything that can't otherwise be accessed or obtained publicly. Furthermore, all collected and processed records are stripped of user-identifying information and are hence anonymised.

This author acknowledges that as a result of having labeled the dataset by themselves, some bias might naturally occur in regards to what is considered malicious or an instance of cyberbullying. However, given the project's problem, such bias is irrelevant as it doesn't impact the evaluation process and results. This project explores the differences between two learning techniques in the context of identical datasets, regardless of the underlying data and its attributes, therefore any inconsistencies or biases equally affect both techniques.

4 Design and Implementation

This section applies the knowledge obtained from literature review to design and implement a complete software product. Furthermore, it extends the scope of explored literature to account for unforeseen topics encountered during the design process. Such topics are discussed and referenced in this section as opposed to Section 2, as design choices follow directly from the topics' logical conclusions.

4.1 Architecture

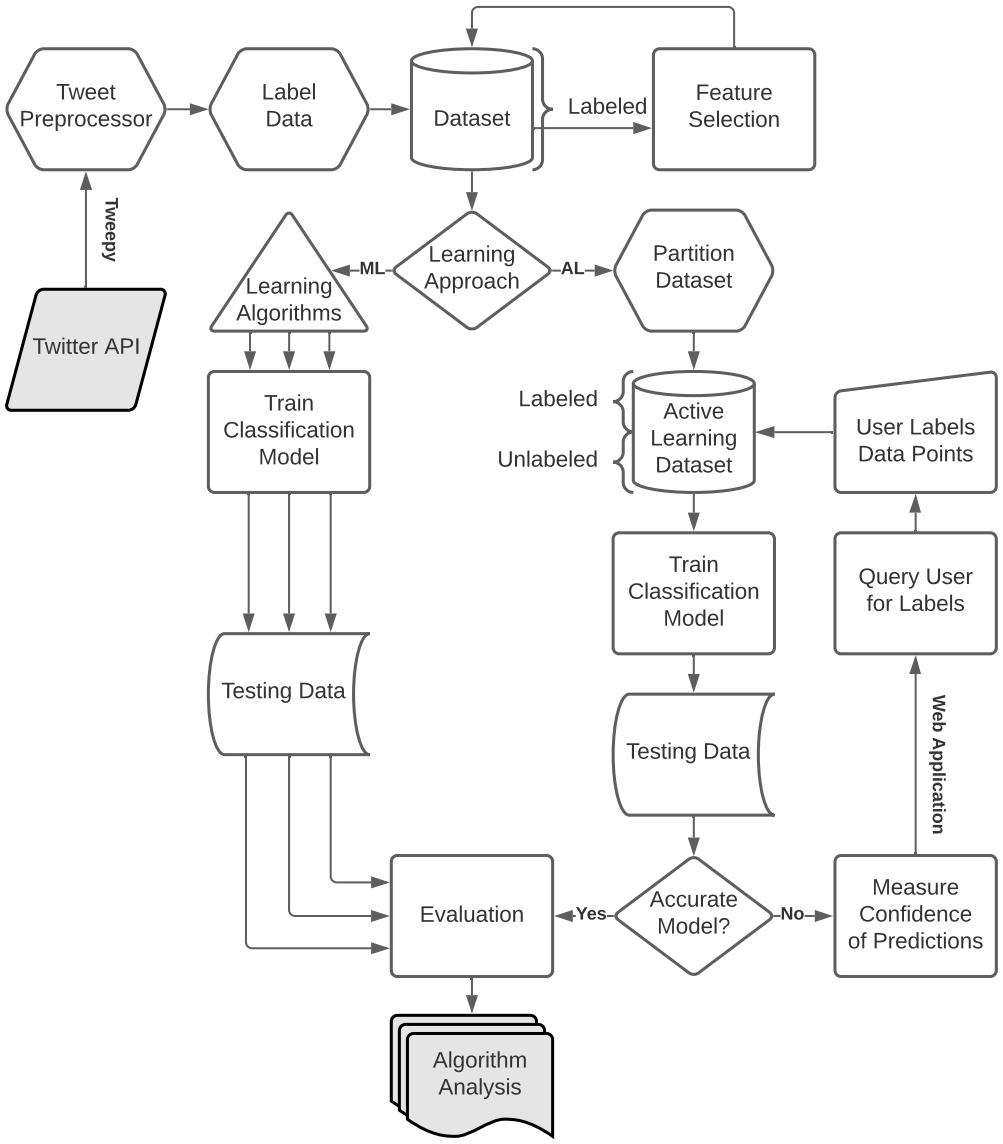


Figure 2: Architecture, modularity and general flow of the implemented system.

4.2 Data Processing

4.2.1 Data Collection

For the purpose data collection, the open-source library *Tweepy* was utilised. It is a python wrapper for Twitter's API, that maps fetched tweets to objects. This reduces complexity significantly, as it makes the subsequent use of data manipulation software like *pandas* much easier. A script was developed that automatically fetches and stores public tweets based on input queries or keywords. Alongside the post text itself, statistical information about the tweet and its author is recorded and later used in conjunction as features, like discussed in Section 2.1.

In order to train a good machine learning model, a sufficiently variant yet balanced dataset is often needed. In fact, compiling an appropriate dataset for a given problem is one of the hardest tasks. To produce one, a strategic approach to data collection was employed. The keywords/phrases used for fetching tweets were broken down into three primary categories based on their attributes:

1. Words/phrases that are offensive and have a high probability of being used in a malicious context
   - *retard, kys, die*[*]
2. Words/phrases that are offensive but not necessarily used in a malicious context
   - *bitch, slut, idiot*[*]
3. Words/phrases that are not offensive but could be found in a malicious context
   - *black, fat, disgusting*[*]

Volume-wise, 40 unique search queries were performed, obtaining 50 samples from each for a total of 2000 tweets. After removing duplicates, 1912 examples constitute the final dataset. Query terms were selected such that all three categories are more or less equally represented. This was done with the idea of producing a naturally balanced dataset, instead of having to apply oversampling or undersampling techniques at a later point. Furthermore, this approach accommodates edge cases well, since tweets based on keywords from the second and third categories are likely harder to predict due to their duality, and so an active learner might prioritise querying their labels early.

To avoid extra processing in the form of translation or language tagging, only English tweets were fetched by utilising Twitter's built-in filtering capabilities. Due to API limitations, the data obtained is temporally restricted to a period of up to seven days before the moment of collection. As a result, the author was not able to take advantage of prior well-known events of cyberbullying or otherwise abusive behaviour.

Nevertheless, fetching Twitter posts based on a fixed range of keywords and search phrases is in turn bound to limit the generalisation power of any machine learning model in terms of overfitting. Due to budget constraints, the initial dataset collected

[*]Not all keywords used for data collection in this project are listed. A keyword's category is assigned by the author per their personal interpretation, which may differ from the reader's.

is final and will not be re-sampled in case all trained classifiers end up being myopic. That is, if they can't accurately classify malicious tweets outside the sampled pool, it will not be considered a fault of the learning technique during evaluation. Given more time, resources and domain knowledge, one could compose an optimized dataset where this wouldn't be an issue, however, this is outside the scope of the project.

4.2.2 Data Partitioning

A common approach to evaluating model performance is to partition a dataset into 3 disjoint subsets - training, validation and testing. The model is first fit on the training set and its performance on the validation set recorded. Then, some form of optimization like hyperparameter tuning or dimensionality reduction is iteratively performed until the score on the validation set is maximised. Finally, the optimized model is evaluated on the previously unused testing set to produce unbiased results. This is known as hold-out. Alternatively, instead of having a static validation set, one could opt to use k-fold cross validation, where the training and validation partitions are re-sampled k times and the mean performance is recorded. Both techniques are appropriate for model selection and the optimal choice is assumed to be data dependent [17].

However, this project isn't concerned with model optimization, so the validation set can be skipped entirely, leaving only training and testing partitions. Because of that, cross validation is inapplicable, as it would not produce consistent testing sets across different learning techniques. That is, active learning would have unique folds after each training iteration, which would not match those of passive learning. When plotting performance per query in comparison to baseline ML later, the results would be meaningless. Besides, cross validation is especially expensive for active learning, due to the iterative nature of training.

It was concluded that a hold-out approach with no validation set is optimal for this project. All explored scenarios are evaluated on equal test partitions. That is, for every model, the dataset is split such that 10% of all examples are withheld and used only for testing, while the rest is available for training. Furthermore, when partitioning the dataset, the seed for generating random split indices is fixed, such that replicability of the samples in each partition is ensured across different models and over subsequent runs.

For the case of active learning, it was initially experimented with a 80:10:10 dataset split ratio for unlabeled pool, initial training set and testing set respectively. However, for a medium-sized dataset, the number of initial training examples in such split proved inappropriate. An observation of the early performance growth rate couldn't be made, since the model had already "seen" substantial amount of training data that wasn't necessarily what would have been queried. For example, when splitting a $\approx 26k$-large dataset, originally used by Thomas Davidson et al. [18] to classify hate speech, ≈ 2600 instances were being randomly selected as initial training data. When testing the model, these instances proved enough to reach adequate accuracy.

To resolve this phenomena and to better replicate a scenario where active learning can be utilised best, the dataset split was adjusted to 10% for testing and 1 sample per class in the data for initial training. The remaining data is used as the sampling pool queried by the learner.

4.2.3 Data Preprocessing

Twitter posts are inherently noisy as suggested by T. Singh [19], which is well accentuated by their imposed limit of 280 characters according to A. Boot et al. [20]. While this could be recognized as a characteristic of the social media platform, it greatly diminishes the quality of information that can be extracted for NLP applications, feature extraction in particular.

In order to prepare the dataset for further processing, all tweets are normalised in a uniform manner. The following list describes the step by step process employed in this project:

1. Filter out all URLs. - ("https://t.co/xxxxxx" $\rightarrow \epsilon$)
2. Transform all emojis to their description. - (U+1F525 $\rightarrow$ "fire")
3. Decode all html-encoded characters. - (&gt $\rightarrow >$)
4. Lowercase all characters. - ("LMAO" $\rightarrow$ "lmao")
5. Expand all contracted forms. - ("hasn't" $\rightarrow$ "has not")
6. Filter out all mentions. - ("@xxxx" $\rightarrow \epsilon$)
7. Remove all non-letter characters. - ("#cancelled" $\rightarrow$ "cancelled")
8. Filter out all stopwords. - ("i hope you" $\rightarrow$ "hope")
9. Lemmatise all words. - ("went" $\rightarrow$ "go")

4.2.4 Feature Extraction

In order to train a classification model, the feature space used as training data must be entirely numerical. Therefore, the raw textual body of tweets must somehow be encoded as real numbers, i.e. transformed into feature vectors or vectorized. Moreover, the resulting vectors must be able to encode some meaningful information about the original text. While, there are lots of techniques that allow for such transformations, this project explores three in particular.

First and foremost, Section 2.1.2 concluded that TF-IDF has been found in external literature to be the best vectorization technique for problems similar to the one discussed in this project. TF-IDF is a numerical statistic that captures the importance of individual words in a document in relation to some text corpus. It is a product of two metrics, the term frequency TF and the inverse document frequency IDF. As such, words that appear frequently in a document, but are only found in few documents overall, are considered more important for that document. The word's discriminative

power is therefore represented with a higher numeric value. The TF-IDF model in this paper is trained on a combination of unigrams, bigrams and trigrams, limited to the top 8000 most discriminative words or sequences.

Another approach to vectorization, which has also been found to work well in practice, is word embeddings. Unlike TF-IDF, which does not take into account the context and order in which words appear, embedding transformers often consider the context surrounding a given word. For example, a two token window to each side of a word, for a 5-gram sequence in total, can be used for training such models [21]. With a word embeddings approach to vectorization, to obtain the vector of a given tweet, the vectors of all individual words that appear in the tweet are added together, and the resulting vector is divided by the number of words for normalisation. Although this is not ideal, it opens the possibility for use of a pre-trained word embeddings model. Such model is bound to be significantly more accurate than one trained on the project dataset, by virtue of training data volume. In this project, a pre-trained GloVe [22] model for word embeddings is used. It is a shallow neural network, trained on 2B tweets with 27B tokens that produces a 50-dimensional vector representation of words.

The final explored approach is an extension of word embeddings - Doc2Vec. The key difference between them is that while one learns vector representations for individual words, the other learns for entire documents or sentences. Because tweets are often contextual and the meaning of a tweet can't really be captured by its individual words, it is probable that one could infer more of the context in which malicious tweets appear by using a document vectorizer. However, because Twitter posts are inherently noisy and tend to be not well-structured, a pre-trained model like in the case of word embeddings can't be used. In this project, a Doc2Vec model is fitted on the collected dataset to explore whether it is beneficial for active learning in particular. All relations learned for document similarity are local to the dataset, which might end up being more discriminative than a general, pre-trained model.

The working hypothesis is that if the vector representation of tweets is somehow able to capture some notion of similarity, then a model could possibly learn more quickly by sampling the most dissimilar tweet at every iteration. When applying a embedding-based model like GloVe or Doc2Vec for vectorization, whenever two tweets are similar, their vector representations are close in vector space. In some sense, a classification model could possibly predict a tweet's label better, if it has previously been taught "similar" tweet examples. By extension of that, the author expects that iteratively teaching an active learner dissimilar tweets would be a higher information gain per query than if such notion of similarity didn't exist, as is the case with TF-IDF. For example, consider a contrived scenario where there is only one tweet previously seen by a learner, and two candidates for sampling in the unlabeled pool:

- Trained example: "I hate Mary"
- $T1$: "I do not like him"
- $T2$: "lmao kys"

With embeddings, the model might be able to infer the label for $T1$ from its training data easier than it would for $T2$, as $T1$ is more similar. As such, it would make sense to teach the model $T2$ first. This relation between the tweets can be observed by constructing the vector of each one and comparing their cosine similarity pairwise:

tweet	I hate Mary	I do not like him	lmao kys
I hate Mary	1	0.972	0.748

Table 1: Cosine similarity between vectorized tweets.

$T2$ is different both syntactically and semantically to the trained example, while $T1$ shares the same sentiment with little to no structural variance. As shown, this relation is captured well - $T2$ is indeed further in vector space and hence more dissimilar.

Because vectorizers don't need annotations for their training data, one could use the entire dataset corpus when fitting them. This especially benefits active learning, as it further leverages the usually large volume of unlabeled data available.

While the body of a given tweet itself is probably the most indicative attribute of whether the tweet is malicious, it is also beneficial to consider other statistical features. In this project, all tweets are further processed to extract the following features:

- Emojis
  demoji is used to obtain a list of all emojis appearing in a tweet
- Polarity and Subjectivity
  TextBlob is used to perform sentiment analysis using a pre-trained model
- Retweet?
  whether a post is a retweet can be detected by traces of "*RT @someone:*"
- URLs, Hashtags, Mentions
  the number of distinct entities in a tweet

While some of these attributes can be obtained directly using Twitter's API, the project implements a general solution that works on any dataset. This is important for comparative analysis, where a uniform approach to data processing is necessary.

4.2.5 Feature Selection

A few unique combinations of features are extensively evaluated in this project. Individually, the types of features can be aggregated into three categories:

1. Text Features
   - *tf-idf, word_embeddings, doc2vec*
2. User Features
   - *user_is_verified, user_posts, user_likes, user_followers, user_friends*

3. Statistical Features
   - *is_retweet, tweet_likes, tweet_retweets, tweet_is_quote, emoji_count, polarity, subjectivity, hashtag_count, mentions_count, words_count, char_count, url_count*

Instead of considering the impact of distinct features, clusters of feature categories are explored, in order to reduce the parameter search space. Text features are present in each combination due to their significant importance to classification performance, as discussed earlier in Section 2.1.2. The combinations are as follows:

- Text features only
- Text + User features
- Text + Statistical features
- Text + User + Statistical features

Although possible to perform additional optimization in the form of dimensionality reduction, such techniques disproportionately affect passive and active learning. In the case of active learning, after each iteration, the entire training set needs to be re-transformed, which is simply too expensive in terms of computational costs. Therefore, dimensionality reduction has not been explored further.

4.3 Active Learning

The design for the active learning process is comprised of two primary components - a server that exposes an API for building and augmenting active learning models, and a web application that interacts with the server to visualise and update a model's state and propagate labeled training examples. These components can otherwise be distinguished as a front-end and back-end layers of the system. Moreover, a further separation of concerns is achieved by following a model–view–viewmodel design pattern, where the server acts as the viewmodel, the web application as the view, and the actual classifier and its state as the model. The view and viewmodel are two-way bound with the help of web sockets, ensuring real-time propagation of update events. Finally, the pattern allows each layer of the system to be developed in parallel, which in hindsight reduced the overall complexity of the project.

The active learning process does not necessarily benefit from having a large amount of annotators simultaneously, as the sampling strategies used in this project are interactive. In other words, they iteratively produce single queries, hence only one annotator can work on the data at a time. However, all components were designed with scalability in mind, as the system can easily be extended by implementing alternative query strategies in the future. The final software product utilises *Docker*[23] containers to encapsulate the front-end and back-end components, such that they are platform-independent and easily maintainable. These containers can be orchestrated with *Kubernetes* for automated deployment and scaling if necessary.

4.3.1 Backend

Everything backend related has been implemented in python for the benefit of a coherent project environment with a high availability of machine learning libraries. The core functionality of both the passive and active learning process is self-contained and can be used without a supporting web application running. The structure of the backend is partitioned in a way that aggregates closely related functionality in distinct modules. More or less, these modules correspond to individual "steps" in the learning process, which were themselves modelled earlier in Section 4.1.

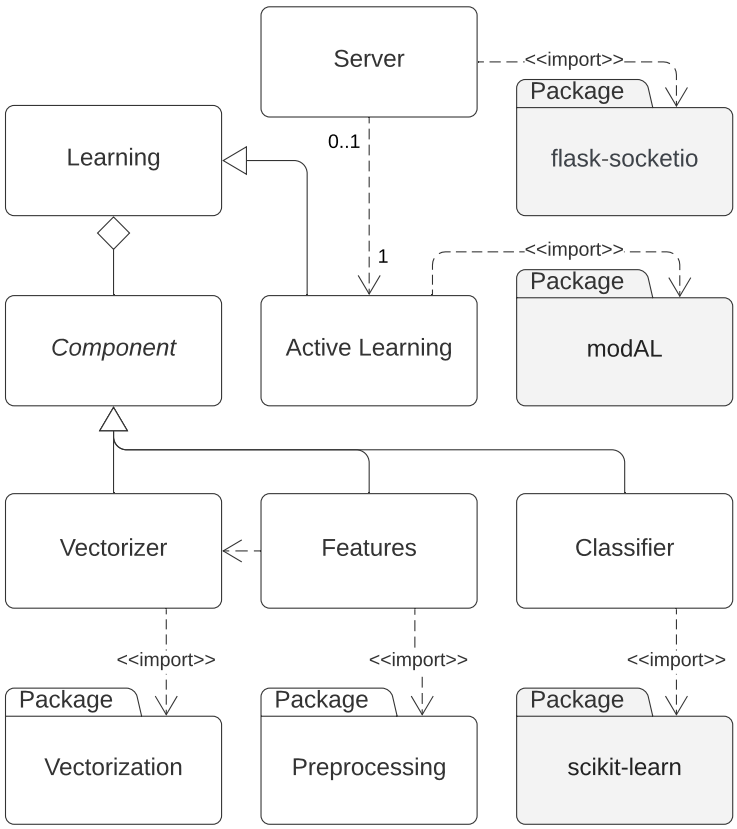


Figure 3: Conceptual model of the structure and composition of the back-end application layer. External dependencies/packages are grayed out.

The *learning* module encapsulates what this author refers to as a "learning scenario". A learning scenario is a high-level view of the combination of individual components that take part in the building of a classification model, i.e. the combination of vectorizer type, classifier class, dataset and features. The module offers the ability to train a model from scratch by only specifying the type of said components. It can also be easily extended with novel components during initialisation. For example, a custom preprocessing function can by supplied as a parameter, which will be executed in place of the default one provided in the *preprocessing* module. All components can

be replaced in a similar manner, as long as they implement the interface of their original counterpart, i.e. a vectorizer need only provide a "vectorize" method if it's already trained, and a "learn_text_model" if it isn't.

The *active learning* module extends the *learning* module by wrapping the classifier component in a thin *modAL* [24] wrapper, that provides the core functionality for iterative labeling. Furthermore, the module implements the active learning dataset partitioning strategy discussed in Section 4.2.2. It also adds support for an additional component - the choice of query strategy. What *modAL* offers specifically, is a class that keeps a reference to a classifier and its previously seen labeled data. When adding new training examples, i.e. teaching the learner, they are stacked row-wise with the existing seen data, and the model is re-fit on the combined result. This is a simple yet effective way to perform active learning with classifiers that don't natively support online training. Furthermore, the class exposes an API for querying the wrapped classifier, given a reference to a function. The only requirement is that the said function must return a sample of the same shape as previously seen data, alongside its index in the unlabeled pool.

4.3.2 Server

While a web application is not required for either passive or active learning, in a real environment, data annotators should be able to enjoy a visual interactive experience. As such, a *server* module was implemented to mediate the communication between the learning process and potential clients. It is designed to be frontend-agnostic by operating entirely via web sockets. All this module does, is wrap an instance of the active learning class, such that it is able to observe its state. Beyond that, it listens for events from connected clients and sends updates when appropriate. These events follow a predefined structure and are named according to the specification in Section 4.3.4. The actual server runs on the web framework *Flask*, with *Flask-SocketIO* as the library of choice for extended web sockets support.

Although it is possible to maintain multiple instances of active learners with different classifiers simultaneously, it is inefficient and ineffective. It would offer more flexibility but the trade-off is a phenomena described by Burr Settles as:

"...a training set built in cooperation with an active learner is inherently tied to the model that was used to generate it (i.e., the class of the model selecting the queries)."[1]

It begs the question, when a sampling query is answered, should one update all learners that use the same dataset with it, because it is now a labeled example? Or rather, because a sample is only informative for the learner that originally made the query, one should update learners independently? Both options are inadequate - one either labels data that isn't asked for (ineffective) or labels the same data more than once (inefficient). Moreover, the computing costs for maintaining multiple models are very high, as each one needs to store and query its unlabeled pool independently. These issues are not as problematic for learners with different datasets, but as there

is no uniform solution for all cases, the author has chosen not to explore the topic of parallelism further.

4.3.3 Web Application

In the early stages of development, a REST API based communication between the server and client was envisioned. However, it proved to be inadequate for the task due to lack of real-time updates. Ultimately, the idea was scrapped in favour of web sockets. To complement that, the web application was developed in *Node.js* due to to its asynchronous, event-driven nature. The frontend sits on top of *React*, which works well in combination with *Node.js* and benefits from a great ecosystem of tools and libraries. For example, the *Material-UI* framework was used to create out-of-the-box stylish and accessible view components that follow Google's Material Design guidelines. The result is a fast, single-page application that offers accessibility, extensibility and looks aesthetically pleasing.

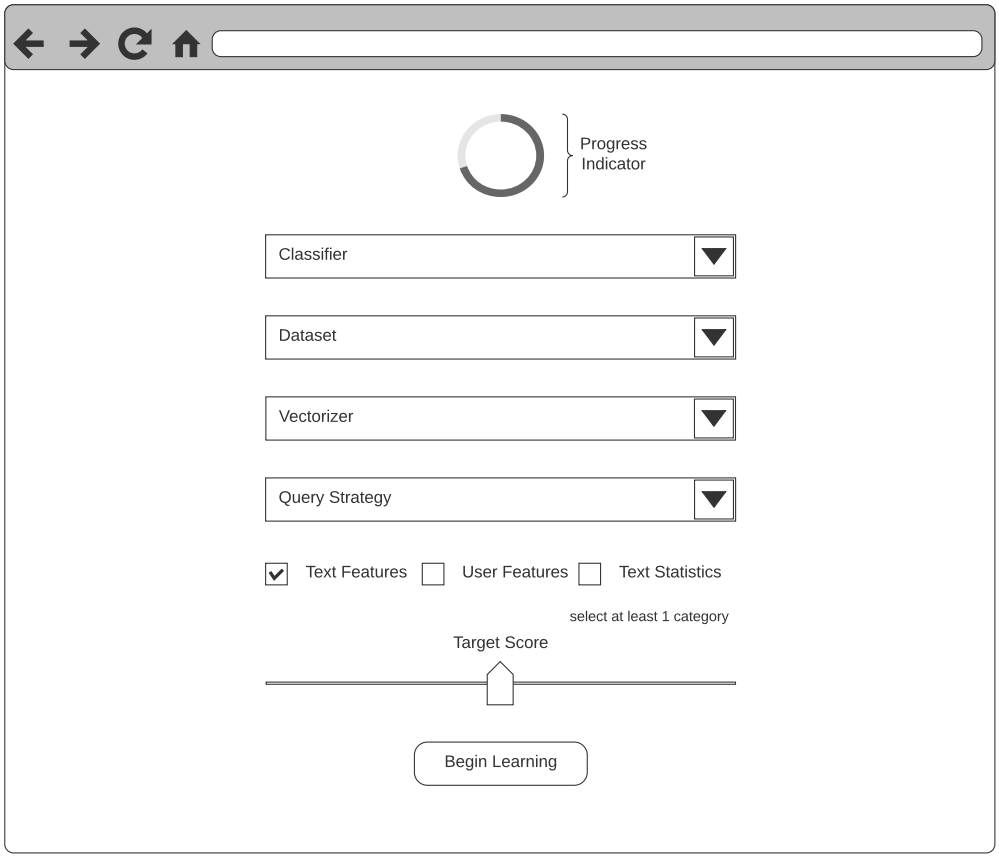


Figure 4: Uninitialised model view for selection of active learning components.

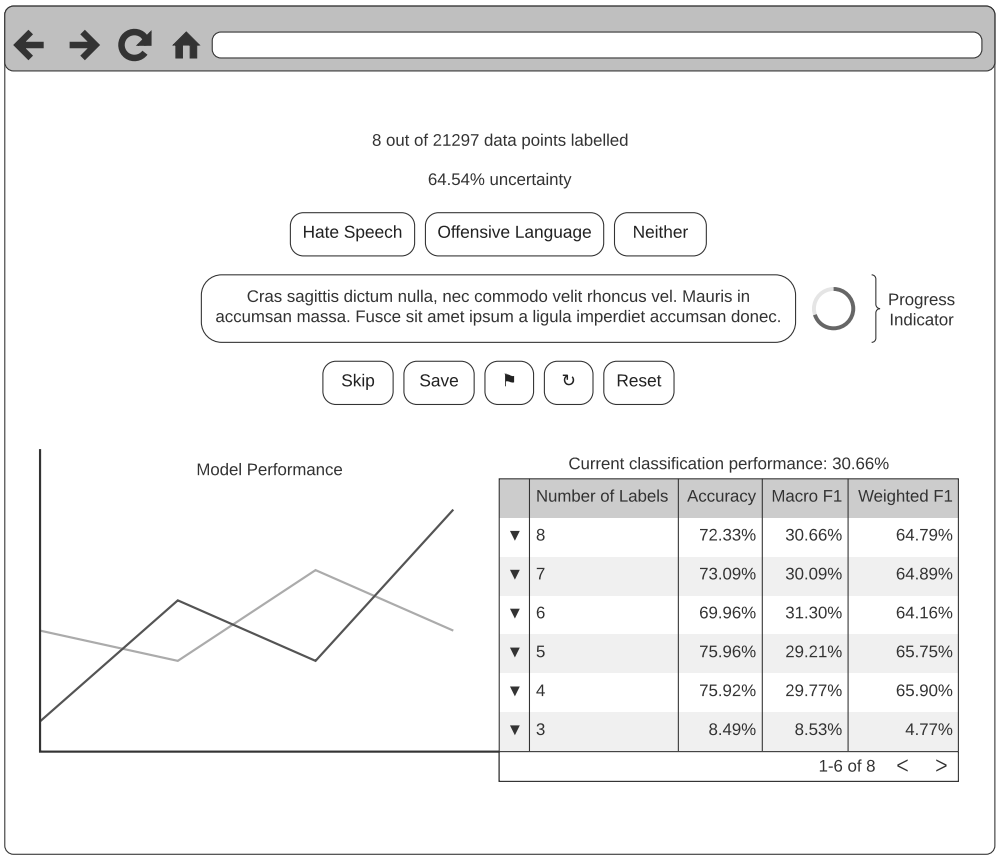


Figure 5: Initialised model view, representing the current state of an active learner.

To facilitate the development of the application, user stories were written to keep track of key software features, gauge their value and prioritise their implementation:

- As a data scientist, I want to interact with the learning process in real-time, so that I can better understand the effect of my choices.
- As a data scientist, I want to skip tweets that make me uncomfortable and those I'm uncertain about, so that I don't introduce personal bias.
- As a data scientist, I want to save my labeling progress, so that I can continue later.
- As a data scientist, I want to save the results and analytics produced during the labeling process, so that I can analyse them later.
- As a data scientist, I want to to adapt the learning process to my preferences, so that it is better suited to my classification problem.

4.3.4 Server-Client Interaction

The server and client components communicate with each other via named events that carry a data payload in JSON format. The payload varies depending on the state of the model, as well as the type of event being sent. However, the range of available message fields is specified in the following list:

- *classifiers* - list of available classification algorithms
- *datasets* - list of available datasets
- *vectorizers* - list of available text vectorization models
- *features* - list of available features in all datasets
- *query_strategies* - list of available sampling strategies
- *idx* - index of the sampled tweet in the unlabeled pool
- *text* - the original tweet text before pre-processing
- *uncertainty* - confidence metric for the tweet produced by the learner
- *series* - list of performance metrics recorded for every iteration
- *labeled_size* - number of labeled examples
- *dataset_size* - total number of examples available for training
- *score* - the current model performance metric
- *target* - the baseline performance target

When a session between the server and the client is first initiated, the server sends all user-defined components available for selection in an *options* event. When the client receives them, they are rendered and presented to the user like shown in Figure 4. After a user selects their desired learning components, the client replies with a matching event, containing the selection.

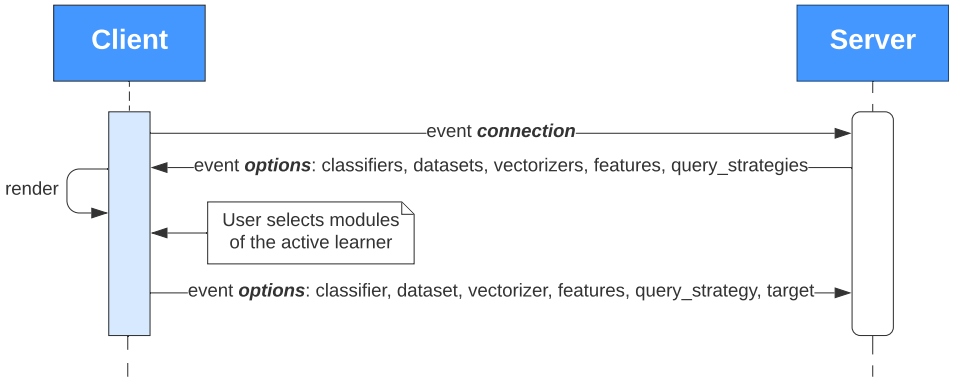


Figure 6: Sequence diagram, depicting the event flow whenever an active learner is not defined and initialised yet.

Alternatively, if the model has been started prior to the client's connection, an *init* event is sent instead. It contains all information regarding the learning process and the model's state. This includes an unlabeled example sampled according to some

query strategy, therefore a user can start annotating data. If the pool of unlabeled data is exhausted, however, an *end* event is sent in place of *init*, which does not include a new sample.

Whenever a user selects what they believe is the appropriate annotation using the graphical interface in Figure 5, a *label* event is transmitted, containing the index of the sample in the unlabeled pool, the selected label value and a hash of the original text. The hash is then used to verify that the index belongs to the appropriate sample server-side.

Subsequent events no longer include fields that are already known to the client as a result of previous interactions, with the idea of reducing network load. Such include *query* and the slightly modified *end* events.

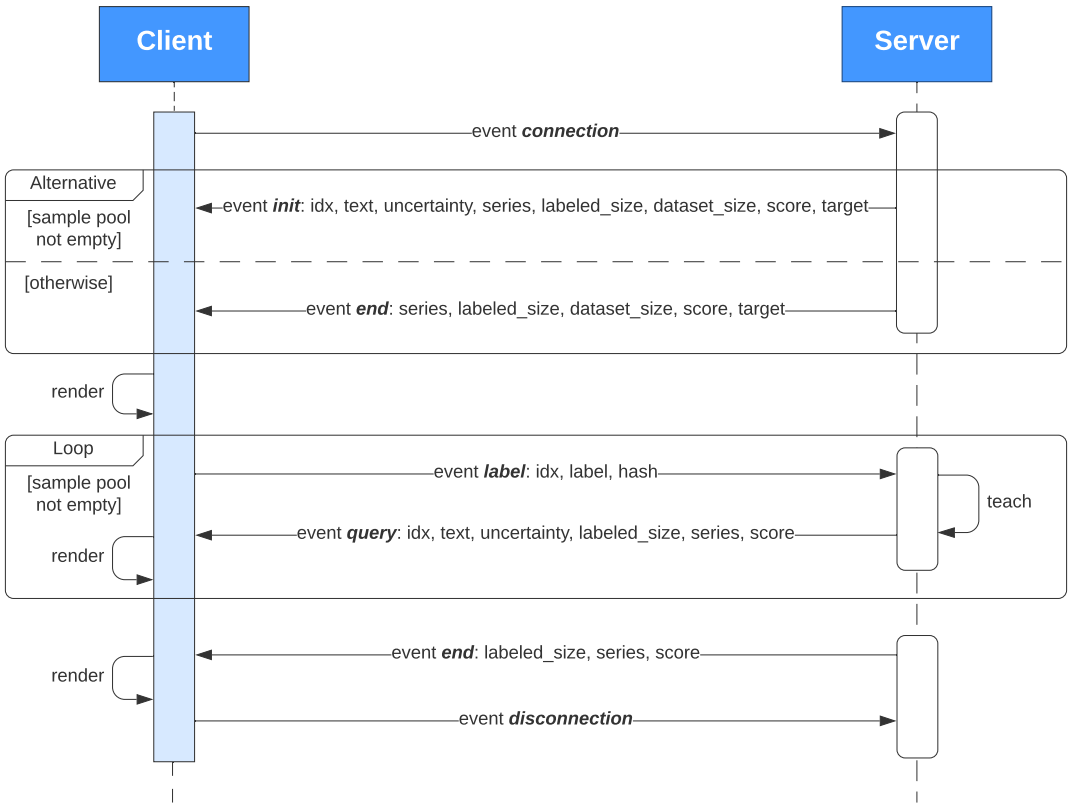


Figure 7: Sequence diagram, depicting the event flow whenever an active learner is initialised and running.

4.3.5 Query Strategies

The choice of query strategy is strongly related to an active learner's performance, as suggested in Section 2.4. All discussed strategies are implemented in this project, in order to evaluate their impact on early performance growth rate.

$$\text{Uncertainty}(x) = 1 - P(\hat{x}|x),$$
where x is a sample and $\hat{x}$ is its most likely prediction.

Figure 8: Uncertainty Sampling

$$\text{Margin}(x) = P(\hat{x}_1|x) - P(\hat{x}_2|x),$$
where $\hat{x}_1$ and $\hat{x}_2$ are the first and second most likely predicted classes for sample x.

Figure 9: Margin Sampling

$$\text{Entropy}(x) = -\sum_k p_k \log(p_k),$$
where p_k is the probability of the sample belonging to the k-th class.

Figure 10: Entropy Sampling

4.3.6 Semi-supervised Learning

To better facilitate the project's evaluation, an alternative approach to querying was developed. In essence, since the author compares both learning techniques on the same dataset, all of the data is already labeled. This makes it possible to simulate the active learning process by first making a copy of the dataset and "forgetting" the labels of say 90% of the data. However, instead of an oracle answering the sampled queries manually, one could simply look up the ground truth labels in the original dataset. While this technique is infeasible in a real world active learning application, it allows for rapid evaluation in a controlled environment. One could essentially switch any module - the classifier, dataset, query strategy, feature selection, feature extraction, etc., and obtain the respective performance metrics automatically, without going through the slow process of labelling. Therefore, a more extensive parameter space could be explored in a shorter amount of time for active learning. Implementation-wise, this was achieved by iteratively sampling the newly unlabeled pool, looking up the label in the original dataset and teaching the classifier n times or until the unlabeled pool is empty. After every sample, the number of labeled instances and F1-score is recorded. Finally, statistical information is extracted from the results, to serve as part of the active learning analysis. Such includes the number of queries necessary to reach the performance of passive learning and the highest labels to performance ratio.

4.4 Model Selection

Support vector machines are an example of a non-probabilistic model that is shown to work well for text classification and does not output confidence metrics by default. In order to adapt it for use with active learning, Platt scaling [25] is implemented, which produces probability estimates from decision values. Nevertheless, there is an inherent drawback to this approach. Active learning requires a model to be re-fit after adding new training data. Hence, the logistic function for Platt scaling needs to be re-learned as well. After a certain amount of queries, this introduces significant overhead, which is undesirable. Despite that, Linear SVM has been implemented and evaluated.

Multinomial Naive Bayes was considered, but since some text vectorizers produce features that allow negative values, and multinomial distributions can't contain negative values, the process was flawed. This is the case with GloVe and Doc2Vec vectorization. Although possible to scale the features to a positive range via min-max normalization, they are intended to be used without further transformations. Multinomial Naive Bayes has therefore not been implemented.

The developed system is abstract enough to support any model that implements "fit" and "predict_proba" methods in its class. This means all *sklearn*[26] classifiers and *sklearn*-esque wrappers are natively supported as components of the learning flow. Regardless, as previously discussed only select few models are important for this project. Although there is an emphasis on naturally probabilistic ones, models that usually perform well on text classification tasks are considered too. Therefore, an *evaluation* script builds and evaluates the following classifiers in both active and passive learning scenarios - Logistic Regression, Linear SVM, Random Forest and K-Nearest Neighbors.

Whenever possible, the models are trained with balanced class weights to account for potential data imbalance. That is, the loss function for the model is altered such that loss on the minority class is penalised more, whereas loss on the majority is penalised less. This is done by adjusting weights inversely proportional to the class frequencies in the dataset. This effect can otherwise be achieved by oversampling the minority class, undersampling the majority class, or both [27]. These alternatives work well with passive learning, as data is already annotated. However, in an active learning scenario, synthetic oversampling would mean that some queries are unintelligible to a human annotator, while undersampling would only limit the pool of data to query from. Oversampling by duplication is viable but after brief experimentation it was not found to be more beneficial than balanced class weights.

5 Evaluation

This paper employs a multi-stranded approach to evaluation, where the benefits of active learning are explored by varying the underlying components that comprise the learning flow - the dataset, features, classifier and sampling strategy. Project success is evaluated in terms of data efficiency gains from applying active learning as opposed to passive learning in a similar context and environment.

5.1 Critical Analysis

While the dataset collected was sampled in a way that suggests a well-proportioned distribution, after labeling all tweets, a considerable class imbalance was observed. Specifically, a ratio of 468 to 1532 occurs in the dataset, with malicious tweets as the minority class.

Figure 11: Most common words in the non-malicious class.

Figure 12: Most common words in the malicious class.

Although this may be closer to how tweets are expected to naturally appear in Twitter, training a model on an imbalanced dataset is bound to introduce bias. Malicious tweets being the minority class means that micro F1 is not an appropriate metric for evaluation, as classifiers tend to be biased towards the majority class. In this scenario, a model would most likely be able to predict non-malicious tweets well, but fail on the more important malicious examples. Nonetheless, the overall performance would still remain high. To account for this phenomena, macro F1 will be used instead, as it factors in the performance on both classes equally.

$$\text{Macro F1} = \frac{1}{N}\sum_{i=0}^{N} \text{F1}_i,$$

where i is the class index and N the number of classes

As a result, trained classifiers will not look overly optimistic because of their performance on the non-malicious majority class. In the following table, macro F1 is denoted as *F1*. Because detecting malicious tweets is not a multi-label classification problem, micro F1 is equivalent to accuracy, denoted *Acc*. Finally, P and R stand for the macro-averaged precision and recall respectively.

T - Text Features | U - User Features | S - Statistical Features

	Features	Logistic Regression				Linear SVM			
		F1	P	R	Acc	F1	P	R	Acc
TF-IDF	T	**71.91**	**69.98**	**76.81**	79.68	62.64	62.51	68.91	70.31
	T-U	71.47	69.62	75.74	79.68	63.51	63.34	70.29	70.83
	T-S	68.01	66.75	74.25	75.52	64.84	64.25	71.26	72.39
	T-U-S	67.54	66.40	73.93	75.00	65.28	64.57	71.58	72.91
GloVe	T	58.08	58.20	61.96	67.70	59.02	59.08	63.35	68.22
	T-U	59.94	59.95	64.74	68.75	60.36	60.24	65.06	69.27
	T-S	61.77	62.23	69.01	68.75	62.20	62.22	68.58	69.79
	T-U-S	62.20	62.50	69.33	69.27	62.20	62.22	68.58	69.79
Doc2Vec	T	49.09	55.09	58.33	53.12	42.20	57.89	60.47	42.70
	T-U	43.91	57.57	60.68	44.79	37.93	57.58	58.65	38.02
	T-S	58.01	62.30	70.19	61.97	50.12	56.63	60.79	53.64
	T-U-S	56.44	61.07	68.16	60.41	52.90	58.42	63.78	56.77
	Features	Random Forest				K-Nearest Neighbors			
		F1	P	R	Acc	F1	P	R	Acc
TF-IDF	T	69.34	68.80	69.97	**80.72**	57.73	63.63	56.83	80.20
	T-U	64.78	63.74	69.01	73.95	62.03	63.32	61.21	78.64
	T-S	68.53	67.07	71.58	78.12	54.43	54.32	54.59	71.35
	T-U-S	64.16	63.11	67.20	74.47	51.58	51.67	52.02	67.18
GloVe	T	59.40	59.15	59.72	74.47	53.34	53.23	53.63	69.79
	T-U	61.49	61.27	61.75	76.04	53.91	53.85	54.80	68.22
	T-S	62.93	63.82	62.28	78.64	58.56	58.47	58.65	74.47
	T-U-S	59.32	59.97	58.86	76.56	50.10	50.10	50.10	69.27
Doc2Vec	T	51.06	53.40	51.70	77.08	54.95	54.76	55.34	70.83
	T-U	47.50	47.38	47.86	70.83	51.39	51.89	52.56	64.58
	T-S	49.41	51.44	50.64	77.08	52.63	52.57	52.99	68.75
	T-U-S	58.51	60.71	57.69	78.12	55.42	55.22	56.51	69.27

Table 2: Baseline performance scores of machine learning for all combinations of classifier, vectorizer and features.

A few observations can be made following these results. First, TF-IDF is dominant across the board and produces the highest scores for every model class evaluated. The GloVe word embeddings model produces mediocre results, despite having been trained on the largest amount of data. Perhaps tweet similarity isn't a very indicative feature for classifying tweets. This is explicable by the fact that both malicious and non-malicious tweets have large variance when it comes to syntax and semantics. The classifiers must have not been able to pick up on a discriminative pattern due to the limited amount of training data available. Doc2Vec clearly suffered from the same issue. Unlike GloVe which is pre-trained, Doc2Vec was fit on the local dataset used for classification, which is rather small. This further handicaps the performance, making it the worst vectorizer in all explored scenarios.

Second, performance scores when using TF-IDF mostly decline with the addition of statistical and user features. In contrast, adding new features when using any of the other vectorizers tends to increase scores. Taking the extremes for example, with a Linear SVM and Doc2Vec, adding extra features increased F1-scores by up to 10.7%, while the combination of Logistic Regression and TF-IDF suffered up to a 4.37% decrease. This observation signifies that TF-IDF features are very discriminative, to the extent that their sole use offers the highest information gain. This confirms the findings of Rosa H. et al.[5] that feature engineering beyond TF-IDF is not rewarding.

To reduce the complexity of visualising active learning applied to all combinations of classifier, vectorizer and features, only the highest baseline results are plotted and analysed further.

From Table 2 one can distinguish the best models for each type of vectorizer in a passive learning scenario. Studying TF-IDF in isolation, Logistic Regression clearly performed best, producing the highest scores not only in its category, but overall. This makes it a prime candidate for further analysis. Active learning with uncertainty as the sampling strategy is applied on all feature combinations to determine the performance growth as training data increases:

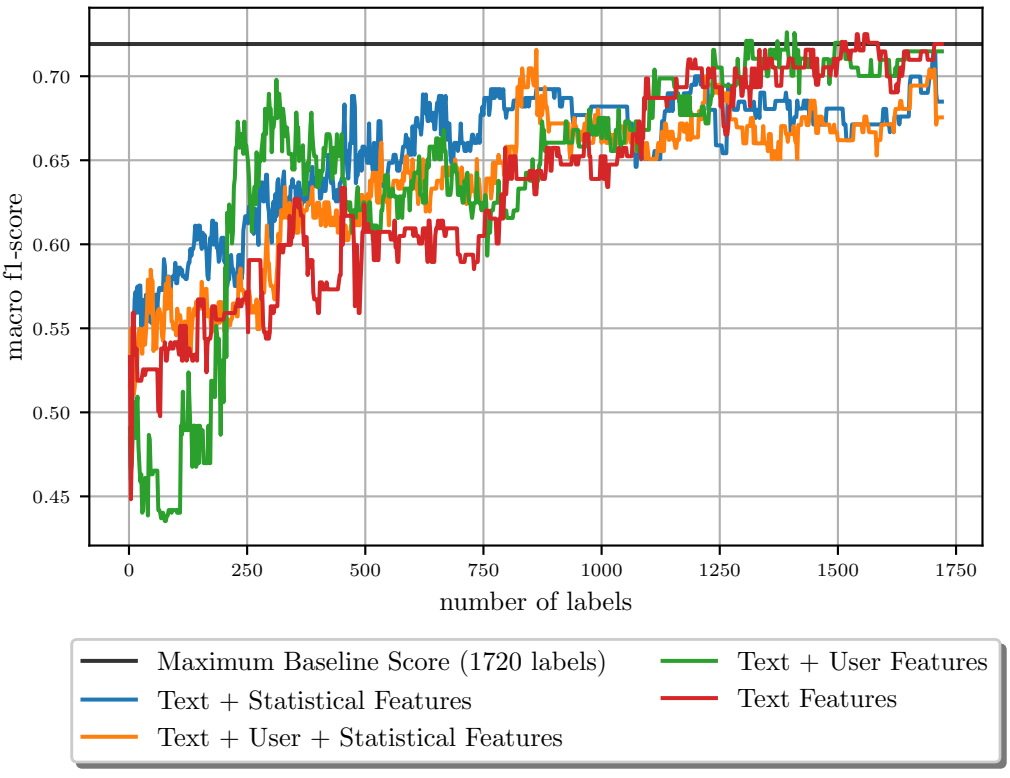


Figure 13: Active learning with Logistic Regression and TF-IDF.

Although it was shown that text features extracted with TF-IDF when used by themselves account for the highest baseline scores, in an active learning scenario they had lesser impact with relatively low performance growth rate. Including user profile features resulted in the model getting stuck in local minima often for the first 200 queries, followed by a rapid performance growth, reaching an F1-score of 69.79% early at 311 labels. In that local maximum, 81.92% less data was used to achieve performance within a 2.35% margin of the combination's original baseline. This is most likely due to outliers in the dataset in terms of user features. Statistical features on the other hand were complemented by a continuous stable growth, despite not peaking as high. Surprisingly, while the combination of all features had the lowest baseline score of 67.54% in a passive learning scenario, it was able to briefly reach a maximum F1 of 71.57% at 862 labels, which is 49.88% less data for a 5.97% increase in its peak performance.

The word embeddings model GloVe performed relatively well with all models except KNN. In combination with Random Forest it produced highest accuracy. However, Linear SVM maintained stable and high F1 and recall metrics in all feature combinations, which as discussed earlier is more relevant. The performance growth is therefore visualised in Figure 14.

Figure 14: Active learning with Linear SVM and word embeddings.

Unlike the case of TF-IDF with Logistic Regression, an earlier and more prominent spike in performance can be observed. In fact, by the 71st label, active learning on the text, user and statistical features combination had already reached its target

baseline, whereas that took at least 455 labels previously. It is plausible that Linear SVM's prediction confidence metrics lead to more optimal queries. However, the other feature combinations were not able to produce similar results, requiring 989 labels at best.

Finally, an example of Doc2Vec's performance growth in an active learning scenario is visualised. It produced the highest baseline results overall, considering all available metrics, in combination with a Random Forest model.

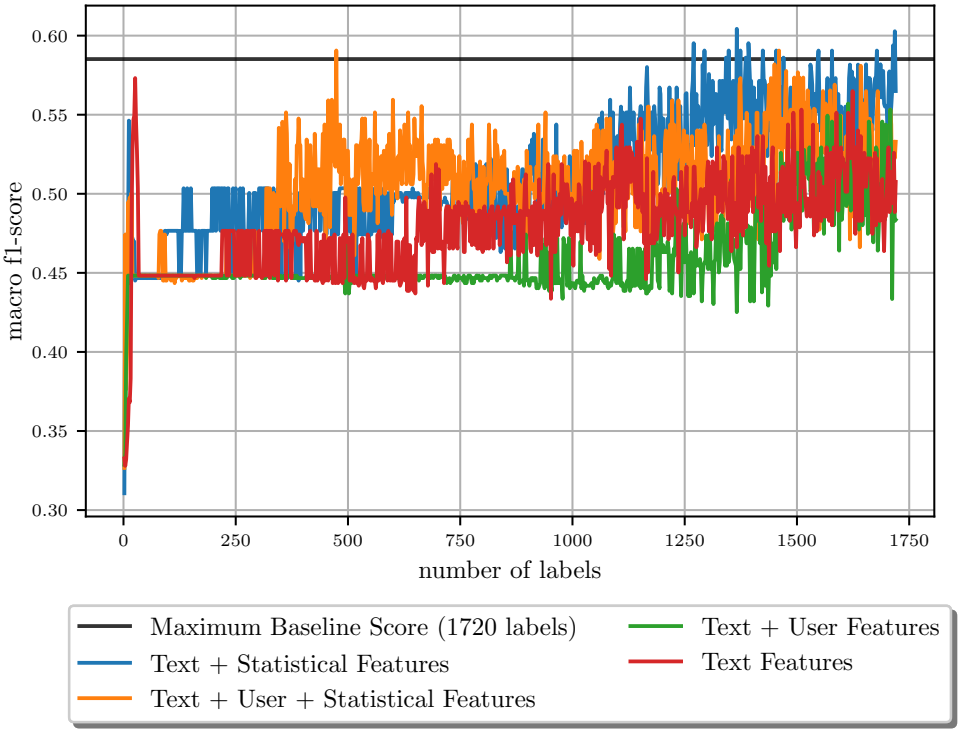


Figure 15: Active learning with Random Forest and document embeddings.

Surprisingly, with approximately 20 trained examples, 2 feature combinations had already reached their baseline. In fact, by using text features only, active learning with Random Forest and Doc2Vec was able to surpass its baseline by 6.26% at mere 26 labels. Shortly after, however, a noticeable drop in growth rate occurred, followed by a discordance that can somewhat be described as noise. When training a Random Forest model iteratively, each iteration comes with randomness in the decision trees that comprise the ensemble forest, which could explain the rather odd pattern.

Since not all combinations can reasonably be explored in detail, the results are concisely presented in Table 3. There, B/L stands for baseline macro F1-score in a passive learning scenario, which corresponds to *F1* in Table 2. *Max* refers to the highest score reached during active learning. Finally, $\#B/L$ and $\#Max$ show the number of labeled examples needed for an active learner with uncertainty sampling to achieve B/L and *Max* respectively.

T - Text Features | U - User Features | S - Statistical Features

	Features	Logistic Regression				Linear SVM			
		B/L	#B/L	Max	#Max	B/L	#B/L	Max	#Max
TF-IDF	T	71.91	1508	72.50	1540	62.64	555	69.87	973
	T-U	71.47	1236	72.62	1392	63.51	192	70.94	992
	T-S	68.01	455	71.41	1702	64.84	73	73.22	932
	T-U-S	67.54	822	71.57	862	65.28	945	68.99	1137
GloVe	T	58.08	66	60.93	998	59.02	989	62.78	1136
	T-U	59.94	809	61.33	1709	60.36	1082	62.40	1500
	T-S	61.77	645	64.72	894	62.20	1065	64.78	1376
	T-U-S	62.20	1339	63.95	1516	62.20	71	63.89	1599
Doc2Vec	T	49.09	7	57.12	66	42.20	5	48.11	144
	T-U	43.91	2	56.71	563	37.93	2	49.53	3
	T-S	58.01	7	62.11	270	50.12	3	58.55	84
	T-U-S	56.44	44	61.35	1121	52.90	28	58.30	28
	Features	Random Forest				K-Nearest Neighbors			
		B/L	#B/L	Max	#Max	B/L	#B/L	Max	#Max
TF-IDF	T	69.34	1359	71.00	1435	57.73	1681	57.74	1681
	T-U	64.78	104	72.65	1449	62.03	165	64.38	165
	T-S	68.53	935	74.39	1371	54.43	3	63.31	139
	T-U-S	64.16	137	74.39	1255	51.58	2	60.20	812
GloVe	T	59.40	1122	63.63	1683	53.34	3	58.14	1149
	T-U	61.49	920	64.26	1194	53.91	7	56.55	7
	T-S	62.93	431	64.98	431	58.56	3	62.27	1095
	T-U-S	59.32	179	66.42	849	50.10	3	57.35	296
Doc2Vec	T	51.06	19	57.32	26	54.95	3	55.94	12
	T-U	47.50	862	55.71	1614	51.39	41	57.37	230
	T-S	49.41	9	60.43	1353	52.63	3	63.59	1193
	T-U-S	58.51	474	59.83	1677	55.42	39	61.47	958

Table 3: Active learning performance improvements over passive baselines for all combinations of classifier, vectorizer and features.

5.2 Comparative Analysis

To reinforce the findings from the small-scale experiment above, the same approach to evaluation is applied on a larger problem. The results of highly-influential work from Thomas Davidson et al. on hate speech detection is used as a reference [18]. The dataset used in their paper is licensed under the MIT License and openly available on GitHub. It is comprised of 24783 tweets, which is approximately 12.4 times more than those collected for this project and used in Section 5.1. Therefore, the following research can be considered an analysis of a large scale problem. The dataset's class distribution is yet again imbalanced, with 1430 examples for hate speech, 19190 for offensive language and 4163 tweets labeled as neither. Naturally, macro F1-score is used as the differential metric in this analysis.

The original paper provides its results after optimisation and in the form of cross validation performance on the entire dataset with 5 splits. However, as discussed in Section 4.2.2, cross validation in active learning can not guarantee equality of the splits, which makes comparisons between active and passive learning sub-optimal. Luckily, the authors of the paper have made available source code for replicating the results on a static 10% hold-out set. Therefore, the results from running the provided implementation are used as a reference. Baselines for passive learning were personally obtained by following the processing steps described in Section 4.2 on the raw dataset. To replicate the actual environment, the combination of Logistic Regression and TF-IDF features was utilised to match the original design. All scores are supported by testing model performance on a held-out set, which is 10% of the entire dataset in volume.

Passive Learning	F1	P	R	Acc
Published Paper CV	77.67	74.00	82.00	90.00
Published Source Code 10%	68.80	66.61	71.62	83.74
Personal T Features 10%	68.78	68.91	68.67	85.00
Personal T+S Features 10%	69.60	69.89	69.36	85.51

Table 4: Passive learning reference scores.

Next, the performance of active learning is compared against the passive baselines. Unlike the problem explored in Section 5.1, this one is defined as multinomial classification on 3 distinct classes. As such, the choice of query strategy should now have a discernible impact on performance growth rate early on.

Active Learning	Query Strategy	B/L	#B/L	Max	#Max
Personal T Features 10%	Uncertainty	68.78	5668	70.40	10796
Personal T Features 10%	Entropy	68.78	7801	70.14	10845
Personal T Features 10%	Margin	68.78	7893	70.17	8739
Personal T+S Features 10%	Uncertainty	69.60	7495	70.44	11361
Personal T+S Features 10%	Entropy	69.60	8848	70.37	9465
Personal T+S Features 10%	Margin	69.60	7500	70.59	10580

Table 5: Active learning improvements over reference scores.

Although there isn't a noticeable improvement in maximum scores, all explored combinations managed to reach their baseline with less than half the amount of labeled tweets. In the best case scenario, only 26.61% of the dataset was used to achieve the same performance as passive learning, whereas in the worst case - 41.54%. To put these results in perspective, evidently it was possible to avoid labeling as many as 15629 tweets, which can be considered a significant reduction to annotation costs. Moreover, all active learners managed to reach the reference hold-out scores obtained from executing the source code of the original research. Since neither cross validation, nor model optimisations were explored per design (Section 4.2.2), the paper's published scores were not reached at any point.

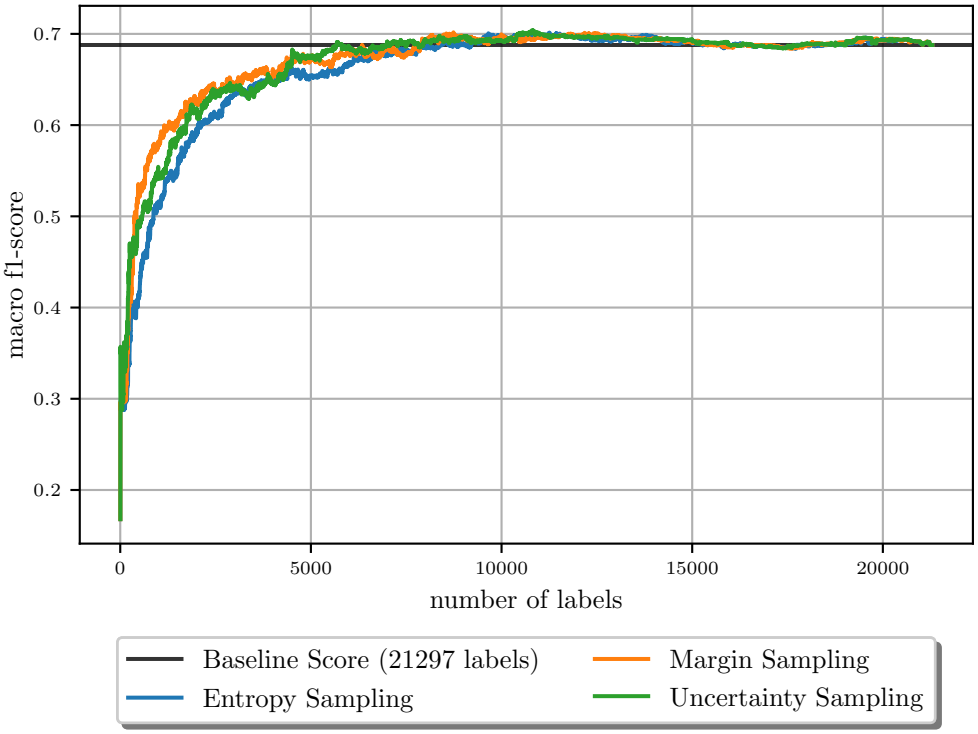


Figure 16: Query strategy macro F1 growth rate differential with Logistic Regression, TF-IDF and text features only.

As expected, there is a subtle difference in the curves of the various sampling strategies. It is most prominent in the first 5000 labels, after which all strategies start to converge. The graph suggests that margin sampling produces better queries with less data, whereas uncertainty sampling performs better later on and is in fact able to reach its target score first. Entropy sampling was the least optimal strategy throughout the learning process.

When adding the more variant statistical features to the feature space, the same correlation could be observed, except that uncertainty sampling had relatively worse initial performance:

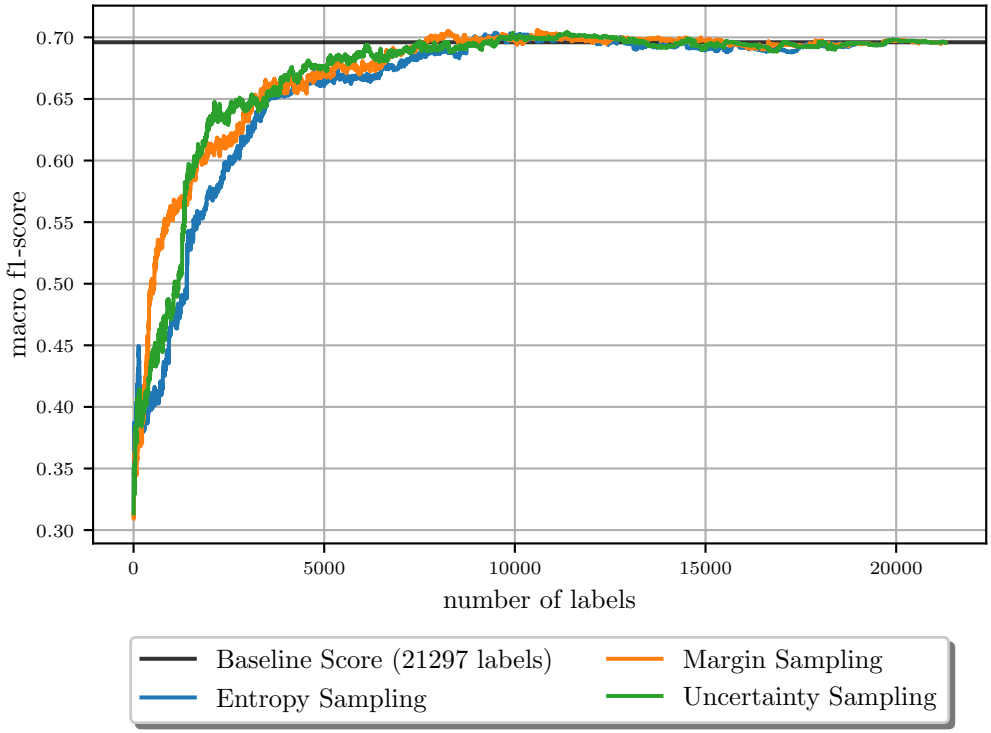


Figure 17: Query strategy macro F1 growth rate differential with Logistic Regression, TF-IDF, text and statistical features.

There doesn't seem to be a large enough statistical significance to determine which query strategy works best in the explored problem. It is likely that the optimal choice of strategy is entirely dependent on the combination of variables discussed - classifier, features and dataset. There is however conclusive evidence, that active learning with any of the discussed query strategies is able to outperform passive learning in terms of data efficiency, and sometimes even in terms of generalisation power.

5.3 Testing

In order to minimise long-term risk and loss from bugs introduced by incorrect code, an extensive test suite was implemented in the early stages of development. As new software features were added, the scope of tests was appropriately extended as well. A phased approach was adopted to ensure low-level errors are detected as early as possible. First, modules are tested in isolation to evaluate their core functionality, e.g. whether the preprocessing module works on data with conflicting attributes. Then, bottom-up integration testing is performed on modules with dependencies to evaluate their interoperability, e.g. whether the learning module correctly vectorizes tweets by utilising the features module. All tests are aggregated according to the functionality being tested. The test suite was ran after each major feature implementation, as well as after code refactoring, to ensure that system correctness is preserved. The final product passes all tests, which is evident in the following sample output:

```
test_1_preprocess (__main__.TestLearning) ... ok
test_2_learn_vectorizer (__main__.TestLearning) ... ok
test_3_split_ml (__main__.TestLearning) ... ok
test_4_build_features (__main__.TestLearning) ... ok
test_5_fit (__main__.TestLearning) ... ok
test_6_split_al (__main__.TestLearningAL) ... ok
test_7_teach (__main__.TestLearningAL) ... ok
test_8_skip (__main__.TestLearningAL) ... ok
test_9_auto_teach (__main__.TestLearningAL) ... ok
----------------------------------------------------------------------
Ran 9 tests in 11.695s
OK
```

The software was also further tested on its interaction with the web application. Due to the asynchronous design of communication between components in the form of web sockets, all tests were performed informally by simulating different use case scenarios. Finally, system testing and validation were performed to evaluate whether the system meets all formally defined requirements in Section 3.1, properly implements informally described features in Section 4.3.3, and offers an appropriate level of usability and accessibility.

Earlier it was hypothesised that tweet similarity could be directly correlated to how the unlabeled pool is queried. That is, a classifier would be most uncertain for examples that are furthest away in vector space. To examine whether this is indeed the case, a minimal working example was constructed in the form of an dummy active learner with only word embeddings as its feature set. The parameters used for initialising the model were Logistic Regression for classifier, GloVe for vectorizer and Uncertainty Sampling for query strategy. The model was taught only a single tweet which would serve as the reference vector. Ideally, the most dissimilar tweets vector-wise would be some of the top candidates queried for labeling. Then, cosine similarity was calculated pairwise between the seen example and the entire unlabeled pool. The

indices of the top 20 most dissimilar tweets were extracted and compared against those of the top 20 tweets with highest uncertainty according to the classifier:

- Least cosine similar tweets to the trained example
  [821, 809, 1384, 661, 1534, 1159, 671, 674, 1163, 977,
  177, 968, 1645, 73, 192, 1633, 1488, 201, 1211, 472]
- Most uncertain tweets w/ uncertainty sampling
  [674, 1372, 1650, 915, 755, 1337, 201, 590, 1645, 719,
  1488, 1068, 871, 1165, 215, 216, 89, 220, 1385, 73]
- Intersection
  [73, 201, 674, 1488, 1645]

These findings support the proposed hypothesis. Out of 1719 available tweets in the unlabeled pool, cosine similarity and uncertainty agreed on 5 from the top 20 sampled according to each metric.

6 Conclusions

Applying active learning in the context of social media text classification was shown to be beneficial, and more often than not results in significantly better data to performance ratios. In fact, in most cases active learning exceeded the baseline scores of passive learning. It was also found that the choice of model and feature extraction/selection techniques are directly correlated to the optimality of samples selected by the query strategy. The choice of query strategy however did not have a large differential - all strategies converged at the baseline ML scores at roughly the same number of labeled examples. That being said, margin sampling tended to produce more informative queries with less data. It is also evident that having a lot of labeled data is not necessarily suggestive of higher classification accuracy. Instead, allowing a model to choose from a large unlabeled pool can be equally effective, but a lot more efficient. Naturally, this makes active learning ideal for social media classification, especially in scenarios where transfer learning is inapplicable, as is often the case with novel media formats.

7 Future work

Because this project necessitates a pool-based sampling approach in order to maintain consistency of the data available in different scenarios, the overall performance gains of active learning are more or less capped by the pool's size, data distribution and quality. When applying active learning outside a controlled environment, this limitation can be overcome by sampling the data from a stream instead of a pool. Doing so both expands the sample space which hence lifts the performance cap, and leverages the abundance of unlabeled data better by lowering the cost per query. In such scenario, uninformative data examples are not stored and continuously re-evaluated with each query, which as discussed earlier is a costly operation. While out of the scope of the project, this author hypothesises that a stream sampling approach could perform better than a pool-based one in the context of social media classification and needs to be explored further.

Evidence was produced that supports the hypothesis that a correlation between uncertainty and tweet similarity exists. To further leverage the information gain from features produced with text models like GloVe, Word2Vec or Doc2Vec, a novel query strategy for active learning is proposed. Instead of sampling data according to a specific classifier's predictions, word/document similarity could be used as a heuristic that is applicable in general, regardless of model class. This could be especially useful for big data, where the query strategies discussed in this project are simply ineffective.

Finally, in Section 4.3, the idea of scalability was discussed. Ranked batch-mode sampling is a natural progression of the interactive sampling implemented. As such, a focus for future work is the introduction of query strategies that generate multiple queries, which would allow annotators to label the same dataset simultaneously.

8 Project Planning

8.1 Work Estimate

An approximation of the time required to reach individual milestones was made in order to better schedule and plan for project deliverables. Each project goal defined in the beginning of this paper corresponds to a major task, with sub-tasks planned in-between to demonstrate progress.

Task	Time
Twitter Crawler	3 weeks*
Data Mining	1 week
Data Analysis	2 weeks
Web Application	2 weeks
Active Learning	2 weeks
Passive Learning	2 weeks
Design Refinement	2 weeks
Active Learning Analysis	2 weeks
Comparative Analysis	2 weeks
Project Refinement	10 days

Table 6: Rough estimate of the time needed to complete project tasks.

8.2 Iterations

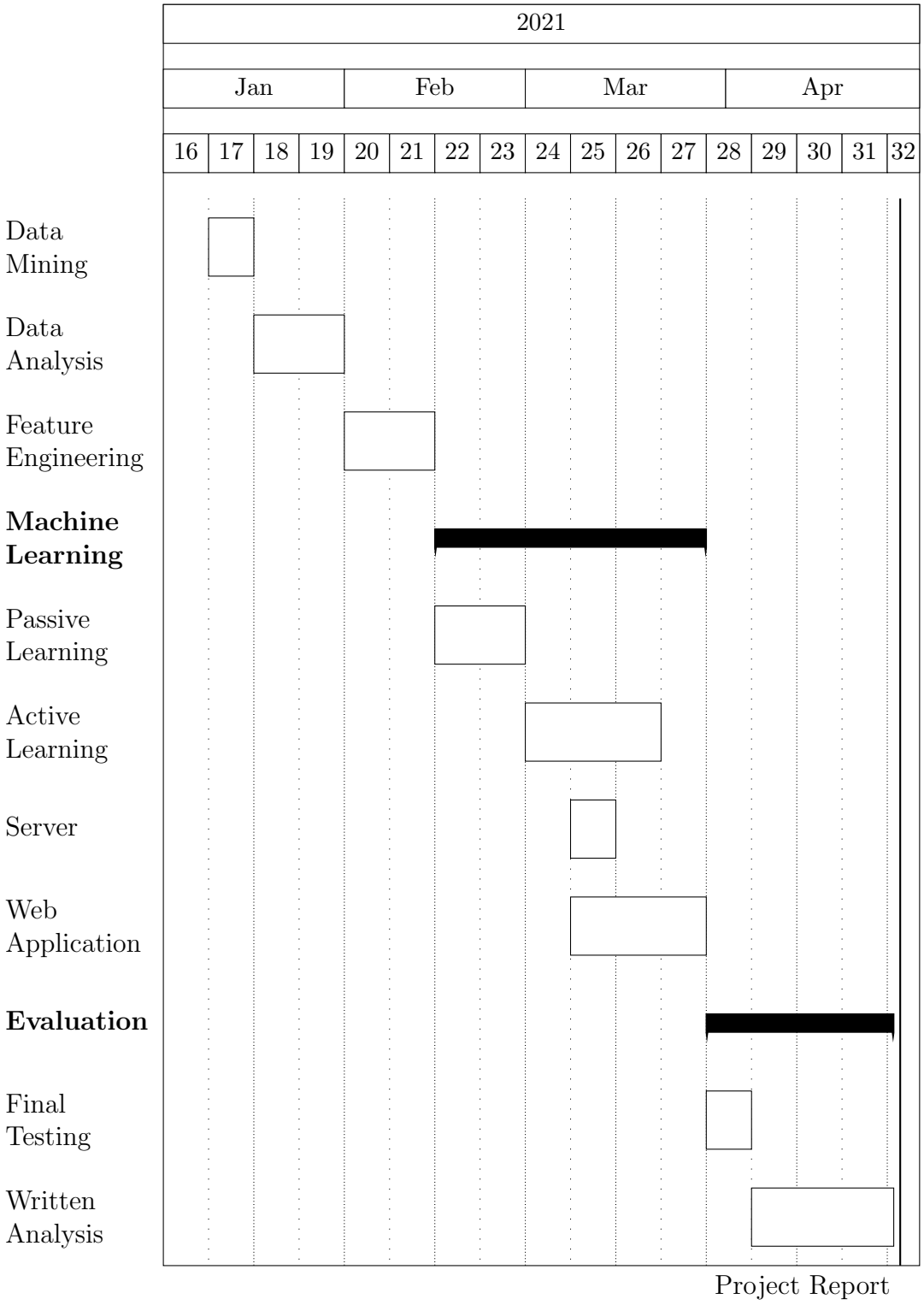
The project has been split into tasks that take approximately a week to prototype or implement. However, two weeks have been allocated per task, as this allows for phased iterative development, where deliverables go through refinement individually. That is, after each component is completed, there is a period of time reserved for the sole purpose of improving the design and implementation, before moving on to the next one. As deliverables depend on the success of their predecessor, this approach ensures the project is always moving forward smoothly. Iteration deliverables and their respective scheduling is visualised in the following section.

8.3 Project Schedule

Initially, the tasks identified earlier were temporally mapped in a sequential manner to a Gantt chart, which can be found in Appendix B. However, as the project was being developed, it became increasingly apparent that due to a lack of experience and knowledge, some tasks had been underestimated. Moreover, the scheduled order for completion did not correspond to the tasks' logical order. For example, active learning came before passive learning, when in fact the former is a natural extension

*Accounts for winter term break and the subsequent exams.

of the latter. At a certain point, the tasks were re-evaluated and an appropriate Gantt chart was plotted. The new chart ended up being an accurate estimate of the project and its deliverables, and in hindsight managed to capture the actual progress made at any given time:



8.4 Tools

The following tools will be utilised as a means of speeding up the development process, scheduling, keeping track of progress and report writing:

Tool	Description	Benefit
Git	Version Control	Tracking of changes during development
Mendeley	Reference Manager	Collection of research papers and citations
Notion	Notes & Workflow	Project management and progress tracking
ECS GPU	Compute Service	High availability, efficient machine learning
Overleaf	LaTeX Editor	Project report sync between devices
Lucidchart	Charts & Diagrams	Design sketching and visualisation

8.5 Risk Assessment

PROBLEM: Lack of experience with machine learning technologies		
LOSS: 3	**PROBABILITY**: 2	**RISK**: 6
PLAN: Enrolled in two machine learning related modules throughout the academic year, which alongside personal reading and experimentation should provide the necessary background to implement this project.		

PROBLEM: Overambitious deliverables - unable to finish task in the planned time frame		
LOSS: 4	**PROBABILITY**: 2	**RISK**: 8
PLAN: The iterative approach discussed earlier allows for up to a week of time after the initial completion of a task and is used for refinement. However, if necessary, this time can be reallocated to completing the problematic task and skipping the iterative process with no significant consequences. This generally shouldn't happen as work is scheduled according to personal capabilities.		

PROBLEM: Data labeling takes too much time, staggered progress and productivity		
LOSS: 5	**PROBABILITY**: 3	**RISK**: 15
PLAN: Data labeling will be outsourced to external party if necessary or the planned dataset size will be reduced to accommodate the loss of time.		

PROBLEM: Illness or otherwise reduced ability to work		
LOSS:2	**PROBABILITY**: 2	**RISK**: 4
PLAN: Similar to the risk of overambitious deliverables, time that would usually be spent refining completed tasks can be reallocated for emergencies with no major setbacks.		

References

[1] B. Settles, "Active learning literature survey," Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[2] M. Dadvar, D. Trieschnigg, R. Ordelman, and F. D. Jong, "Improving cyberbullying detection with user context," vol. 7814 LNCS, pp. 693–696, Springer, Berlin, Heidelberg, 2013.

[3] M. A. Al-Garadi, K. D. Varathan, and S. D. Ravana, "Cybercrime detection in online communications: The experimental case of cyberbullying detection in the twitter network," *Computers in Human Behavior*, vol. 63, pp. 433–443, 10 2016.

[4] V. Balakrishnan, S. Khan, and H. R. Arabnia, "Improving cyberbullying detection using twitter users' psychological features and machine learning," *Computers and Security*, vol. 90, p. 101710, 3 2020.

[5] H. Rosa, N. Pereira, R. Ribeiro, P. C. Ferreira, J. P. Carvalho, S. Oliveira, L. Coheur, P. Paulino, A. M. V. Simão, and I. Trancoso, "Automatic cyberbullying detection: A systematic review," *Computers in Human Behavior*, vol. 93, pp. 333–345, 4 2019.

[6] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," 2015.

[7] J. Moon, J. Kim, Y. Shin, and S. Hwang, "Confidence-aware learning for deep neural networks," *arXiv*, 7 2020.

[8] J. Kremer, K. S. Pedersen, and C. Igel, "Active learning with support vector machines," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, pp. 313–326, 7 2014.

[9] R. Schumann and I. Rehbein, "Active learning via membership query synthesis for semi-supervised sentence classification," in *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, (Hong Kong, China), pp. 472–481, Association for Computational Linguistics, Nov. 2019.

[10] B. Settles and M. Craven, "An analysis of active learning strategies for sequence labeling tasks," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, (USA), p. 1070–1079, Association for Computational Linguistics, 2008.

[11] T. Scheffer, C. Decomain, and S. Wrobel, "Active hidden markov models for information extraction," in *Proceedings of the 4th International Conference on Advances in Intelligent Data Analysis*, IDA '01, (Berlin, Heidelberg), p. 309–318, Springer-Verlag, 2001.

[12] A. I. Schein and L. H. Ungar, "Active learning for logistic regression: an evaluation," *Machine Learning*, vol. 68, pp. 235–265, Oct 2007.

[13] P. Kumar and A. Gupta, "Active learning query strategies for classification, regression, and clustering: A survey," *Journal of Computer Science and Technology*, vol. 35, pp. 913–945, Jul 2020.

[14] R. Zhao, A. Zhou, and K. Mao, "Automatic detection of cyberbullying on social networks based on bullying features," vol. 04-07-January-2016, pp. 1–6, Association for Computing Machinery, 1 2016.

[15] V. K. Singh, Q. Huang, and P. K. Atrey, "Cyberbullying detection using probabilistic socio-textual information fusion," pp. 884–887, Institute of Electrical and Electronics Engineers Inc., 11 2016.

[16] R. Zhao and K. Mao, "Cyberbullying detection based on semantic-enhanced marginalized denoising auto-encoder," *IEEE Transactions on Affective Computing*, vol. 8, pp. 328–339, 7 2017.

[17] Y. Xu and R. Goodacre, "On splitting training and validation set: A comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning," *Journal of Analysis and Testing*, vol. 2, pp. 249–262, Jul 2018.

[18] T. Davidson, D. Warmsley, M. Macy, and I. Weber, "Automated hate speech detection and the problem of offensive language," in *Proceedings of the 11th International AAAI Conference on Web and Social Media*, ICWSM '17, pp. 512–515, 2017.

[19] T. Singh and M. Kumari, "Role of text pre-processing in twitter sentiment analysis," *Procedia Computer Science*, vol. 89, pp. 549–554, 2016. Twelfth International Conference on Communication Networks, ICCN 2016, August 19– 21, 2016, Bangalore, India Twelfth International Conference on Data Mining and Warehousing, ICDMW 2016, August 19-21, 2016, Bangalore, India Twelfth International Conference on Image and Signal Processing, ICISP 2016, August 19-21, 2016, Bangalore, India.

[20] A. B. Boot, E. Tjong Kim Sang, K. Dijkstra, and R. A. Zwaan, "How character limit affects language usage in tweets," *Palgrave Communications*, vol. 5, p. 76, Jul 2019.

[21] O. Levy and Y. Goldberg, "Linguistic regularities in sparse and explicit word representations," in *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, (Ann Arbor, Michigan), pp. 171–180, Association for Computational Linguistics, June 2014.

[22] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

[23] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[24] T. Danka and P. Horvath, "modAL: A modular active learning framework for Python," available on arXiv at https://arxiv.org/abs/1805.00979.

[25] J. C. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," in *ADVANCES IN LARGE MARGIN CLASSIFIERS*, pp. 61–74, MIT Press, 1999.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[27] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

A Original Project Brief

Background

With social media being crucial for maintaining social status now more than ever, the notion of "cancel culture" has become increasingly prevalent. Malicious people aim to "cancel" or deplatform others in the form of online shaming.

Problem

This project explores the efficacy of Active Learning in the context of social media post classification, compared to alternative Machine Learning techniques. Specifically, the project aims to show how applying Activate Learning in natural language processing can reduce data labeling costs and improve data efficiency.

Goals

- To compile an appropriate data set for Machine Learning using Twitter's API, feature selection and feature engineering.
- To build an application that interactively queries the user to label data.
- To apply Active Learning to train a model for classifying Twitter posts.
- To produce an in-depth analysis of the characteristics and outcomes of applying Active Learning.
- To compare Active Learning to other Machine Learning approaches in regards to data efficiency.

Scope

While the methods and outcomes of this project could be generalised to all social media, this project focuses on Twitter specifically. The project does not necessarily aim to produce a highly accurate model for post classification, but rather show data cost in relation to F_1-score of models trained using various Machine Learning techniques.

B Planned Schedule

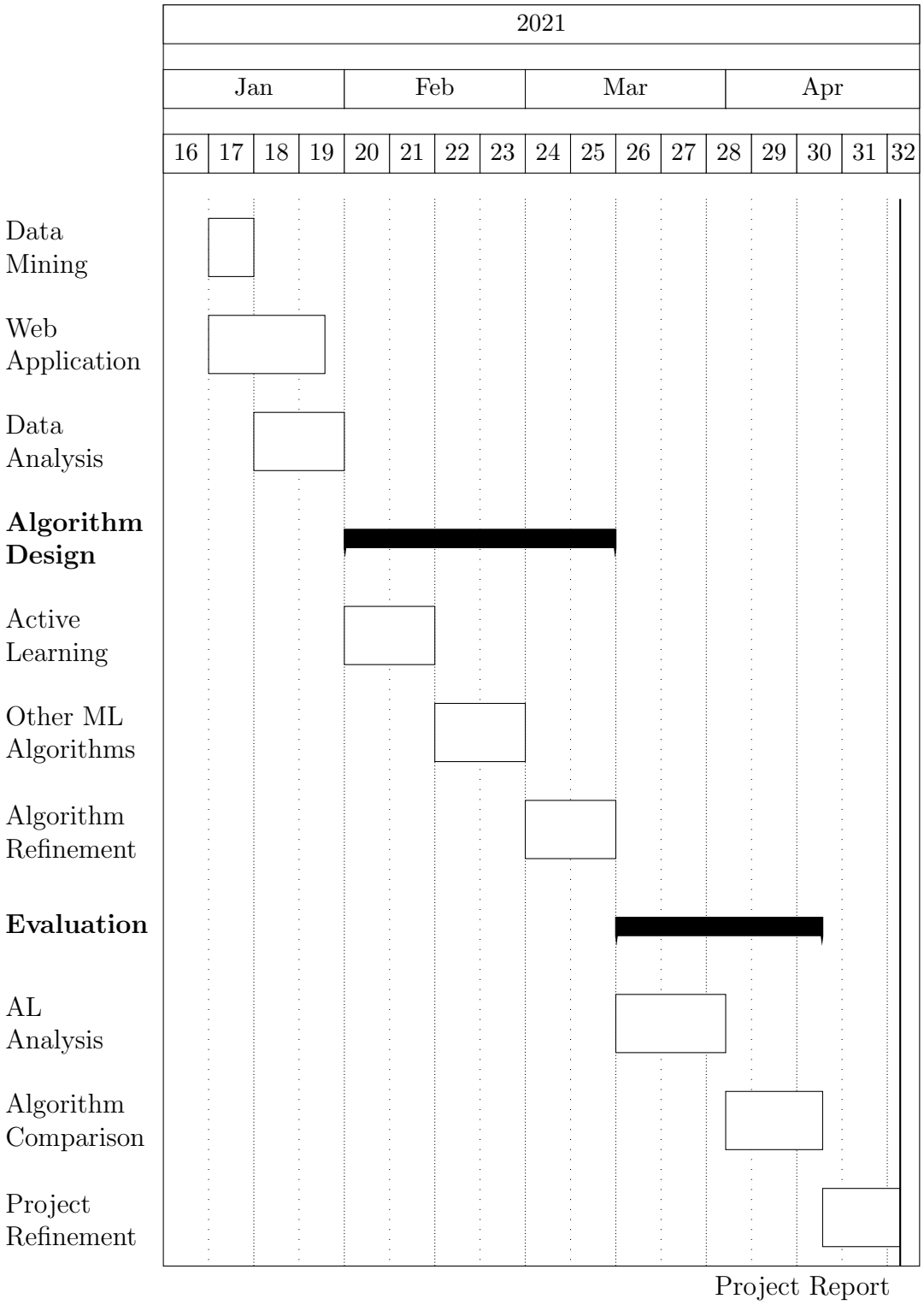


Figure 18: Term II planned project schedule.

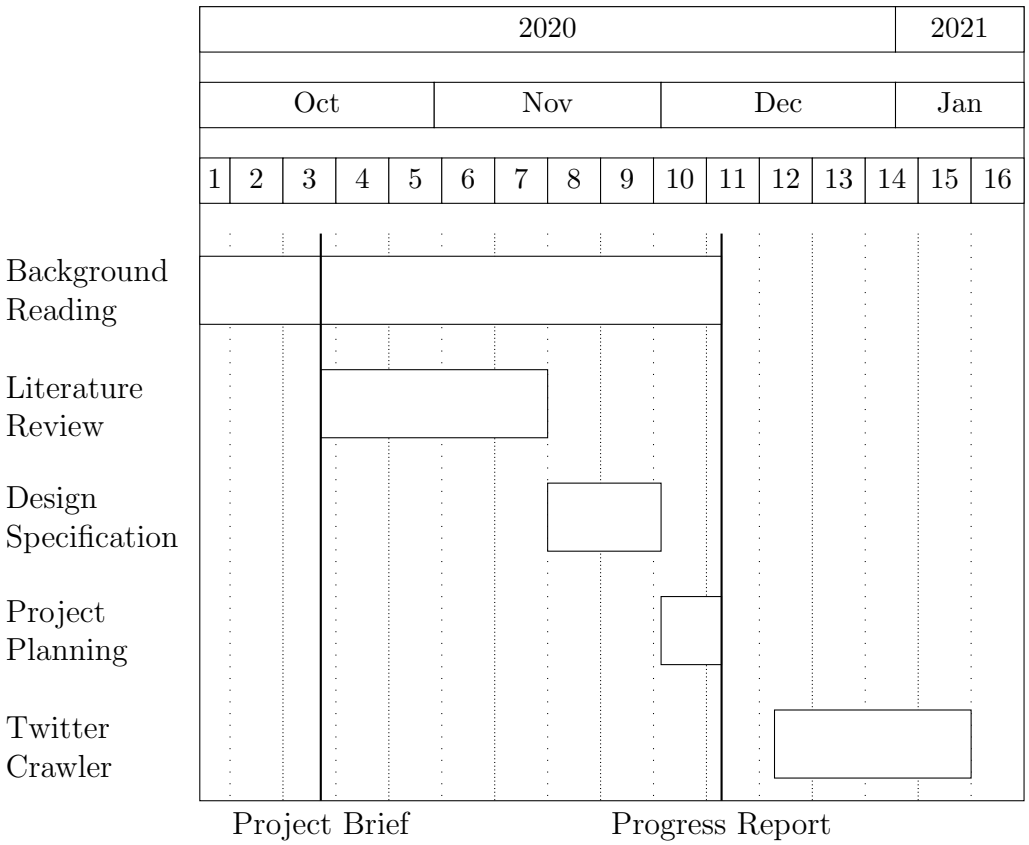


Figure 19: Term I planned project schedule.