An Active Learning System for Text Classification

Sean Floyd, Mark Kasun, and Rayan Safieddine, *

University of Ottawa, Ottawa, Canada

Abstract. "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" [4]. However, implementations in this area of research are sparse.
In this paper, we will present the active learning system we implemented. This system is geared towards text classification and is general enough to classify different types of data. Our system supports various types of active learning strategies; this paper also provides an analysis of these strategies on our datasets.

1 Introduction

Sources of interesting text documents are plenty, especially with the internet, but they typically do not come with labels. We wanted to make a system that would predict labels for documents while requiring a domain expert to spend as little time as possible labelling initial documents. This system would be useful in many areas such as labelling email as spam, or finding news articles about security breaches.

A combination of supervised and semi-supervised learning techniques were used to build our system. More specifically, we leverage support vector machines and active learning to build an efficient system for text classification.

2 Background and Related Work

Our system is built from a combination of supervised and semi-supervised learning techniques. We use a linear support vector machine as our classifier and try out multiple active learning strategies to provide accurate classification while requiring an oracle to spend as little time as possible labelling documents.

2.1 Supervised Learning

Supervised learning is the traditional method for building classifiers. With supervised learning, all of the training data must be accompanied by their labels. The labels and the data are used to train a classifier which is then used to predict the

* Herna Viktor

label of new instances fed into the system. With traditional supervised learning, a large amount of labelled data is required to initially train the classifier if you want accurate predictions.

Support Vector Machines (SVM). SVMs are a well-studied supervised learning technique. SVMs attempt to find a hyperplane that separates the classes. There are always infinitely many hyperplanes to choose from that separate the classes but SVMs attempt to find a hyperplane that creates the widest gap between the classes so that any new instances are likely to fall on one side or the other, even if they exist closer to the hyperplane than the items from the training set. Training items that lie on the edge of the gap are used as support vectors for finding the best hyperplane. SVMs are useful for text classification as they are dimensionality-independent [3]. Text documents can have a significant number of features since each individual word can be considered a feature and representing a document must consider all the different words in the entire corpus of documents. We specifically chose to use a linear SVM as non-linear SVMs have higher generalization error when dealing with smaller pools of labelled data and we're specifically dealing with small pools [3].

2.2 Semi-Supervised Learning

Semi-supervised learning techniques try to generate accurate classifiers with significantly less labelled data than normal supervised learning techniques. These techniques include self-training, co-training, multi-view learning, and active learning [4]. Most of these techniques involve using the classifier or different models to train itself and thus use less data. Active learning is different in that it is a method for getting newly labelled data. The techniques are not mutually exclusive and an interesting area of research is to attempt to combine the multiple techniques.

Active Learning. When building a classifier, labelled data is necessary. Since most data does not come with labels, an expert (or oracle) is needed to provide the initial training labels. Labels and oracle time are both valuable when training classifiers. The goal of active learning is to find items to be labelled that are more useful to producing an accurate classifier so that the oracle can spend significantly less time providing labels for training data. Instead of labelling a lot of data initially, the system starts off with a very small pool of labelled data. A poor classifier is trained on this small pool and an item is taken from the unlabelled pool that will help produce a better classifier if it were labelled. This item is given to an oracle to provide the correct label for it and then it is added to the labelled training pool. The classifier is then re-trained with the labelled data that includes the newly added one. This cycle repeats until the oracle is satisfied with the classifier that has been produced or the classifier has completed a pre-set number of cycles. With the right selection technique of items from the unlabelled pool, significantly less labelled data is needed to build an accurate classifier when compared with traditional supervised learning techniques.

2.3 Active Learning Strategies

For our project, we tested multiple active learning strategies for text classification and compared their results.

Random Sampling. This strategy is employed to create a baseline for how a normal supervised learning strategy would perform if it was given the same amount of labelled data. Ideally, this would be the worst method of active learning as any other method that performed worse than random sampling would not be using an effective selection technique for new labels.

Uncertainty Sampling. This strategy is the simplest and most commonly used [4]. In our trials, it appeared to work extremely well for text classification problems. Uncertainty sampling simply asks the classifier for its predictions on the unlabelled pool and its confidence in those predictions. The item with the least confidence is then given to the oracle to provide a label for it. For support vector machines, unlabelled items that exist close to the current hyperplane of the trained classifier are considered to have a low confidence in their predicted label. By finding the label for these items, the support vector machine is able to create better support vectors and improve the accuracy of the classifier significantly with each item.

Query By Committee (QBC). QBC is another active learning strategy that tries to find hard to classify instances in the unlabelled pool. Instead of asking one classifier about its confidence in predicting a label for an item, QBC maintains a pool of trained classifiers that all try to predict labels for items in the unlabelled pool and provide their confidence in those predictions. The item chosen from the unlabelled pool for the oracle to classify is the item that the pool of classifiers disagree on the most. In our system, we used multiple support vector machines that had different learning rates so that they would come up with different predictions. Further testing and tweaking could be done to find more optimal pools of classifiers.

Active Learning by Learning. This strategy is a relatively new approach to active learning proposed by Hsu and Lin [1]. A pool of active learning strategies is used instead of only one strategy. During training, it learns which strategies are performing well on the given dataset so it can focus on using the same ones more frequently in the future. This allows the same strategy to work on multiple different datasets without selective tweaking. In our system, we use a pool of different uncertainty strategies along with a query by committee strategy. Further testing would need to be done on different text classification datasets to find a good general pool, but the pool we used performed quite well on our test datasets.

QUerying Informative and Representative Examples (QUIRE) QUIRE is also a relatively recent active learning strategy that we attempted to use in our project. Both uncertainty sampling and query by committee try to find informative items in the unlabelled pool, while QUIRE tries to find the best mix of informativeness and representativeness of an item in the unlabelled pool. Focusing only on informativeness does "not exploit the structure information of the unlabelled data, leading to serious sample bias and consequently undesirable performance for active learning" [2]. Unfortunately, in our tests, QUIRE performed rather poorly (even worse than random sampling). Whether this is an indication that the algorithm is in general poor for text classification or if there is an issue with the implementation requires further research.

3 Methodology

In this section we will cover the implementation details and the decisions that led to them.

3.1 Underlying Technologies

Python offers a plethora of third-party libraries, including machine learning libraries, one of the very few active learning libraries, and a mature web model-view-controller framework. Due to the requirements of the active learning library, libact, we selected scikit-learn as the machine learning library. Together, the libact and scikit-learn libraries implement the work covered in the previous section. For these reasons, we chose Python to implement the backend of our system.

Django. Django is a Model-View-Controller web framework implemented in Python. Our active learner is called and executed in the context of a web application. The Oracle can view the documents index, view a specific document, import data, label instances requested by the active learning process, and perform many other tasks through a web interface that is served by Django.

Our models interact with a SQLite database, which we selected for its simplicity as it does not require a separate server.

scikit-learn. This library is one of the most popular machine learning libraries available. It is generally considered low-level as it provides APIs that help developers implement solutions to preprocess data, select a model, perform dimensionality reduction, as well as classify, regress, and cluster data.

libact. This library supports pool-based active learning problems by implementing active learning strategies. libact provides a few basic classifiers by providing wrappers of scikit-learn classifiers.

We wrote our own wrappers for other scikit-learn classifiers that could support text classification.

WebSockets. WebSockets is a communications protocol standardized in 2011 that provides full-duplex communication channels over a single TCP connection. In 2013, it was implemented in all major browsers. We make use of WebSockets channels to provide a constant connection between the client and the active-learner. We use a free third-party service to provide the WebSockets connection.

3.2 System Overview

Figure 1 shows a high-level diagram of our system. It presents various components and how they interact and communicate with each other. The Oracle uses their web browser to make HTTP requests to view HTML documents and communicates with the active learner by using WebSocket channels through the WebLabeler. The HTTP server (Django) and the active learner both use Django's models to request data from the database.

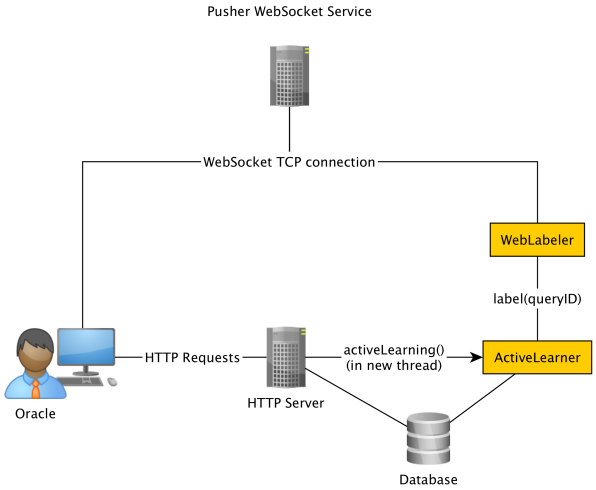


Fig. 1. System Overview

3.3 Detailed View

This subsection describes Figure 2.

The Oracle initiates the learning process by sending an HTTP get request to the server with the dataset ID that the Oracle wants to learn on. Upon receiving the request, the server will immediately send a response back to the Oracle. It will then asynchronously call the learn function which will run in a new thread. The Oracle will start listening for events on two channels: "pleaseLabel" and "showResults".

In this function, we retrieve the data relevant to the specified dataset. In our case, the data retrieved is an array of strings. We convert these strings into a TF-IDF sparse matrix with the row corresponding to a document and a column corresponding to a word in the corpus of strings.

After obtaining the TF-IDF sparse matrix, we use the holdout method to split our data into training and testing sets. The training set contains 70% of the original data while the testing set contains the remaining 30%. The training set is then modified in place to simulate a scarcity of labelled instances to active learn on.

We use a linear Support Vector Machine model as our classifier and train it using the labelled instances in the modified training set. In order to plot the quality of our classifier with respect to each instance labelled by the Oracle, we make label predictions using the initially trained model against the instances in our training set. The label predictions are stored to later compute accuracy metrics.

Given a number of instances to label, for example 20, we iterate through a loop and do the following:

1. Query the active learner for an unlabelled training instance
2. Request the Oracle to label the instance through the WebLabeler.
   (a) The WebLabeler sends the URL of the instance to label on a WebSocket channel ("pleaseLabel").
   (b) Listen for a response on a different WebSocket channel ("labelled") and setup a callback to release a semaphore currently blocking execution.
   (c) The Oracle obtains the label request and fetches the instance over an HTTP get request.
   (d) The Oracle selects a label and sends it over the "labelled" channel.
   (e) The WebLabeler returns the label.
3. The instance requested is updated in the training set with the label obtained from the Oracle.
4. The linear SVM model is retrained against the updated training set
5. The model is used to make label predictions against the testing set to see how the active leaning iteration did

After a set number of iterations, we compare the set of label predictions with the true labels from the test set to compute the final results. The final results are comprised of the confusion matrix obtained after the last active learning iteration as well as the precision, recall, and F-Measure for each class before active learning and after each active learning iteration. For each class, we display a graph showing the contribution of each instance labelled by the Oracle towards the quality of the trained classifier. Figure 3 shows the results when running the algorithm explained above for a dataset comprised of 1075 instances, starting with 4 instances labelled before the active learning process begins.

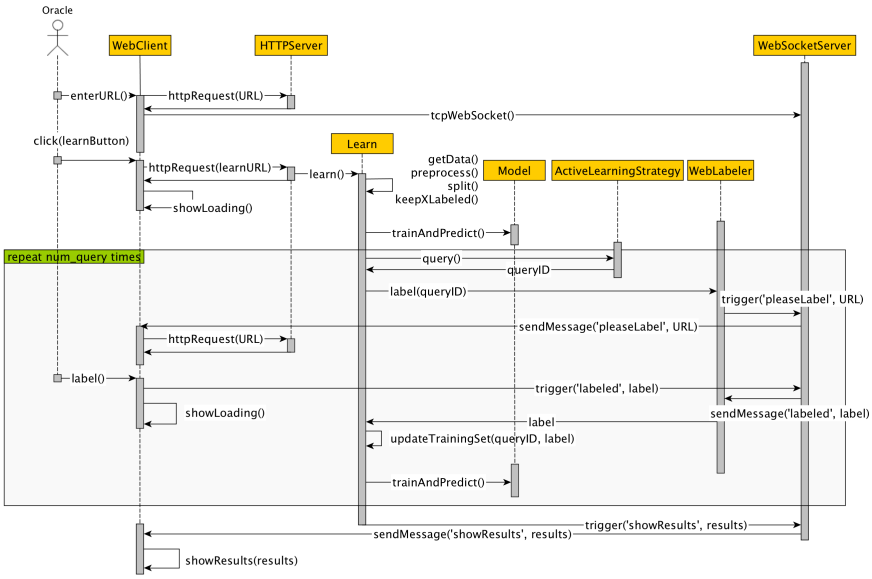


Fig. 2. System Sequence Diagram

4 Experimental Design

4.1 Experimental Data

Our system was tested using two different datasets. The first set is based on real-world data, obtained through the Bing News API. We fetched articles both related to data breaches and articles completely unrelated to data breaches. We labelled the articles based on the query term used and from this we tried to see if our system could correctly classify the articles.

The second dataset we used is a well-studied classification dataset called the 4-University Dataset. In this dataset we try to correctly classify web pages related to course websites. The dataset is noisy and contains a class imbalance, making it a good test set because it poses a challenge for our classification system.

4.2 Preprocessing

We preprocess the data in several ways to make it easier to work with. We extract the meaningful data and store the processed data in our database. Before using our data to learn, we transform it into a TF-IDF matrix. This matrix contains about 40,000 features in the 4-University dataset.

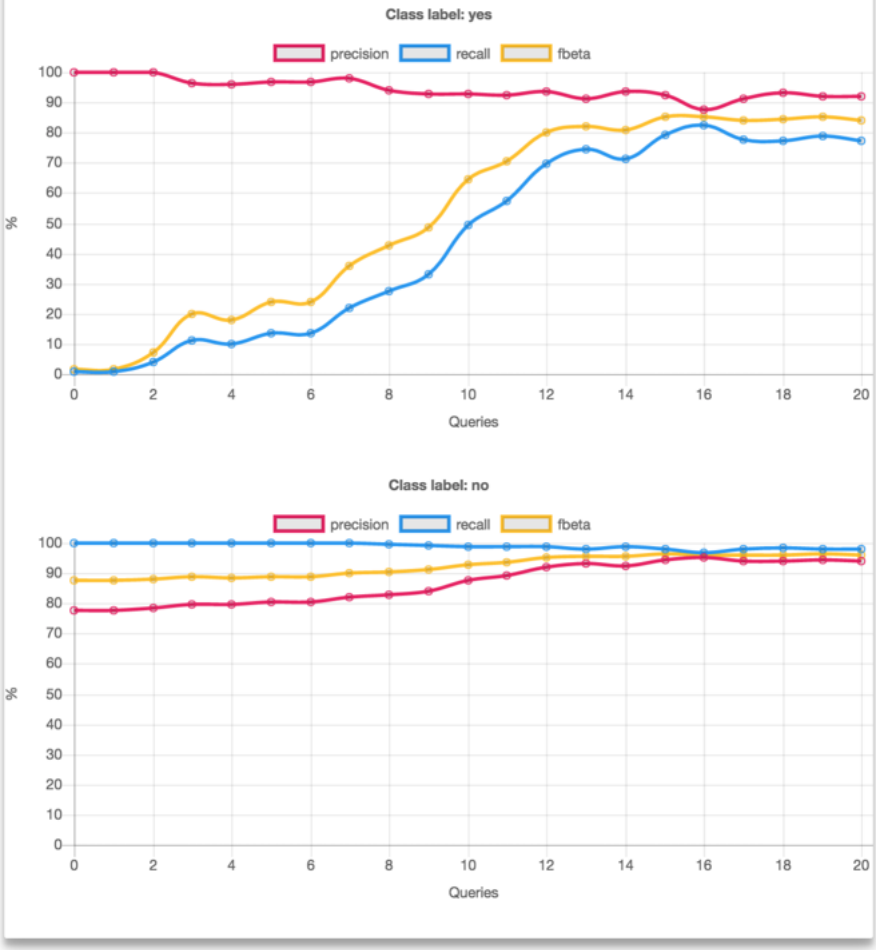
Text Extraction. Depending on the source of our data, we employ different preprocessing steps:

Active Learning Results

Strategy: Active learning by learning

Runtime: 40.53 seconds

▼ Charts per class label



▼ Confusion matrix

	Predicted: yes	Predicted: no
Actual: yes	173	51
Actual: no	15	805

Fig. 3. Active Learning results

newspaper3k is a library that allows us to fetch HTML documents and automatically extract the article text from it.

BeautifulSoup is another library we used. It allows us to easily extract data from HTML and XML files using predefined methods to navigate through the various markup tags. This facilitates extracting relevant text and titles.

Feature Extraction. In our system, we represented the data using term frequency divided by inverse document frequency (or TF-IDF).

The first task once the data has been retrieved and before classification can begin is how to represent the data to the system. Most methods involve storing a document as a vector indicating which words are present compared to all the words in all the documents. The vector can simply be boolean values representing whether the word is present, or integer values representing the number of times of times it is present. The vector stores the count of a word in a document divided by the count in all the documents. This method attributes more weight to words that are frequent in fewer documents.

While counting the term frequencies, we exclude any stop words, which are words that do not provide any important information; usually the most common in a language (the, and, such, any...)

5 Experimental Results

After running numerous tests, we attained the following results for the 4-University Dataset. We validated the results using K-fold cross validation (10 folds in this case). For testing, we started off with 5 labelled instances and performed 20 queries on the data. From the results we further determined the mean accuracy and f-score in order to perform a comparison of the different strategies. The accuracies are very high with the exception of random sampling. However, random sampling still seems to do fairly well. For this reason, we decided to also take a look at the f-score. Doing this emphasized the fact that random sampling is an unstable algorithm and although it may have a fairly high accuracy it isn't indicative of a good strategy. In the case of a largely imbalanced dataset, if everything is labelled as the majority label then the accuracy may be high but that doesn't mean the learning strategy is guaranteed to perform well.

Overall, all the learning strategies, except random sampling, seem to provide us with good results. However, uncertainty sampling seems to be the best strategy from our test results and is consistent with the lowest standard deviations.

6 Conclusion

6.1 Conclusion

In conclusion, the system developed was made to be intuitive and with high performance. In an experiment against a supervised SVM the system achieved

Strategy	Accuracy	Standard Deviation	F-Score	Standard Deviation
Meta-Learning	90.0%	3.6%	85.7%	3.1%
Query by Committee	89.6%	3.2%	83.3%	3.5%
Uncertainty Sampling	91.3%	3.0%	87.9%	2.5%
Random Sampling	80.6%	1.4%	58.9%	8.6%

Table 1. Accuracy and F-Score for active learning Strategies (5 randomly, 20 active-learned labelled instances)

an accuracy of 91% when fewer than 2.5% of the instances were labelled whereas the SVM achieved an accuracy of only 96%. Looking at the f-score of our system in comparison to the supervised SVM, we achieved an f-score of 87.9%, 2.9% higher than the supervised SVM (at 85%). Both results were validated using 10-fold cross validation. Even with the current system, there is still a lot that can be done to enhance the system.

6.2 Future Work

Some of the improvements we hope to implement in the future include parallel cross validation, for increased performance, or an ensemble method in place of our classifier, for more accurate results. Other improvements include multi-view learning on the text and title, incorporating live-updating results, and experimenting with other active learning strategies in the meta-learner. In addition, we would like to add the ability for users to import different file types into the system. This would allow us to incorporate more datasets into our testing in order to evaluate the generality of our approach. There are still many more enhancements to look at as we have just scratched the surface of machine learning. However, this system provides a starting point for those interested in the field.

References

1. Hsu, W.-N., and Lin, H.-T. Active Learning by Learning. *AAAI Conference on Artificial Intelligence*, 29 (2015), 2659–2665.
2. Huang, S.-J., Jin, R., and Zhou, Z.-H. Active Learning by Querying Informative and Representative Examples. *IEEE Transactions on Pattern Analysis and Machine Intelligence 36*, 10 (oct 2014), 1936–1949.
3. Jin, C., and Wang, L. Dimensionality Dependent PAC-Bayes Margin Bound. In *Advances in Neural Information Processing Systems 25* (2012), Neural Information Processing Systems (NIPS), pp. 1043–1051.
4. Settles, B. Active Learning Literature Survey. *Computer Sciences Technical Report 1648* (2010).