Web Services for Stream Mining: A Stream-Based Active Learning Use Case

Martin Saveski and Miha Grčar

Jožef Stefan Institute, Ljubljana, Slovenia
{Martin.Saveski, Miha.Grcar}@ijs.si

Abstract. The nature of data on the Web is becoming more and more stream-oriented and in this context, the idea of mining Web-generated data streams is becoming a hot topic. Web services, on the other hand, have become an inevitable tool for the future development of the Web. While Web services have been very successful in providing distributed computing environments, they have not been exploited for building and executing stream mining workflows. In this paper, we discuss a service-based environment suitable for stream mining and present a service-oriented stream mining workflow for sentiment classification through active learning. In the context of this use case, we present the general idea of active learning as well as an empirical evaluation of several active learning methods on a stream of opinionated Twitter posts.

Keywords: stream data mining, Web services, active learning, sentiment analysis, Twitter.

1 Introduction and Motivation

Handling vast Web-generated streams is a relatively new challenge emerging mainly from the self-publishing activities of Web users (e.g., blogging, twitting, and participating in discussion forums and social networks). Furthermore, news streams (e.g., Dow Jones, Business Wire, Bloomberg, Reuters) are growing in number and rate, which makes it impossible for the users to systematically follow the topics of their interest. The number, heterogeneity, and rate of streams, "produced" by the Web, are expected to grow substantially in the upcoming decades. In accordance with the "Web of Things"[1] vision, many devices (such as wireless sensors and even household appliances) are expected to be connected to the Web, producing data streams for the purpose of surveillance (incl. alerting and visualization) and data analysis (such as stock price prediction and monitoring, environmental and traffic monitoring, and vital signs monitoring). From this perspective, we can talk about a "Web of Streams".

In the European project FIRST (Large-scale information extraction and integration infrastructure for supporting financial decision making), the goal is to provide an infrastructure for analyzing vast streams of user-generated Web content and news feeds from the domain of financial markets. The integration infrastructure devised in

[1] Web of Things, http://en.wikipedia.org/wiki/Web_of_Things

the project will be based on SOA[2] principles. Service-oriented architectures are more and more often the technological choice for software integration and exposure of public interfaces (e.g., Yahoo!, Google, Twitter, and eBay APIs). In a standardized manner, SOA allows software interoperability across multiple, separate systems and domains. Furthermore, running on the provider's servers, it hides the proprietary implementations from the user, ensures the sufficient hardware resources, and allows the (indirect) use of proprietary data. However, SOAs suffer from several drawbacks in stream-based real-time scenarios, which have to be accounted for as further discussed in Section 2.

In this paper, we present a use case of service-oriented stream-based active learning for the purpose of sentiment analysis. The aim of sentiment classification is to decide, given a fragment of text or full text, whether the sentiment attributed to the object discussed in the text is positive or negative. The approaches to sentiment classification are in general either rule-based [1] or based on machine learning [2, 3]. We use a machine-learning approach by training a classifier on a labeled dataset of messages (i.e., tweets) posted on Twitter. One of the main problems in this setting is the lack of labeled data, especially in the domain of financial markets. Creating labeled datasets manually is an expensive and time-consuming process as it requires the active participation of domain experts. To reduce these costs, we employ active learning, a machine-learning technique designed to actively query the domain expert for new labels by putting forward data instances that, upon being labeled, contribute most to the model being built. In effect, after a certain relatively low amount of carefully chosen instances were labeled, the model performs better than if the same amount of instances were selected randomly. Our design of putting an active learning "loop" into a stream-based SOA is discussed in Sections 3 and 4.

2 Web Services for Stream Mining

Service-oriented architectures suffer from several drawbacks in specific scenarios. In their traditional form, they employ a request-response protocol, such as REST[3] or SOAP[4]. If the requests and responses are large and frequent, the connection between the client and the server most likely represents a bottleneck. In data mining workflows, inputs to elementary services (e.g., a labeled dataset required by an algorithm for building a classification model) and their outputs (e.g., the parameters of a classification model) are often very large (thousands or millions of data instances and model parameters). This situation is illustrated in Figure 1. In stream mining workflows, the inputs and outputs are usually smaller because only the newly arrived data instances and changes to the models can be passed around. However, the frequency tends to be much higher and is in fact posed by the input stream speed. In both of these two scenarios, a lot of (unnecessary) traffic happens between the client and the server and can result in a bottleneck.

[2] Service-Oriented Architecture, http://en.wikipedia.org/wiki/Service-oriented_architecture
[3] Representational State Transfer, http://en.wikipedia.org/wiki/Restful
[4] Simple Object Access Protocol, http://en.wikipedia.org/wiki/Soap

Two relatively simple techniques, pipelining and parallelization, can be employed to increase the throughput of a stream mining workflow. Pipelining refers to the parallel execution of elementary services in the "horizontal" direction (i.e., one after another). Even though the services are executed one after another, the pipeline maximizes the throughput by processing several data units at the same time. Alternatively or even in addition, parallelization in the "vertical" direction can be implemented so that two or more elementary services, either processing the same data unit or performing load balancing, are executed simultaneously. In the SOA paradigm, these techniques need to be implemented on the client side which creates a lot of (unnecessary) engineering overhead. Furthermore, as the server (or the service provider) is unaware of the workflow which is defined on the client side, it is difficult to optimize the workflow execution and workflow topology on the server side in order to increase the quality of service (QOS).

Last but not least, because streams are infinite (i.e., continuously flowing into the system) and the client is acting as a broker, the client is required to have constant access to the services. This is highly unrealistic, especially if the client is not a server machine (which is usually the case) and is eventually shut down or loses internet connection.

The solution to the discussed problems is to employ a publish-subscribe mechanism (such as ZeroMQ[5] and Java Messaging Service [14, 15, 17]) that allows elementary services to communicate with each other directly without the client acting as a broker. In this setting, the client first builds (instantiates) the workflow, which can be distributed across several servers, by "subscribing" inputs of services to outputs of services preceding them in the workflow. The client then uploads the data (and other required parameters) to the first service in the workflow. When the request is successfully processed by the first service, the results are passed directly to the next service (or services) in the workflow without the client's intervention. The client can at any time query the services about the status of its request and eventually download the results stored on the server by the last service in the workflow. The "heavy" client-server network traffic is thus limited to the upload of the data and parameters at the beginning and the download of the results when the workflow finishes processing the request.

When dealing with streams, there are two alternatives to how a data stream enters a workflow. It can either be provided by the client (constantly sending updates to the workflow) or it can be obtained from the Web by one of the services. Our use case luckily falls into the second category: the data is obtained from the Twitter API[6]. This means that after the client instantiates and configures the workflow, it can disconnect from the server without shutting down the workflow. The data-mining models built by such stream-based workflow are constantly being updated (to be up-to-date with the stream) and can at any time be queried by the client. This is illustrated in Figure 2. Apart from this clear advantage of the publish-subscribe mechanism, pipelining and parallelization come naturally with the way the inter-service communication works. Furthermore, a server is aware of the workflow fragments built by the users with the services hosted by the server. It is also aware of the other servers to which the outputs

[5] ZeroMQ, http://www.zeromq.org/
[6] Twitter API, http://dev.twitter.com/doc

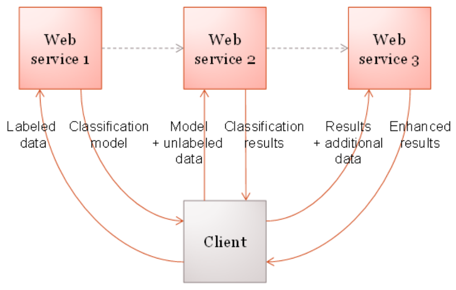

Fig. 1. The main shortcoming of Web services in data mining workflows: a lot of data is passed between the client and the server(s).

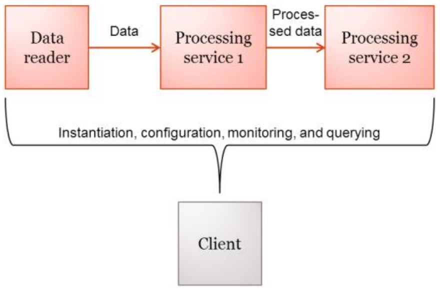

Fig. 2. The publish-subscribe paradigm in the stream-based setting: severely reduced traffic between the client and the server(s). Clearly advantageous in stream-based service-oriented environments.

are sent and from which the inputs are received in the case of a distributed environment. This allows the service owners to analyze the workflows and optimize either the elementary services that represent bottlenecks or the entire topology if, for example, the bottleneck results from the communication between the servers.

In the following sections, we present a use case for such a service-oriented publish-subscribe environment. We first present the individual services and then the workflow for stream-based active learning for building sentiment classification models.

3 Stream-Based Active Learning Workflow

In this section, we present a particular use case based on the service-oriented principles discussed in the previous sections. Specifically, we present a service-oriented workflow which dynamically builds and applies a model for sentiment classification of financial Twitter posts. To deal with the lack of manually labeled data and the dynamic nature of the data stream, we include active learning as one of the main components in the workflow. The concepts of active learning and sentiment analysis are further discussed in Section 3. In addition, we present the workflow components: Twitter API, language detector, near-duplicate detector, and active learner in more detail. Each component is implemented as a separate Web service and directly communicates with the subsequent service without the client acting as a broker. Thus, for instance, the output of the language detector component is provided as an input to the near-duplicate detector. In addition, each component provides an interface through which the client can receive a status report or configure the service.

Twitter API. Since we are interested in analyzing financial Twitter posts (tweets), the main data resource in our workflow is the Twitter API. By the informal Twitter conventions, all tweets discussing stocks contain “$” as a prefix to the stock symbol which is discussed. For instance, the “$GOOG” tag indicates that the tweet discusses the Google stocks or the “$AAPL” tag refers to the Apple stocks. This convention makes it easy to retrieve financial tweets. Twitter provides three types of APIs: REST, Streaming, and Search API. To collect as much tweets as possible, we combine the Streaming and Search API. Through the Search API, we constantly poll for a predefined set of stock symbols and through the Streaming API, we consume the Spritzer tweet stream, approximately 1% of all public tweets, and we filter out the non-financial tweets.

Language Detection. We have observed that many of the collected tweets are not in English. To ensure a better performance of the text processing components in our workflow, we have constrained the workflow to process only English tweets. Although the Twitter API provides the information about the language of a particular Twitter user, this information is often incorrect (e.g., non-English speakers often "tweet" in English). Therefore, we have developed a custom n-grams-based language detection model [4]. N-grams are *n* characters long sequences created by slicing up the text tokens. Using several text corpora in the languages we want to be able to detect, we developed a *profile* – histogram of n-gram frequencies – for each language. Thus, to detect the language of a tweet, we count the n-gram occurrences and we find the profile which makes the best match. Tweets that do not match the English language profile are discarded.

Near-duplicate Detector. We also noticed that tweets with very similar content occur many times in the stream. We observe that this is mainly caused by re-tweets and spam. Twitter provides a feature with which users can re-tweet the posts of other users, i.e. tweet the same post, but with "RT" tag and a link of the original user. Spammers, on the other hand, flood the stream with tweets posted from different accounts, but with very similar content. These tweets represent noise and may negatively influence the performance of the subsequent components in the workflow. Since using simple hashing of the tweets will not allow us to detect such tweets, we have employed the near-duplicate detection algorithm proposed in [18]. Specifically, we represent each tweet as a set of 5-shingles, i.e. set of all 5-character sequences contained in the tweet, and compute the Jaccard similarity of the shingle sets. If this similarity is above a given threshold, we consider the tweets as near-duplicates. By default, this method would require that each new tweet in the stream is compared to all the exiting tweets, which is unrealistic for the fast stream we have in this use case. To minimize the number of comparisons, we constantly keep an inverted index, which as keys has bi-grams (two word sequences) and as values has a set of tweets where the bi-gram is contained. Thus, when a new tweet arrives, we use the inverted index to retrieve a set of candidate near-duplicates and we only compare those to the tweet currently being processed.

Active Learning. This component implements the active learning principle and its output is a model for sentiment classification of financial tweets. The component keeps three pools of tweets: labeled, unlabeled, and query tweets. All tweets labeled so far are placed in the pool of labeled tweets. The pool of unlabeled tweets contains the most recent unlabeled tweets which are the candidates for the query pool. As the stream flows into the system, this pool is updated: new tweets come in, old flow out. The query pool, on the other hand, contains all unlabeled tweets which, according to the current model, if labeled will improve the model the most. Every new tweet in the

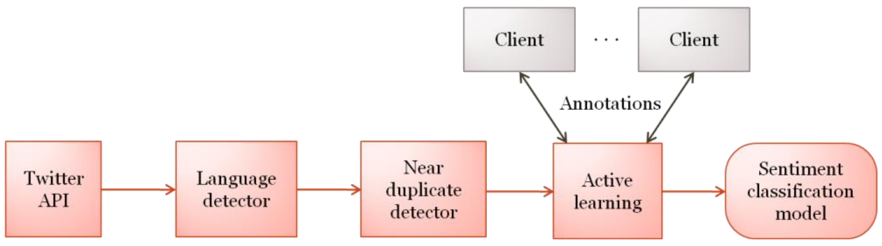

Fig. 3. Our workflow for stream-based active learning.

stream, based on the current model, is either placed into the query or unlabeled pool. With every new labeled tweet, the model is updated and accordingly, the query pool is changed. This component also exposes a Web interface through which the domain experts can label tweets (depicted as Client in Figure 3). The tweets to be annotated are taken from the query pool. The domain expert is shown one tweet at a time and can either provide a label or ask for another tweet if he is unsure about the label. This feedback is afterwards propagated to the model. As shown in the figure, many domain experts can provide annotations simultaneously. More details about the idea of active learning for sentiment analysis and the methods which we have considered for this purpose are provided in the next sections. It is important to note that due to the dynamic nature of the data, this component must include not only the application of a previously trained model, but also the model building phase. The financial sentiment indicators are constantly changing their polarity and the model must be at all times updated and in line with the current "events" evident from the stream in order to correctly capture these changes. For instance, if the word "Greece" is seen in a tweet, due to the current financial crises in the country, it may be considered as a negative financial indicator. However, before the crises, the same term was most likely a signal of neutral or positive sentiment (e.g., history and culture, holidays).

4 Active Learning for Sentiment Analysis

4.1 Sentiment Analysis

Textual information can generally be of two main types: facts and opinions. While facts consist of objective expressions, opinions are typically subjective expressions that represent people's sentiment, views, or impressions towards different events, entities, and their properties. Although factual information is very important, it is in the human nature to be interested in other people's opinions. Opinions are so important to us that every time we need to make a decision, we seek the opinions of others on the matter.

With the emergence of the Web 2.0[7], the amount of user-generated content on the Web has rapidly increased, resulting in large amounts of opinionated text. The social networks such as Twitter, Facebook, MySpace, and Digg have allowed the "word of

[7] Web 2.0, http://oreilly.com/web2/archive/what-is-web-20.html

mouth" to spread with enormous speed. This phenomenon has triggered a whole new track of research called sentiment analysis or opinion mining (in the remainder of the text we use the first term). The main problem that sentiment analysis tries to solve is to extract sentiment from text and to detect its polarity (positive or negative). Other, more complex definitions may also involve detecting the object and the object feature towards which the sentiment is expressed as well as the opinion holder.

In the literature, we generally observe two approaches to solving the problem of sentiment analysis: (1) the natural language processing (NLP) and (2) the machine learning (ML) approach. The NLP approach is mostly unsupervised and uses advanced linguistic analysis (dependency parsing), rules, and knowledge resources (WordNet) to analyze opinionated text [5, 6]. The ML approach, on the other hand, takes a supervised learning approach and defines the problem as a classification task where documents/sentences are classified into predefined categories (positive, negative, and possibly neutral), based on their sentiment. However, in some studies, these approaches are combined, for example, part-of-speech tagging can be used to construct the features used for classification [7]. With the large availability of labeled data from Web sites like Trip Advisor, IMDB, Epinions, etc., the number of studies which successfully apply the ML techniques has grown significantly [8].

In this study, we perform sentiment analysis of Twitter posts (tweets). With 145 million users (reported in September 2010[8]) and growing, Twitter is no doubtingly one of the most popular sites on the Web. Moreover, it has promoted itself as a platform where people express their personal opinions and sentiment towards companies, products, events, etc. While tweets come with a lot of metadata (user information, topic tags, references to other users), the nature of the data poses some specific challenges. Tweets can be up to 140 letters long (1–2 sentences) and usually contain informal language and expressions (slang). Consequently, this makes most of the standard NLP techniques less applicable or not applicable at all. Therefore, we apply the ML approach to sentiment analysis and we define the problem as a document classification problem. However, the limited amount of manually annotated tweets makes the classical supervised learning techniques less applicable and demands the use of other more sophisticated techniques such as active learning.

4.2 Active Learning

Many of the supervised learning systems, especially in text mining, need hundreds or even thousands of labeled examples to achieve good performance. Sometimes these labels come at little or no cost at all, for example when we flag emails as spam or when we rate movies on our favorite social networking Web site. But many times, acquiring labeled instances for more complex learning tasks can be time-consuming, difficult, and expensive. On the other hand, acquiring large amounts of unlabeled instances can be easy and without any costs. The key hypothesis of active learning is that: if the learning algorithm is allowed to choose the data from which it learns – to be "curious" – it will perform better with less training [9].

[8] Twitter Statistics, http://blog.twitter.com/2010/09/evolving-ecosystem.html

As mentioned in the previous section, in our particular use case, we are not able to build an accurate sentiment classification model because we lack the availability of tweets manually annotated with their sentiment polarity. On the other hand, we can easily consume the Twitter stream of public tweets and acquire plenty of unlabeled instances at no cost. Therefore, we use the unlabeled data and active learning to build an accurate model while minimizing the number of labeled examples and thus the associated annotation costs. We have looked at several active learning methods: Active Learning with Support Vector Machines, Hierarchical Sampling for Active Learning, and K-Means Clustering and SVMs for Active Learning.

In [10], the authors propose an algorithm for Active Learning with Support Vector Machines (SVM). In each iteration of the algorithm, an SVM is trained on the data labeled so far and the instance whose feature vector is closest to the hyperplane is queried next. The main idea behind the algorithm is that choosing the instance closest to the hyperplane will maximally reduce the version space, the space of all possible hyperplanes, thus making a more efficient search through this space to find the best hyperplane. They empirically show that the algorithm significantly reduces the number of instances needed to train an efficient SVM model.

However, in [11] the authors observe that in each iteration of a closest-to-the-boundary (boundary being a hyperplane in the previously discussed method) algorithms, the selected instances increasingly diverge from the underlying data distribution, resulting in *sampling bias*. Thus, the sampled subset of instances is hardly representative. They propose a method which starts by building a hierarchical clustering tree from the data and, given the labeled instances so far, tries to find the best pruning of the tree.

The Active Learning with SVMs (AL-SVM) algorithm starts by querying random instances to set the initial hyperplane. In an attempt to combine the two approaches, the efficient search through the hypothesis space and the exploitation of the clustering structure of the data, we have augmented the AL-SVM algorithm to include k-means clustering as the initial step. To make a better sampling of the space, instead of selecting random data instances, we first cluster the data into *k* clusters, where *k* is the number of samples we want to take, and take the medoids of the clusters as the first set of instances to be labeled.

4.3 Experiments

To measure the performances of each of the active learning algorithms discussed in the previous section, we developed a data set of noisy labeled tweets. We consumed the Twitter Spritzer stream (~1% of all public tweets), from 23rd of February to 5th of April, and collected 50 million tweets. We used the positive (:), :-), :D, ;), etc.) and the negative (:(, :-(, :’(, etc.) emoticons to assign labels to the tweets. Tweets which contained no emoticons or both positive and negative emoticons were ignored. The idea of using emoticons for tweet sentiment annotation has already been used in several other studies [12, 13]. To further process the tweets, we have used the language detection and near-duplicate removal workflow components (Section 3), filtering out the non-English and the near-duplicate tweets. This resulted in a set of 703,584 positive and 189,695 negative tweets or 881,069 tweets in total. As it can be

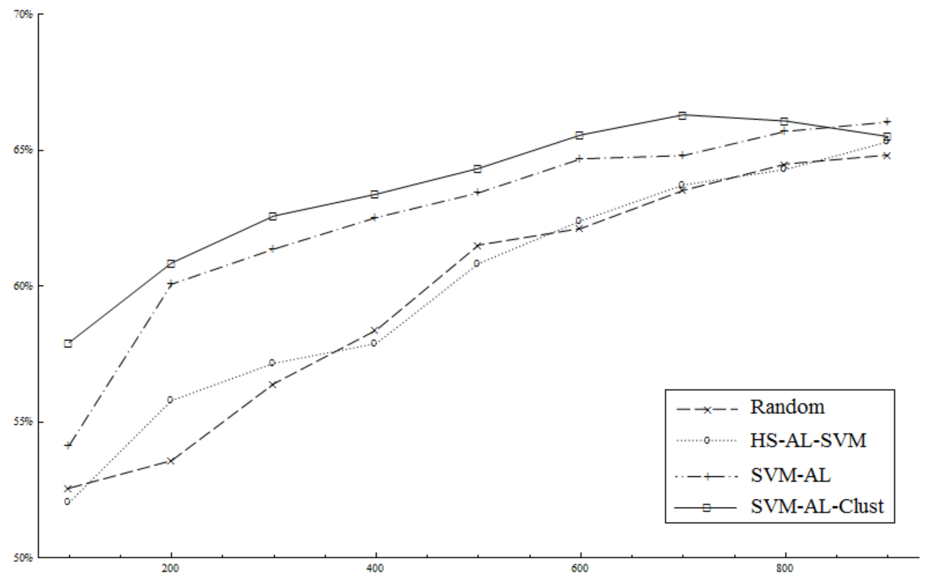

Fig. 4. Comparison between the performances of the different active learning algorithms (*x*-axis: number of labeled instances, *y*-axis: accuracy obtained with a10-fold cross-validation).

seen, the data set is highly unbalanced and the positive tweets dominate. Although this may depict the real distribution of the sentiment polarity in the stream, in these experiments we are more interested in measuring the ability of each algorithm to predict the sentiment polarity according to the features and not the prior probabilities. Thus, we balanced the data set so that it contains the same number of positive and negative tweets. All negative tweets were retained, while the positive ones were evenly sampled, resulting in a data set of 379,390 tweets. To carry out the experiments, we used the temporal meta-information provided with the tweets to simulate a data stream. In this way, we ensured that the experimental setting is the same as the one encountered in a real-life application.

Figure 4 depicts the performance of each of the active learning methods. The *x*-axis shows the number of instances sampled, while the *y*-axis shows the classification accuracy of the algorithms obtained with a 10-fold cross-validation. For reference, we have also the classification accuracy of the random selection policy which represents passive learning.

In all the experiments, we employed SVM for the sentiment classification. It is important to note that the model needs to be updated every time a new instance is labeled. Using the standard SVM implementations, such as SVM^{light} or LIBSVM, would require re-training on all instances labeled so far, which is computationally too expensive to handle the fast data stream in real time. Instead, we employed the incremental SVM implementation proposed in [16], which incrementally trains the model one instance at a time. In the case of hierarchical sampling (HS-AL-SVM), we used the sampling method for choosing instances to be labeled and we used the labeled instances to train an SVM model and test its performance. For this experiment, we have used the open-source implementation made publicly available by the authors[9].

As depicted in Figure 4, the HS-AL-SVM method shows poor performance and fails to improve the random sampling. In our opinion, this is a result of the nature of the Twitter data (short texts) which makes it hard for this method to find patterns. The SVM-AL, on the other hand, shows significant improvement, ranging from 3% to 7.5%, over the random sampling. It is also interesting to note that by performing clustering instead of random sampling, as the initial step of SVM-AL, the classification performance in the first iterations is significantly improved. This also further influences the performance in several next iterations, showing an improvement over SVM-AL. Finally, we observe that as the number of instances increases, the differences in performance of all methods (incl. Random) decreases. Thus, in the case when more than 1,000 tweets are labeled, employing active learning loses its advantage over a passive learner.

5 Conclusions and Future Work

In this paper, we discussed service-oriented stream mining workflows. We pointed out several drawbacks of traditional SOAs when mining Web-generated data streams and explained how the publish-subscribe mechanism counters these shortcomings. Furthermore, we presented a use case on building a sentiment classification model from tweets in the domain of financial markets. Since the required training data (i.e., sentiment-labeled tweets about stocks and companies) is not available, we resorted to active learning (AL) to reduce the cost of labeling the data manually.

Our preliminary experiments showed that AL helps significantly when only a few tweets (e.g., 100–200) are labeled. After 200 tweets are labeled, the accuracy of the SVM-AL-Clust algorithm is 7.5% higher when compared to the random selection policy. Unfortunately, when more and more tweets are labeled, the differences between the evaluated algorithms (incl. Random) diminish. Also, the tested algorithms fail to "pick up" rapidly and after labeling 800 tweets, the accuracy of any of the algorithms (incl. Random) accounts for roughly 85% of the final accuracy achieved by labeling the entire dataset (i.e., 379,390 tweets). We believe that, to some extent, this is because tweets are very short texts and consequently, given a small number of labeled tweets, the resulting bag-of-words space has a relatively low number of dimensions (i.e., words and n-grams). This makes it difficult for SVM to model the sentiment vocabulary early in the process.

Our next step is to measure the dimensionality of the bag-of-words space in each iteration of the AL loop. In addition, we will assess the orthogonality of the test set with respect to the training set. We assume that by putting forward the tweets that reduce the orthogonality between the training and test set, the model's accuracy will increase faster. On the other hand, the resulting bag-of-words space tends to be extremely sparse. We will employ various dimensionality reduction techniques in an attempt to reduce the orthogonality between the training and test set even further. Last

[9] Hierarchical sampling implementation, http://cseweb.ucsd.edu/~djhsu/codes.html

but not least, we will employ transductive learners in an attempt to make use of unlabeled data as well.

Acknowledgments. This work has received funding from the European Community’s Seventh Framework Programme (FP7/2007-2013) in the context of the project FIRST under Grant Agreement n. 257928.

References

1. Das, S. R., Chen, M. Y.: Yahoo! for Amazon: Sentiment Extraction from Small Talk on the Web. 53 (9), pp. 1375-1388. (2007)
2. Pang, B., Lee, L., Vaithyanathan, S.: Thumbs up? Sentiment classification using machine learning techniques.Proceedings of EMNLP’2002. (2002)
3. Dave, K., Lawrence, S., Pennock, D. M.: Mining the Peanut Gallery: Opinion Extraction and Semantic Classification of Product Reviews. Proceedings of the 12th WWW. (2003)
4. Cavnar, W. B., Trenkle, J. M.: N-Gram-Based Text Categorization. Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval. (1994)
5. Wilson, T., Hoffmann, P., Somasundaran, S., Kessler, J., Wiebe, J., Choi, Y., Cardie, C., et al.: OpinionFinder: A System for Subjectivity Analysis. Learning, 2-4. ACL. (2005)
6. Esuli, A., Sebastiani, F.: SentiWordNet: A publicly available lexical resource for opinion mining. Proceedings of LREC. (2006)
7. Barbosa, L., Feng, J.: Robust sentiment detection on Twitter from biased and noisy data. In Proceedings of COLING. (2010)
8. Pang, B., Lee, L.: Opinion Mining and Sentiment Analysis. 2 (1-2), pp. 1-135. (2008)
9. Settles, B.: Active Learning Literature Survey. University of Wisconsin-Madison, Computer Sciences Technical Report. (2008)
10. Tong, S., Koller, D.: Support Vector Machine Active Learning with Applications to Text Classification. ICML, (2000)
11. Dasgupta, S., Hsu, D.: Hierarchical sampling for active learning. Proceedings of the 25th ICML, p.208-215, Helsinki, Finland. (2008)
12. Bifet, A., Frank, E.: Sentiment Knowledge Discovery in Twitter Streaming Data. Discovery Science 6332, 1-15. (2010).
13. Go, A., Bhayani, R., Huang, L.: Twitter Sentiment Classification using Distant Supervision. Processing 1-6 (2009).
14. Eugster, P.T., Felber, P.A., Guerraoui, R.,Kermarrec, A. M.: The many faces of publish/subscribe. ACM Computing Surveys 35, 114-131 (2003).
15. Baldoni, R., Virgillito, A.: Distributed event routing in publish/subscribe communication systems: a survey. DIS Universita di Roma "La Sapienza". Technical Report (2005).
16. Cauwenberghs, G., Poggio, T.: Incremental and Decremental Support Vector Machine Learning. Neural Information Processing Systems (NIPS) (2001).
17. Eggen, R., Sunku, S.: Efficiency of SOAP Versus JMS.International Conference on Internet Computing, pages 99–105 (2003).
18. Broder, A., Glassman, S., Manasse, M., Zweig, G.: Syntactic Clustering of the Web. International World Wide Web Conference (Apr. 1997), 393-404. (1997).