Analysis of Correlation between Flight Data and Twitter

Erfan Zamanian, Amin Amiri, Georg Polzer
Department of Computer Science, ETH Zürich
[erfanz | amamin | polzerg]@student.ethz.ch

Technical Report
July 8, 2011

1. Introduction

In times of growing activity of the broad public in social networks such as Facebook or Twitter, more and more aspects of their daily life is available to the public in real time. Beside a fundamental impact on the social graph, this also allows researchers tapping a new source of real time data on events around the globe.

The fundamental goal of our project was to assess the possibility to predict flight delays from Twitter. Assuming that a person would for example report severe weather conditions already hours before a flight's departure should provide us with data to predict flight delays. The initial assumption of flight delays being strongly dependent to weather conditions proved to be wrong after analysis of the delays data.

Without any intuitive set of words possibly correlating with sources of delay such as mechanical failure of an aircraft hours before the incident, we chose to perform a Naïve Bayes classification over the whole set of words in order to find possible ways of predicting flight delays from Tweets.

2. The data

Our project was combining flight delay data with Tweets. The flight delay data has been provided by Amadeus, a travel transaction operator. It consists of flight delay data for all major American airports and airlines, both for arrival and departure delay. For our analysis we chose to focus on the departure delay, a comparison of the results to an analysis based on arrival delay could be subject of further investigations. The delay data covers the time between 1st of November 2010 and 31st of January 2011.

For every flight the data breaks down the different sources for delay, these are Delay Carrier (delay caused by the carrier), Delay Weather (delay caused by weather), Delay NAS (delay caused by the National Airspace System), Delay Security (delay caused by airport security) and Delay Late Arrival (delay caused by late arrival of the plane of the previous leg). The distribution of delay among the different sources of the delay is shown in Table 1.

Table 1: *Delay statistics*

Source	Contribution
Carrier	31%
Weather	5%
NAS	24%
Security	0%
Late Arrival	40%

The Twitter data is a random subset of all Tweets sent between 19th of November 2010 and 27th of February 2011 consisting of

roughly 100 million tweets. The tweets are not filtered for location or language and also do not contain geo-tags.

As the considered delays are exclusively from airports in the US, a safe assumption is to filter the tweets for English language, which results in around 15 million tweets. By this we significantly reduce the number of words to be considered in the training and classification which is a prerequisite for reasonable runtimes when we hold a hash table over all delay data in memory. To improve the quality of the classification we also normalized the words by stemming and stop-word removal.

3. Training and Classification

We chose as a method for classifying tweets on their potential of predicting flight delays the Naïve Bayes classifier as a standard method for classifying text documents.

When implementing the classification algorithm there are several parameters to choose. The first is the number of classes we want to assign to the Tweets. We first chose the obvious classification as true/false, i.e. being related to delay or not. In the course of our development we added the classification for four classes, each describing different levels of relation to delay.

The second choice which had to be made concerns the way we aggregate Tweets respectively the bag of words appearing in them. In the first step we aggregated all words appearing in one-hour time-frame. In a second step we aggregated bag of words in three-hour timeframes, which then get compared to the delay in a relevant time-frame. In total have thus four different methods which we compare regarding their ability to predict flight delay from tweets.

3.1. Training

The initial challenge towards the implementation of a Naïve Bayes classifier was deriving the labeling criterion to decide whether a Tweet is related to delay or not. Here a very fundamental assumption has to be made in the regard of characteristics of a significant delay as well as the "prediction power" of a Tweet (i.e. how long before a delay is a Tweet able to predict its occurrence).

We settled for a pessimistic assumption of a Tweet being able to predict a delay 0 to 3 hours after being sent. Labeling the timeframe of the three hours after a Tweet is accomplished with the simple model of comparing the accumulated total (i.e. from all sources) delay within this window to its average value. If we observe above average delay we label the corresponding tweet as positive, otherwise as negative (see Figure 1).

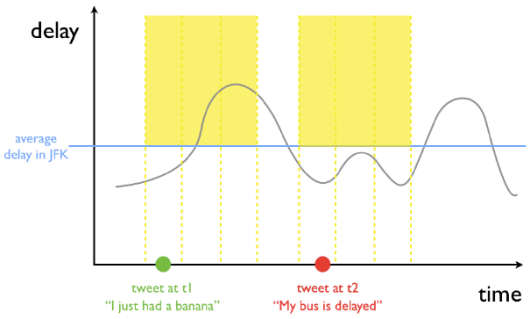

Figure 1: *Delay Window Aggregates*

In the case of multi-class labeling we differentiate here between above 1.5-times the average, above the average, above 0.5 of the average and above 0. As result we receive the table of probability of appearing in a certain class for every word. This table is then used for classifying our input documents. In our implementation we used 70% of the tweets as input for the trainer and 30% of the tweets as input for testing our classifier.

To calculate the probability P of occurrence per class for every term t we use:

$$P(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V} T_{ct'} + B} \tag{1}$$

where T_{ct} is the number of occurrences of term t in class c and B the number of terms in the vocabulary V. We add 1 for every term to avoid zero factors in Equation 1 for terms which do not appear in one of the classes. In

order to avoid underflow when calculating the probabilities for very rare word we use log scale in all probability calculations.

3.2. Naïve Bayes Classifier

The Naïve Bayes classifier gives us a tool to classify text documents based on previous training by calculating the probability for a document belonging to one of the classes. There are two derivatives of the Naïve Bayes classifier, namely the multinomial model and the Bernoulli model. Initially we chose to work with the multinomial model but after first disappointing results switched to the Bernoulli model as this model is considered to be more appropriate for short text documents.

We compute the probability P for a document d being in a particular class c with:

$$P(c|d) = P(c) \cdot \prod_{t \in d} P(t|c) \cdot \prod_{t' \in V, t' \notin d} 1 - P(t'|c) \tag{2}$$

and

$$P(c) = \frac{N_c}{N} \tag{3}$$

where $P(c)$ is called the prior, N total number of terms over all tweets and N_c the number of terms in class c. Again in the implementation we use log-scale for all calculations.

4. MapReduce Architecture

For the whole project we used in total 40 Java classes. One part is consisting of pure auxiliary code which for example runs statistics on the Twitter data or builds a per-time-window aggregated delay table. A second part deals with cleaning up the Twitter data by using stemming and stopword removal. The third and most important part consists of the trainer and classifier.

4.1. Training

The first MapReduce job of the trainer runs (*TrainerTermCountBooleanClass.java*, *TrainerTermCountMultiClass.java)* over a 70% training-subset of the total Twitter dump. For both Boolean class and Multi class mode we run in the mapper over the whole subset, extract all words, aggregate them in one hour time-windows and test for a corresponding above-average delaywindow-aggregate.

In the reduce phase we receive every term per class with the list of tweet, where this term occurred in, which we only need to add up for a total count of tweets per class. Note that we are looking at number of Tweets and not on the number of terms, as we are using the Bernoulli model (as described in Section 3.2).

We added an additional functionality to the trainer, namely also counting the total number of tweets we are dealing with. This number is used in the second phase to calculate the probabilities of terms being in a particular class.

The second phase of the tester then calculates based on the output of the first phase the probabilities for every term to be in a particular class $P(t|c)$ as described in Equation 1 (*TrainerTermProbabilityBooleanClass.java*, *TrainerTermProbabilityMultiClass.java*).

As can be seen in Equation 2 for the classifier we also need $\prod_{t' \in V, t' \notin d} 1 - P(t'|c)$. As discussed in section 4.2 below it would come with big overhead for the classification calculating the product over all number of terms in the vocabulary which do not appear in the considered Tweet for every single Tweet again. Instead we use the second phase of the trainer to precalculate $\prod_{t \in V} 1 - P(t|c)$. This is then in the classifier simply divided (as we are working in log-scale subtracted) by $\prod_{t \in d} P(t|c)$, which is computationally much more efficient (and therefore faster) than computing the full product every time again.

The results from the second phase of the testing phase can be used to gain a first insight into the prediction power of our system. By calculating the relative log likelihood of every term per class we receive a measure of significance for a particular term and class. We calculate the relative log likelihood P' as follows:

$$P'_c(t) = \prod_{c' \in Classes, c' \neq c} \log \frac{P_c(t)}{P_{c'}(t)} \tag{4}$$

With this we receive a measure for signif-

icance of a term t for each class. The smaller $P'_c(t)$ is, the higher the probability a term occurred in the testing set in that class and thus the more important (i.e. significant) a term is for a particular class. In Tables 2 and 3 we show the smallest and biggest relative log likelihoods i.e. the most and least significant terms respectively. The complete lack of connection of any of these words to matters of flights, airports or delays gives a first indication that Twitter actually might not be the optimal data source for flight delay prediction.

Table 2: *Relative Log Likelihood for Class 1 - Most significant terms*

Term	RLL
celebperfum	0.11
thebrokefriend	0.11
replacewordsinthetitlewithpussi	0.11
sexisgreat	0.12
askaiden	0.13
thatawkwardmomentwhen	0.14
inmiddleschool	0.14
thatakwardmomentwhen	0.14
leakitgaga	0.15
sbsgayodaejun	0.15

Table 3: *Relative Log Likelihood for Class 1 - Least significant terms*

Term	RLL
howareyouathug	0.69
thingsweallh	0.69
questionsidontlik	0.70
delllaptop	0.71
iiňĄwinthelotteri	0.72
unfbert	0.72
confessionhour	0.72
hoodho	0.74
wheniwaslittl	0.74
nmlbelieb	0.76

4.2. Classification

We designed our architecture for the classification as a Map task which shares the same job as the tester running in the Reduce-phase of the job. As we are working in four different modes (see Section 3, one and three hour time windows, two and four classes), we are also using four different Java classes for the classification and testing (*TesterModIBooleanClass.java*, *TesterModIMultiClass.java*, *TesterModIIBooleanClass.java*, *TesterModIIMultiClass.java*).

In the classification we run over the 30% subset of the tweets. For each tweet we calculate with the help of 2 the probability of a document (Tweet) to belong into a particular class. Here we use the precalculation of $\prod_{t \in V} 1 - P(t|c)$ in the trainer. The Mapper emits the time frame the Tweet appeared in as key together with its calculated class as value.

4.3. Testing

The tester is implemented in the Reduce phase of the same job as the classifier. For every time frame emitted from the classifier in the Map phase it receives a list of classifications. It then compares the label of the classifier with the delay window aggregate corresponding to the time frame for above average delay in the case of Boolean class and in the case of multi class for delay above 1.5-times the average, above the average, above 0.5 of the average and above 0.

Towards a ROC curve (see also Section 5) we are building a statistic over this data, counting a match in the label as a true positive/negative respectively true class 1/2/3/4 and the other way round as negative in case of a mismatch. As result we emit for each timeframe the number of true and false classifications for each class.

5. ROC Curve

5.1. ROC for Two Classes

To finally assess the possibility to actually predict flight delays from Twitter we were build-

ing a Receiver Operating Characteristic (ROC) curve. By this we were able to assess the characteristic of our prediction algorithm using classification using different sensitivities.

In the case of two boolean classes we were comparing the probability of a document (Tweet) being in the positive class, i.e. having above average delay in the 3-hour time window following this Tweet or formally $P(d|c)$, with a τ between 0 and 1. With this we vary the requirement on how strongly a Tweet has to belong to the positive class in order to mark it as positive. In the case of multiple classes (k-many) we use τ_k for every separate class and then assign the class with $\arg\max_x abs(P(d|c_k) - \tau)$, i.e. the class-assignment-probability being the most above the threshold τ.

		Actual	
		Positive	Negative
Predicted	Positive	TP	FP
	Negative	FN	TN

Figure 2: *Delay Window Aggregates*

The result is a confusion matrix as can be seen in 2. We then calculate the *Recall* and the *Specificity* with:

$$Recall = \frac{|TruePositives|}{|TruePositives| + |FalseNegatives|} \tag{5}$$

and

$$Specificity = \frac{|TrueNegatives|}{|TrueNegatives| + |FalsePositives|} \tag{6}$$

We then plot the Recall versus $1 - Specificity$. Typically the resulting ROC curve then gets benchmarked against a coin flip which has $\frac{Recall}{Specificity} = 1$ for all values.

5.2. Multi-class ROC

For the multi-class case ROC is not a standard tool as it can only deals with two classes, namely positive and negative. Nevertheless there are ways to map a multi-class case to a two-class case. Figure 3 shows two different ways of combining different classes. In case of *Merge* we simply combine multiple classes to one. In case of *Fix* we fix as many classes as needed to receive only two remaining classes. These two approaches can of course also be combined to reduce the number of classes to two.

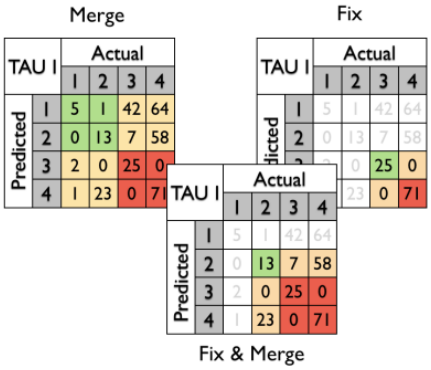

Figure 3: *Confusion Matrix for multi-class ROC*

After having reduced the number of classes considered we simply apply the calculation of Recall and Specificity as in the two-class case.

5.3. Results

The resulting ROC curves for all four combination of number of classes and modes can be seen below.

For the case of boolean classes (*true/false*) and mode 1 (terms in single tweet) we receive the curve in Figure 4. As we can see the result of the classifier resembles the performance of the benchmark (flipping a coin) closely and thus does not give an advantage over pure guessing.

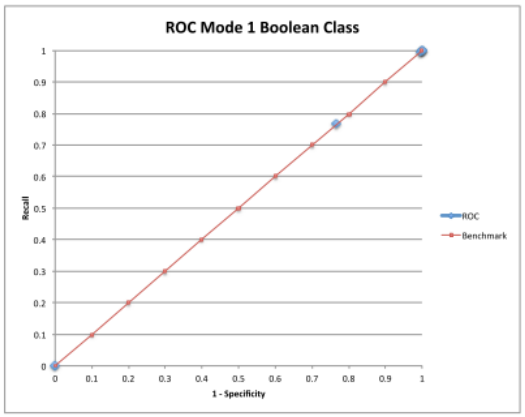

Figure 4: *ROC - Mod 1, Boolean Class*

For the boolean case in mode 1 we also considered a subset of terms which are related to flights and delays. The ROC for this case is plotted in Figure 5 and again does not offer any advantage over a flipped coin.

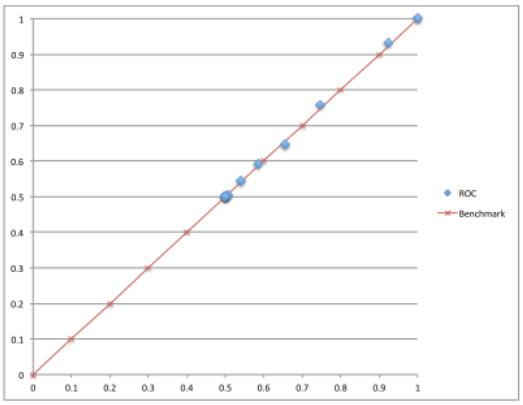

Figure 5: *ROC - Mod 1, Boolean Class, Subset of terms*

In Figure 6 we can see the ROC for 4 classes and mode 2 (terms in a bag of words). Again our classifier does not perform better than a flipped coin.

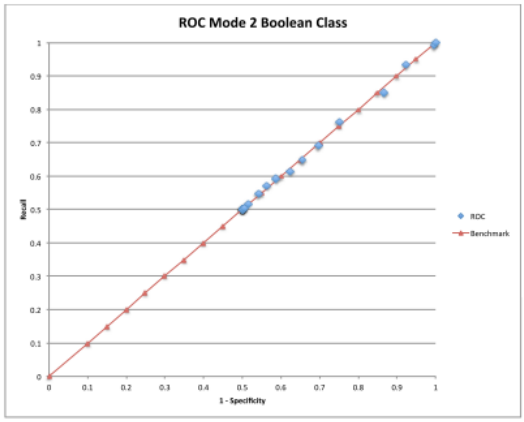

Figure 6: *ROC - Mod 2, Boolean Class*

For the multi-class case we reduce as described in Section 5.2 the number of classes considered to two. In Figure 7 we applied merging to classes 2 to 4 of the confusion matrix. The result is a very interesting ROC curve that actually suggest the possibility to simply invert the result of the classifier to receive a relatively reliable way to predict flight delays from Tweets. At the moment we do not see any justification for this result and in the light of previously shown results would encourage further benchmarking of the classifier before drawing a positive conclusion.

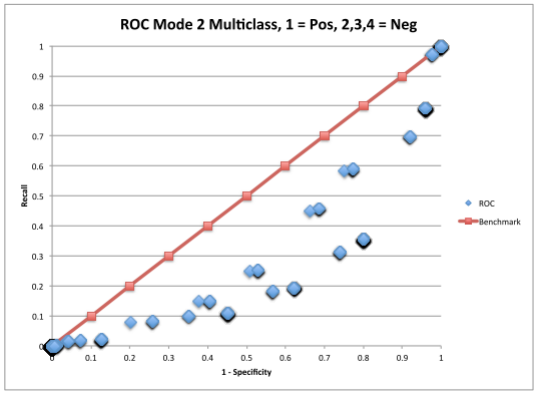

Figure 7: *ROC - Mod 2, Multi Class*

When applying fixing of classes to the confusion matrix we receive for example Figure 8 and Figure 9.

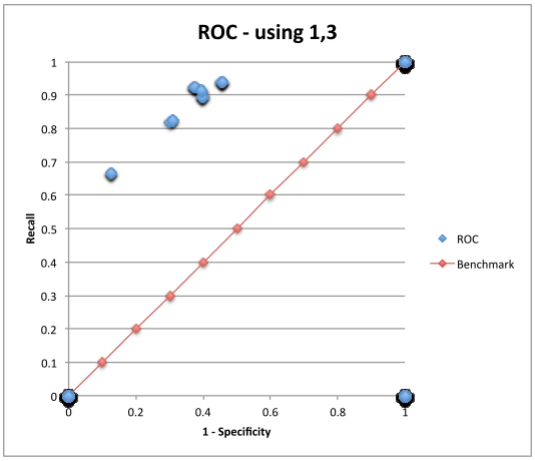

Figure 8: *ROC - Mod 2, Multi Class - 1,3*

In case of classes 1 and 3 we observe a good performance of our classifier distinguishing classes 1 and 3, while in case of 1 and 4 it does not perform better than a flipped coin. These diagrams are a strong indication of a non-sustained classification performance of our system and we therefore conclude that the our system can not reliably predict flight delays from Twitter.

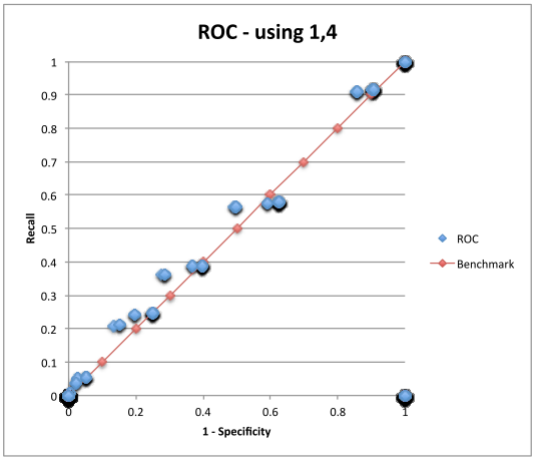

Figure 9: *ROC - Mod 2, Multi Class - 1,4*

6. Error Analysis

There are several factors which might have led to significant error in the result of the analysis. First the choice of the size for the time window aggregates has big impact on the prediction and our choice of three-hour aggregates is an arbitrary one. Here further tests with different time windows should be conducted. Also the relative delay between a Tweet sent and the occurrence of the delay might have had a severe impact on the quality of classification.

7. Conclusion

In conclusion the goal of predicting flight delays from Twitter could not be achieved. Judging from our results in Section 4.1 Twitter does not seem to contain enough significant information for reliable predictions. Since aspects of flight delays such as air traffic control or technical problems are the main reason for delay while on the same time difficult to foresee from a passenger hours in advance our result seems plausible.

Other authors (e.g. [2]) were able to predict for example the stock market prices from mere sentiment analysis of Twitter. Here the predicted event is strongly dependent to externalities and thus a connection between Tweets and stock price is intuitive. On the other hand technical problems of an aircraft are completely independent to this. Thus we conclude Twitter is no reliable data source for flight delay prediction.

7.1. Future Work

The results of this semester project could be expanded in different directions. First different airports (we only considered JFK) as well as different time-window aggregates (we used three-hour time windows) could be considered. Furthermore one could use a Support Vector Machine (SVM) instead of the Naïve Bayes classifier. Also the labeling thresholds for multi-class classification could be adjusted. In addition we focussed in our work solely on departure delay. Here a look at the Arrival delay data could bring new (and possibly different) results.

8. References

[1] Christopher D. Manning et al. *Introduction to Information Retrieval*, Cambridge University Press, 2008

[2] Bollen, Johan, Huina Mao, and Xiaojun Zeng. *Twitter mood predicts the stock market*, Journal of Computational Science 2.1 (2011): 1-8

[3] JE, Jonathan E. Fieldsend, and Richard M. Everson RM. *Visualisation of multi-class ROC surfaces*, in Proceedings of the ICML 2005 workshop on ROC Analysis in Machine Learning, 2005