Sentiment classification of Swedish Twitter data
– Technical report –

Niklas Palm

June 2019

Contents

1 Introduction

This document serves as a short summary and description of the artifacts produced during my master's thesis project at Business vision. All files referenced in this report can be found in the directory *sentiment-classifier* provided to Business vision upon terminating the project.

The project attempted multiple machine learning classifiers on Swedish manually annotated data Twitter data, varying the level of preprocessing, text-representation method and data-sampling method. The results indicate that using best practices in English research may not be suitable for the Swedish language, as, in particular, popular methods for preprocessing in English research was found to lower performance of the Swedish classifiers. The best classifiers produced were compared with a simple classifier constructed with Microsoft's cognitive API, which was found to perform surprisingly poor.

2 Data

In this section the gathered data is presented along with a description of the contents of 'sentiment-classifier/data/'.

In total over 22000 tweets were labelled in the annotation process, of which many were deemed unusable due to only containing URL links, automatically posted content or commercials. 12085 tweets were labelled as either negative, neutral or positive, the distribution of which can be seen in figure 1. In sentiment-classifier/data/ there are three csv files:
balanced-test-set.csv contains a balanced test set with 336 tweets from each class, totaling 1008 tweets. This dataset was used for all testing throughout the project.
train-set.csv contains all remaining tweets, that is 12085 - 1008 = 11077 tweets.
ALLtweets-2017.csv contains all tweets gathered from 2017, some preprocessed but the vast majority in their original form, totalling 13904702 tweets of both original tweets and responses.

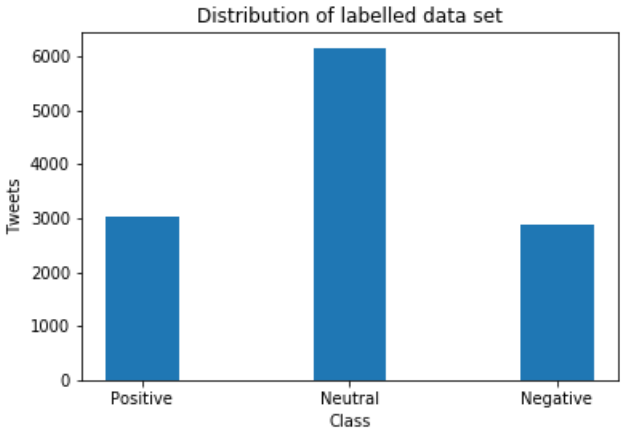


Figure 1: Labelled tweets class distribution

3 Classification models

In this section the different models used and how they were evaluated are presented.

In total, 4 different machine learning classifiers were tested: Random forest, Multinomial logistic regression (baseline model), a support vector machine (SVM) with a linear kernel and an SVM with a nonlinear RBF-kernel.

The results of the thesis indicate that both random forest and multinomial logistic regression under-performed the two SVMs significantly, why they are not further detailed in this report. The two SVMs performed similarly, the RBF-kernel surpassing the linear kernel on average, however only marginally. The main difference of the two SVMs is the inner workings of the algorithms themselves. The linear kernel creates a linear hyperplane that separates the different classes and maximizes the classification accuracy. That is, it iteratively alters the hyperplane until it **cannot** serve as a better separator. The RBF kernel applied nonlinear transformations to the data in order to create a nonlinear hyperplane. The two models are very similar and as earlier mentioned, achieves similar results. The linear kernel, however, is more than 100 times faster in both training and inference, than the RBF kernel. When classifying 1000 tweets, the linear kernel requires 0.4 seconds while the RBF kernel requires over 100 seconds, rendering the RBF kernel too inefficient for some applications.

3.1 Microsoft

A simple classifier based on the scores of Microsoft's API was constructed. Generating scores with the API, a simple algorithm found the optimal subset $S = [r, t]$ such that $pred(x) = negative$ if $x < r$, neutral if $x \in [r, t]$ and positive if $x > t$ which maximized classification accuracy with respect to the manual annotations. The optimal subset was found to be $S = [0.415, 0.681]$. Using the classifier, 51% classification accuracy was acquired with a top-precision within the positive class of 57% and 55% in the negative class. Note that this was calculated on a subset of the data of 1500 tweets and different optimal subsets may be present given a different set of tweets.

The most probable reason for the poor results is that Microsoft's cognitive API is general-domain sentiment analyzer, able of analyzing text of any domain. The literature on the subject is clear that training sentiment models on one type of data will perform significantly worse on data from different domains. A phenomenon known as the domain-transfer problem.

3.2 Metrics

When classifying text, the main evaluation metrics used are precision and recall, and sometimes classification accuracy. Classification accuracy is not recommended to use, since differences in the difficulty of the classes can vary greatly, and from an application perspective, being able to classify all tweets accurately is rarely the main goal of the application. Instead, applications usually seeks to identify all negative or all positive class samples, where class precision and class recall are two more appropriate metrics, accompanied with a confusion matrix.

3.2.1 Confusion matrix

A confusion matrix provides more in-depth characteristics of the model and what type of errors it makes. In table 1 there is a confusion matrix for a hypothetical sentiment classification problem, where there are three classes; positive, neutral and negative with 17 samples in each class. The correct classifications, marked in green, can be found along the diagonal. In this example, 12 of the 17 positive class samples were accurately classified as positive, whereas four were classified as neutral and one as negative.

		Predicted class		
	Class	Positive	Neutral	Negative
Observed class	Positive	12	4	1
	Neutral	3	9	5
	Negative	2	5	10

Table 1: A confusion matrix with three classes showing the relationship between prediction and observed classes.

A confusion matrix for a binary classification problem, that is deciding whether a sample belongs to a class or not, can be seen in table 2, where the problem is determining whether or not a sample belonged to the positive class or not. Again, the correct classifications can be found in green along the diagonal.

		Predicted class	
	Class	Positive	Non-positive
Observed class	Positive	True positive (TP)	False negative (FN)
	Non-positive	False positive (FP)	True negative (TN)

Table 2: A binary confusion matrix for the positive class.

3.2.2 Precision

Precision can be described as the ratio between the number of correctly classified samples belonging to class X and the number of predictions that a sample belongs to class X, as seen in equation 1. In the multi-class problem in table 1, precision for the positive class is the amount of correctly classified positives, over the sum of all positive-class predictions, which is $\frac{12}{12+3+2}$.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

3.2.3 Recall

Recall, or sensitivity, is the metric describing the ability to identify and classify all samples belonging to a certain class correctly. If the precision metric describes how often we are correct when classifying a sample as belonging to class X, recall describes how much of class X we manage to identify as belonging to class X, as seen in equation 2. In the multi-class problem in table 1, the recall

is $\frac{12}{12+4+1}$

$$Recall = Sensitivity = \frac{TP}{TP + FN} \tag{2}$$

3.3 Method

In this section the method of constructing and evaluating the models are described briefly.

3.4 Data sampling

Since the training data set was very imbalanced, two methods for sampling a balanced training data set was attempted, described in figure 2. The first method uses only under-sampling, meaning that the majority classes are randomly under-sampled to match the number of tweets in the minority class. The other method, using both under- and over-sampling, uses a class prevalence number and under-samples the classes containing more samples the prevalence number, and randomly over-samples the classes containing fewer samples. With the latter method, a larger data set can be constructed, though it contains duplicates of the over-sampled tweets.

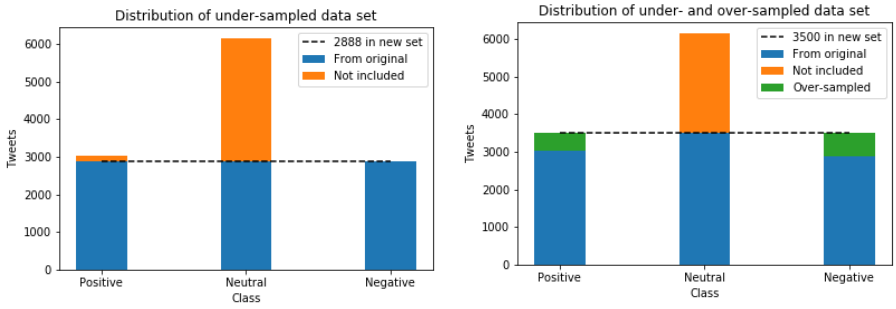
When training the models, there is a variable *only-under-sample* that is defaulted to False. With that variable set to False, the variable *new* determines the new class prevalence.[1] There is a third variable *no-sampling* which is defaulted to False, which, if set to True, orders the training to include ALL available tweets in the training set. That is, using an imbalanced training data set. This is not recommended as it introduces an algorithmic bias to the model, skewing the predictions to favor the majority class without increasing precision. It is there, however, for testing purposes.

3.4.1 Nota Bene!

It was found in the thesis that using the method of both under- and over-sampling produced the best results. In a way, since the negative and positive classes are minority classes, over-sampling those strengthens the notion of what positive and negative is. It is concluded in the literature that positive and negative sentiment have much fewer ways of expression than neutral sentiment.

[1] Note that *new* is set to a certain value in figure 2, but is easily changed in the code

That is, in essence, negative and positive sentiment have a smaller vocabulary than neutral sentiment, and including duplicates in those classes strengthens the classifiers notion of what positive and negative is.



(a) Under-sampled data set

(b) Under- and over-sampled data set

Figure 2: Class distributions of one under-sampled and one under- and over-sampled data set, illustrating how each class contributes to the generated data set.

3.5 Preprocessing

How the data was preprocessed and normalized can be found in the cleanTweet()-function of each algorithm with documentation specifying the steps taken in the code. In general, the approach is demonstrated in figure 3.

3.5.1 Tokenization

The first step of cleaning and preparing the data for the machine learning models consisted of tokenizing the text data. Tokenization is the process of separating words from each other, splitting the text on each white space creating a list where each element consists of some sequence of characters, considered as words even if some are not part of any language. This is done to be able to process each word individually.

3.5.2 Text normalization

The second step in pre-processing was normalizing the data. This was accomplished by turning all characters into lowercase characters, stripping away white

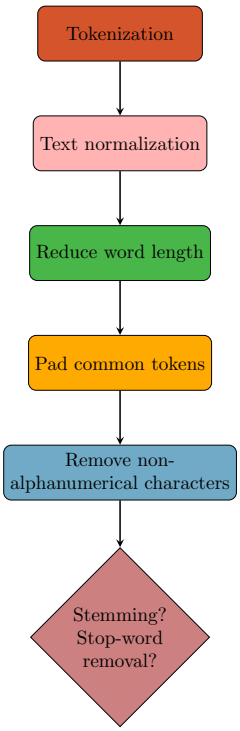


Figure 3: Flowchart showing the data cleaning process

space padding and replacing URL links, hashtags and Twitter ID handles with the meta words $< URL >$, $< HASHTAG >$ and $< ID >$. Though the replacement might affect performance, as information is removed from the tweet, it is not thoroughly studied in the literature as a bias towards certain Twitter ID handles or frequently used hashtags is considered more damaging. The meta words, however, are useful as language may vary depending on whether a person or web domain is mentioned so it is important to keep some parts of the initial information.

Additionally, it is not uncommon for tweets to contain multiple hashtags or Twitter ID handles in sequence, in which case the sequence was replaced with only one meta word.

3.5.3 Reduce word length

Due to the informality of the language used on Twitter, it is not unusual that tweets contain words where one or more characters are repeated for emphasis, such as ”sååååå bra” instead of ”så bra”. In Swedish there are no words contain-

ing characters appearing more than two times in a row, why all those sequences are reduced to only contain two characters. In the example above, ”såååå bra” is reduced to ”såå bra”, which still is not accurate. However, it limits the possible variations of the word ”så” to two, which reduces model complexity.

3.5.4 Pad common tokens

As earlier mentioned, variations on how both dashes and slashes are used was found which prompted the use of a small dictionary, containing similar characters that does not affect sentence context but word sequence. Since Twitter only allows 240 characters per tweet, many tweet authors have reduce their tweet lengths by removing white spaces around ampersands and similar characters, for instance. Hence, comparable tokens or characters are padded with white space to ensure that separate words are not mistaken for one word when creating the vocabulary.

3.5.5 Remove non-alphanumerical characters

In this step, all non-alphanumerical characters are removed in order to reduce model complexity, as is common in literature. Initial tests were run to test whether keeping numbers affected performance. It was found that removing numbers affected performance negatively, why they were left in the data.

3.5.6 Stemming and stop-word removal

Stemming is the process of reducing the number of word inflections present in the data. For instance, ”jägaren”, ”jägarens” and ”jägarna” are all inflections of the same word, ”jägare”. For stemming, the NLTK library and its built-in stemmer for the Swedish language is used. Using the stemmer on the above example, the three variations are all stemmed to ”jägar”.

Stop-words are words that have no lexical meaning but provide grammatical relationships between words within a sentence. For instance, ”ju”, ”dess” and ”sådan” are typical Swedish stop-words that satisfies the above. The NLTK library includes a list of common stop-words for a variety of languages, including Swedish, which is used to identify and remove stop-words from the data.

3.5.7 Nota Bene!

Many studies conclude that both stemming and removing stop-words can increase model performance but there are studies that achieve competitive performance with neither. The results in the thesis strongly indicated that neither stemming nor stop-word removal improved performance. In fact, the results indicated the opposite. Stemming and removing stop-words reduced model precision across the board, why the final artifact doesn't include those preprocessing steps. However, the cleantweet()-function is commented with the appropriate places for the two preprocessing-methods if they are desired in the future.

4 Text representation

While preprocessing text data is paramount for sentiment classifiers, how to represent and present the text data to the classification model is equally important. Different types of numerical representations were investigated in the thesis, including a bag-of-word approach with both uni- and bi-grams as well as word embeddings using the word2vec method.

In order to feed the data to the classifiers a vocabulary was created for the preprocessed text data. All unique grams were identified and stored in a dictionary together with an index. Each tweet is converted to a vector of the same size as the vocabulary, where each element represents the occurrence or absence of a corresponding gram in the vocabulary. Each gram in the vocabulary is therefore considered a feature, the amount of which depending on data sampling method.

4.1 Bag of Words

BoW only includes information weather a word is present or not in a sentence, for instance giving the word "love" as much importance irregardless of its position in the sentence. For the two example sentences "The dog jumps over the pond" and "The cat jumps over the fence", a vocabulary containing each unique word is constructed:

[the, dog, jumps, over, pond, cat, fence]

.
The sentences can then be represented as a vector with the count of each word in the vocabulary in its corresponding position:

[2,1,1,1,1,0,0]
[2,0,1,1,0,1,1]

4.2 N-grams

An *n-gram* is a representation that 'remembers' the $n-1$ previous words, where *bigrams* is common and have achieved good results in English research. The representation itself is not different from that of the BoW approach - it is still represented as a vector with frequencies - but the vocabulary is created by looking at $n-1$ words. A *bigram* approach of the two example sentences above would create the vocabulary

[the dog, dog jumps, jumps over, over the, the pond, the cat, cat jumps, the fence]

with the corresponding input vectors:

[1,1,1,1,1,0,0,0]
[0,0,1,1,0,1,1,1]

However, as the vocabulary size increases, the input vector becomes increasingly sparse and more computationally complex.

4.3 Word embedding

As the underlying training data and the high-dimensional input grows, the model complexity increases making very large models computationally unfeasible. By using unsupervised learning Mikolov at Google introduced a technique for learning "high-quality word vectors from huge data sets with billions of words, and with millions of words in the vocabulary".

In essence the technique assigns each gram, be it *unigram* or any *n-gram*, a vector of d random numerical values. When training it parses a huge corpus and directly employs the distributional hypothesis, which states that "words that appear in similar context tend to have similar meaning", and maximizes the cosine similarity[2] between grams that appear in similar context. It has been shown that Mikolov's 'Word2Vec', as it is called, for instance, can capture semantic similarities between words such that

$$w2v(king) - w2v(man) + w2v(woman) \approx w2v(queen)$$

[2]Cosine similarity is a measure of the cosine angle of the inner product space between two vectors

where $w2v(gram)$ is the embedding of the word 'gram' after training. Transforming high-dimensional *n-gram* input vectors into continuous vectors have multiple advantages. Besides producing a more computationally efficient word representation, the possibility to cluster similar words together can make classifying previously unseen words easier and render the model more robust.

Learning word embeddings is done in an unsupervised fashion, why labelled data is not required. All of the almost 14 million tweets are therefore used in the training with unigrams as base words, using a larger vocabulary with 50000 words and an embedding dimension of 128. As earlier mentioned, word2vec creates a random embedding for each word and then modifies it in training, minimizing the distance to words that appear in similar context. Using a larger vocabulary, the intuition is that the model will be able to correctly classify previously unseen data, given that similar words were present during training of the classifier. For instance, if the words "best" and "better" are grouped together in the embedding space but only best is present in the labelled training data, better will be treated similarly by the classifier as the two words have similar embeddings. In table 3 there is a sample of four words and their five closest neighbours (descending order) in the embedding space.

Word	Five nearest neighbours
mycket	mkt
	många
	jättemycket
	ofta
	massa
Sverige	Norge
	Tyskland
	Kina
	USA
	Europa
1	2
	3
	4
	5
	0
sämst	dåligt
	uselt
	jättedåligt
	dålig
	kass

Table 3: A sample of four words and their five closest neighbours in the embedding vector space.

4.4 Nota Bene!

When varying text representation methods I encountered memory errors. I could not test embedding vocabularies larger than 50000 and bigram-vocabularies larger than 30000 due to the 8Gb memory limit on the provided computer. The best results produced in general were those with an SVM using unigrams as text-representation method, but a major contribution to that fact may be the memory limitations. With the unigram method, ALL unique words could be included in the vocabulary, whereas only 40% of the bigrams could. In addition, using the word2vec method with 50000 words in the vocabulary achieved similar results to that of the best unigram SVM. There is a big chance that using ALL bigrams would increase performance and an even greater chance that increasing the word2vec vocabulary to, maybe, 200000, would increase model performance even more and make it more robust to unseen words. Additionally, learning the embeddings over a longer period of time might be beneficial, as the current version only trained for 24 hours. I recommend some experimenting using more computational resources!

5 Results

The best applicable results were, as earlier mentioned, achieved using the linear kernel SVM, confusion matric of which can be seen below in figure 4. With a negative and positive class precision of 70% and a negative and positive class recall of 49% and 69% respectively.

		Predicted class		
	Class	Positive	Neutral	Negative
Observed class	Positive	233	87	17
	Neutral	54	228	54
	Negative	45	125	166

Table 4: Confusion matrix using the linear kernel SVM. Correct classifications are found in green along the diagonal

Note that these results were acquired with a class prevalence of 3100, under-sampling neutral tweets and over-sampling negative and positive tweets. That is, retraining the model will most likely alter the results slightly.

5.1 Nota Bene!

One massive source of error here are the annotations. Though I labelled the vase majority, I received help from multiple people, the subjectivity of which most certainly influenced their labelling. For instance, as described below, looking at the predictions versus class probabilities along with the tweet, even I do not agree with the label at times and sometimes I even believe the prediction is more correct than the label.

6 The artifact

The directory *sentiment-classifier* includes all code and data produced in the thesis, along with a small demo, creating a simple Flask API that serves a classification-lambda which accepts a string (tweet) and returns a prediction along with class probabilities.

6.1 Notebooks

All training and evaluation was in jupyter notebooks, provided at the root of the directory. Each of the four notebooks serves a specific purpose, where three trains and evaluates models using one specific method of text representation, and one learns word2vec embeddings. Each model automatically creates a directory for the saved models and mappers, to be copied to the demo-directory if live testing is desired.

Looking at the code in one of the notebooks, a models is trained and evaluated, after which, a new calibrated classifier using Platt scaling is created. It is that model that produces class probabilities. Beneath training the calibrated Platt model, you can see the missclassified tweets in their original form, the model prediction and the class probabilities.

6.1.1 Nota Bene!

It's important to understand that the calibrated model is a model constructed besides the prediction model and that the probabilities in no way represent the inner working of the prediction model. This becomes apparent when looking at the missclassifications as the predictions sometimes differ from the max probability class. However, the class probabilities serve as an excellent indication of

how certain the prediction is, where instances with high probability for each of the classes can be considered very uncertain.

6.2 Demo

The demo directory contains the Flask demo described above. To run the demo, have a look at the README.md in the directory. To change the underlying model in the demo, copy the notebook-produced model files into the *model directory*, replacing *linear-svm.pkl*, *prob-linear-svm.pkl* and *word2index.pkl* with the corresponding files produced by one of the notebooks. Note that using the word2vec or bigram model would require some minor changes to the *helper-methods.py* file, such as the getVector()-function. The corresponding function names from within the notebooks can be copied into the helper-methods file. Currently the demo is using a linear kernel SVM.

7 Application

This technical report describes the artifact produced as a result of the project and outlines how the assets can be used. All though minor tweaking and re-training most likely is necessary, especially in order to thoroughly investigate how the bigram and word2vec representation methods perform, I'd like to offer my thoughts on how the system could be used in application.

7.1 Sample infrastructure

It all depends on how the tweets are gathered but I would recommend a distributed microservice infrastructure to ensure isolated scalability while hosting the sentiment application. An example would be an internet-facing REST API and an internal sentiment-analyzer API. The internet-facing API would accept a list of tweets and pass them to the internal sentiment-analyzer API. The sentiment-analyzer returns both sentiment prediction and class probabilities, like the demo, back to the REST API where further aggregation can be done depending on application.

7.2 Sample applications

Throughout the master's thesis, I've had two separate services in mind.

7.2.1 Intercept negative communications

The most simple service I imagined was a service directed at customers that are very keen on intercepting negative communication across their platforms. Given that we have permission from the customer, Twitter's streaming API can be used to automatically filter all tweets in real-time and tag negative ones, sending them, along with some discovery information, to another service that the customer can access. The service the customer can access could be a simple web application that displays all current unanswered tweets in relation to their brand, ordered by sentiment, where the most negative ones (or the ones deemed most pressing) are displayed in descending order. In a sense creating a service the customer can use to prioritize which tweets to respond to first, or to ensure that negative communication is intercepted as soon as possible.

7.2.2 Brand trend dashboard

This service is what I personally feel most passionate about and one I believe would create much value for the customer. For companies that value how their brand is trending on social media, I imagine a dashboard that monitors how the customers brand is "trending" online. Given that the customers brands are mentioned on Twitter, the sentiment data can be presented as a time series, illustrating how the sentiment of the brand has changed over time or in relation to some specific event. A sentiment average can easily be calculated using the predictions or probabilities procided by the demo described in section 6, but a more interesting metric could be how polar the brand mentions are. For instance, 100 positive and 100 negative tweets would produce a neutral sentiment average, but an average could be complemented by a polarity measure that over time describes whether the mentions are neutral or sentiment polar. A simple metric could be summing the absolute amount of each sentiment prediction (since -1 = negative, 0 = neutral and 1 = positive) and dividing by the total amount of samples. A score between 0 and 1 would be obtained, where a 1 would mean that the brand is mentioned in highly polar terms, whereas a 0 the opposite.

There are several different metrics that could be constructed by aggregating the predictions and probability scores and a pretty presentation in a dynamic and responsive dashboard would be a great addition to many brand-aware companies.