Contents

Abstract

In this paper I would like to investigate various methods of text analysis to improve quality of predictions based on text messages. A trade-off between prediction quality and time required is also considered.

Text analysis, which has other names, in particular, text mining and text analytics, is continiously developing. Labor-intensive manual text mining approaches first appeared in the mid-1980s, but technological advances have enabled the field to advance during the past decade. Text mining usually involves the process of structuring the input text, deriving patterns within the structured data, and finally evaluation and interpretation of the output. Typical text mining tasks include text categorization, text clustering, concept/entity extraction, production of granular taxonomies, sentiment analysis, document summarization, and a plethora of others. The term text analytics describes a set of linguistic, statistical, and machine learning techniques that model and structure the information content of textual documents for research and business intelligence,

Sentiment analysis, which refers to the use of natural language processing, text analysis and computational linguistics to identify and extract subjective information in source materials, is the most crucial problem for businesses related to social media and market advertisement. An average client may produce about 100 messages per day. What is more, sometimes sentiment data is generated by web crawlers which aggregate data from various sources. Given the amount of data those enterprises poses, it is almost infeasible to process all the messages manually. Therefore, automatic text analytics is indispensable and even methods with low precision are acceptable solution.

A basic task in sentiment analysis is classifying the polarity of a given text at the document, sentence, or feature/aspect level - whether the expressed opinion in a document, a sentence or an entity feature/aspect is positive, negative, or

neutral. Advanced, "beyond polarity" sentiment classification looks, for instance, at emotional states such as "angry," "sad," and "happy." In this work only the basic task is considered.

Existing approaches to sentiment analysis can be grouped into three main categories: knowledge-based techniques, statistical methods, and hybrid approaches which combine the aforementioned methods.

Knowledge-based techniques classify text into categories based on the presence of words such as happy, sad, afraid, and bored. Some knowledge bases not only list obvious affect words, but also assign arbitrary words a probable "affinity" to particular emotions. Those techniques can deliver better accuracy because they make use of the domain knowledge. Unfortunately such techniques heavily depend on the domain of the document and as a result the classifiers cannot be ported to other domains.

Statistical methods leverage machine learning and statistics. Those techniques have probabilistic background and focus on the relations between the words and categories. Statistical methods have 2 significant benefits over the Knowledge-based techniques: we can use them in other domains and languages with minor or no adaptations and we can use Machine Translation of the original dataset and still get quite good results. This obviously is impossible by using knowledge-based techniques.

In this paper I consider using statistical methods to perform sentiment analysis of the data refered to a specific domain. I leverage different common approaches and suggest improvements to them for solving my particular task. Moreover, I analyze some of the common problems like feature vector selection. Finally, I provide a performance comparison between my approach and common solutions in practice.

Chapter 1

Introduction

1.1 Sentiment analysis and its applications

A marketing campaign is efforts of a company or a third-party marketing company to increase awareness for a particular product or service, or to increase consumer awareness of a business or organization. It has a limited duration. A marketing campaign consists of specific activities designed to promote a product, service or business. A marketing campaign is a coordinated series of steps that can include promotion of a product through different mediums (television, radio, print, online) using a variety of different types of advertisements. The campaign doesn't have to rely solely on advertising, and can also include demonstrations, word of mouth and other interactive techniques.

After a particular campaign is finished, the business is interested in aggregating people's opinion about the products which were being promoted. The vast majority of the reviews come from social media and shoping websites where no numerical grades are presented. Basically, a review is a short text which is either positive or negative. As was mentioned before, because of large quantities of customers and review sources, manual review analysis is not feasible. The solution is to use an automated review analysis. A classificator can be built on the data that have been already analysed by humans. The larger the corpus the better.

In general, the problem of sentiment analysis depends heavily on the application's domain and can benefit from additional metadata available along with the text data. For example, in case of reviews related to political domain, the analyst might expect some amount of sarcasm and indirect implications. Moreover, a language which is used also plays important role; similar expressions in different languages have different shades of meaning. Therefore, it is often beneficial to target a specific domain and a specific language in text sentiment analysis problem. In this parer though, I consider applying only statistic methods for detection of domain specific features. There is an evidence that those techniques often produce sufficient performance [12] and remain generaly applicable.

Most of the solutions focusing on global review classification consider only the polarity of the review (positive/negative) and rely on machine learning techniques. Solutions that aim to create is a more detailed classification of reviews (e.g., three or five star ratings) with usage of linguistic features including negation and modality.

In this paper I am going to target customer reviews of baby products in English available on amazon website, the biggest retailer in the world. The data is available at [11]. This data contains 183532 labeled products reviews. The polarity grade is 1-5. 1 - means the most negative review and 5 - the most positive.

1.2 Definitions

Let us denote a set of all possible words W and a set of all possible permutations G. And let R be a set of all the reviews available. So $r \in R$ - product review. $R = \{(w, g) : w \in W, g \in G\}$. P - review polarities set. Each review has corresponding polarity $p = \{1, 2, 3, 4, 5\}, p \in P$. $|R| = |P| = m$.

X feature matrix for machine learning. X is an $m \times n$ matrix, where m is a number of training samples and n is a feature vector cardinality.

Y - label vector for machine learning. In this case it is a mapped polarity of a review. $Y = \{-1, 1\}$, where -1 correspond to negative opinion and 1 - to the positive. $|Y| = m$.

$X_{train} \subseteq X$ - training data.

$X_{test} \subseteq X$ - test data.

$Y_{train} \subseteq Y$ - training labels.

$Y_{test} \subseteq Y$ - test labels.

$extr : R \to X$ - feature extraction function. Given a review r, this functions maps it to the feature vector x.

$pol : P \to Y$ - polarity map function. Maps polarity to binary vector.

$f : X \to Y$ - classification function. Given a feature vector x, this function predicts its label y.

Now text analysis process can be defined. The sets R and P are available. Firstly, an extraction function *extr* must be defined. After feature extraction process we should be able to form a feature matrix X. Also the representation of the polarities set P is changed by applying a map function *pol*. Hence, X and Y are defined.

At this moment, I shall split the data into training and cross-validation set. Or in other words training and test data. Now f must be found by a classification algorithm on the training set.

The ultimate goal is to obtain a decision function $F : R \to \{extr, f\}$ that would maximize F1 score for training data.

The F1 score, commonly used in information retrieval, measures accuracy using the statistics precision p and recall r. Precision is the ratio of true positives (tp) to all predicted positives (tp + fp). Recall is the ratio of true positives to all actual positives (tp + fn). The F1 score is given by

$$F1 = 2\frac{precision * recall}{(precision + recall)}$$

Hence, F1 score can be interpreted as a weighted average of the precision and recall.

Given the fact that I am considering a problem of multiclass classification, I am going to use averages of precision, recall and F1 score as a measure of classification quality.

Chapter 2

Classification Techniques

2.1 k Nearest Neighbors

k Nearest Neighbors (KNN) is part of supervised learning that has been used in many applications in the field of data mining, statistical pattern recognition, image processing and many others. The idea is to predict class for a given feature vector using so called majority voting. Suppose we want to predict label for a feature vector $x_i \in X_{test}$. Then we look on labels of k training vectors which are "the closest" to x_i and use the most frequent label as a prediction. To define "closeness" some distance metric is used. There are several options for distance functions.

A commonly used distance metric for continuous and discrete variables is Euclidean distance:

$$\sum_{i=1}^{n}(a_i - b_i)^2$$

, where a,b - feature vectors. $a_i - b_i$ - vector difference.

For discrete variables, such as for text search, another metric can be used, such as the overlap metric (or Hamming distance).

The best choice of k depends upon the data; generally, larger values of k reduce the effect of noise on the classification, but make boundaries between classes less distinct. A good k can be selected by various heuristic techniques. In our case though I am going to use a fixed value.

This alorithm does not require training step. Having sets X_{train} and Y_{train} and distance function *dist* is enough for making predictions. On the other hand, the classification of test data takes time, because for every review without label, the whole training corpus must be processed to calculate distance. For one prediction expected time complexity is $O(m_{train} * n)$. Performing indexing on the training set can decrease the running closer to $O(n)$. However, indexing is a computationally heavy operation and in case data is updated often, this approach becomes infeasible. Moreover, precission of this classification algorithm is usually low compared to others. Another drawback of the basic "majority voting" classification occurs when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the k nearest neighbors due to their large number.

In practice KNN algorithm performs worse than other algorithms. Therefore KNN model is mostly used as a baseline to compare it with other models.

2.2 Naive Bayes

Naive Bayes algorithm is a classification algorithm based on Bayes' theorems, and can be used for both exploratory and predictive modeling. The word naive in the name Naive Bayes derives from the fact that the algorithm uses Bayesian techniques but does not take into account dependencies that may exist. This algorithm is less computationally intense than most other classification algorithms, and therefore is useful for quickly generating mining models to discover relationships between input columns and predictable columns [7].

Naive Bayes relies on very simple representation of the document - Bag of words.

Input:

- A document d
- A fixed set of classes $C = \{c_1, c_2, ..., c_j\}$
- A training set of m hand-labeled documents $(d_1, c_1), (d_2, c_2), ..., (d_m, c_m)$

Output:

- A learned classifier $y : d \to c$.

According to the Bayes theorem, for a document d and class c:

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

.

This probability can be used to predict a class of a document.

$$c_{best} = \underset{c \in C}{\operatorname{argmax}} P(c|d) = \underset{c \in C}{\operatorname{argmax}} \frac{P(d|c)P(c)}{P(d)} = \underset{c \in C}{\operatorname{argmax}} P(d|c)P(c)$$

Taking into account the fact that d is a bag (set) of words:

$$c_{best} = \underset{c \in C}{\operatorname{argmax}} P(d|c)P(c) = \underset{c \in C}{\operatorname{argmax}} P(w_1, w_2, ..., w_n|c)P(c)$$

Now let us find the way to compute two probailities in the equation above. Let us start with $P(c)$. This probability can be estimated by computing how often this class occurs. It can be computed by counting relative frequencies in a corpus.

Computing $P(w_1, w_2, ..., w_n|c)$ is more subtle. There is $|W|^n * |C|$ parameters to be computed, which is impossible in practice. In Naive Bayes classifier two simplifying assumptions are made:

- **Bag of Words assumption:** Assume position does not matter
- **Conditional Independence:** Assume the feature probabilities $P(x_i|c)$ are independent given a class c.

Despite these simplification, in practice a problem can be solved with a high degree of accuracy. The result of the simplyfying assumptions is the following equation:

$$P(w_1, w_2, ..., w_n|c) = P(w_1|c) \times P(w_2|c) \times ... \times P(w_n|c)$$

Therefore:

$$c_{best} = \operatorname*{argmax}_{c \in C} P(w_1, w_2, ..., w_n|c)P(c) = \operatorname*{argmax}_{c \in C} P(c) \prod_{w \in W} P(w|c)$$

Now we have to find the way to compute these probabilities. The most natural way to implement this is simply to use frequencies in the data.

$$P(w_i|c) = \frac{doccount(C = c)}{N_{doc}}$$

$$P(c) = \frac{count(w_i, c)}{\sum_{w \in W} count(w, c)}$$

So I am going to compute the fraction of times word w_i appears among all words in documents of class c. To do that it is possible to concatenate all available documents of a particular class c into one mega-document. In case of this paper there is two mega-docuemtns D_{pos} and D_{neg}, which correspond to positive and negative reviews.

Unfortunatelly, there is a problem with the approach described above. What should happen if we have not seen training example with the word **fantastic** and classified in the positive?

$$P("fantastic"|positive) = \frac{count("fantastic", positive)}{\sum_{w \in W} count(w, positive)} = 0$$

As can be seen, even though the word **fantastic** is usually associated with positive sentiment, the probability of a document which contains this word belongs to D_{pos} is zero. Here is why:

$$c_{best} = \operatorname*{argmax}_{c \in C} P(c)P("fantastic"|c) \prod_{w \in W/\{"fantastic"\}} P(w|c)$$

For positive class this product is equal to zero.

The solution to this problem is adding Laplace (add-1) smoothing for Naive Bayes. Simply one is added to each of the counts:

$$P(w_i|c) = \frac{count(w_i, c) + 1}{(\sum_{w \in W}(count(w, c) + 1)} = \frac{count(w_i, c) + 1}{\sum_{w \in W} count(w, c)) + |W|}$$

Now the algorithm can be defined:

- From training corpus, extract Vocabulary - the list of words.
- Calculate $P(c)$ terms

  For each $c_j \in C$ do

  $docs_j$ <- all documents with class c_j

$$P(w_i|c_j) = \frac{|docs_j|}{|total\#documents|}$$

- Calculate $P(w_k|c_j)$ terms

  $Text_j$ <- single document containing all $docs_j$

  For each $w_k \in W$

  n_k <- number of occurences of w_k in $Text_j$

n <- total number of words in the class j.

$$P(w_k|c_j) = \frac{n_k + \alpha}{n + \alpha \times |W|}$$

Note: In the last equation α is used instead of one to generalize the model.

It turns out Naive Bayes has a very close relationship to language modeling. Each class in Naive Bayes classifier is a unigram language model. TODO link on unigram language model.

Probaility of word: $P(word|c)$

Probability of sentence: $P(s|c) = \prod P(word|c)$

Let us see how it works. Imagine that we have a class positive: And we have table of likelihoods of each word.

Likelihood	Word
0.05	I
0.2	love
0.002	this
0.05	beautiful
0.03	dress

Then the probability of sentence "I love this fun film" will be $P(s|pos) = 0.00000003$.

A model negative can look like:

Likelihood	Word
0.05	I
0.002	love
0.002	this
0.0001	beautiful
0.03	dress

Then the probability of sentence "I love this fun film" will be $P(s|neg) = 0.0000000000006$.

As we can see $P(s|pos) > P(s|neg)$. Therefore, this sentence is classified as positive, which is correct. Hence, Naive Bayes can be viewed as a unigram language model conditioned on the class.

Summary:

- Very fast algorithm, low storage requirements
- Robust to irrelevent features: Irrelevant features cancel each other without affecting results.
- Very good at domains with many equally important features
- Optimal if independence assumption is true.

2.3 Logistic regression

Linear regression can easily be used for classification in domains with numeric attributes. Indeed, we can use any regression technique, whether linear or nonlinear, for classification. The trick is to perform a regression for each class, setting the output equal to 1 for training instances that belong to the class and 0 for those that do not. The result is a linear expression for the class. Then, given a test example of unknown class, calculate the value of each linear expression and choose the one that is largest. This scheme is sometimes called multiresponse linear regression.

Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function.

One way of looking at multiresponse linear regression is to imagine that it approximates a numeric membership function for each class. The membership func- tion is 1 for instances that belong to that class and 0 for other instances. Given a new instance, we calculate its membership for each class and select the biggest.

Multiresponse linear regression often yields good results in practice. However, it has two drawbacks. First, the membership values it produces are not proper probabilities because they can fall outside the range 0 to 1. Second, least-squares regression assumes that the errors are not only statistically independent but are also normally distributed with the same standard deviation, an assumption that is blatently violated when the method is applied to classification problems because the observations only ever take on the values 0 and 1.

A related statistical technique called logistic regression does not suffer from these problems. Instead of approximating the 0 and 1 values directly, thereby risking illegitimate probability values when the target is overshot, logistic regression builds a linear model based on a transformed target variable [5].

In logistic regression the probability of classes is estimated in the following way:

$$P(1|w_1, w_2, ...w_n) = \frac{1}{1 + e^(- a_0 - a_1w_1 - ... - a_nw_n)}$$

- where a_i is weight.

$$P(0|w_1, w_2, ...w_n) = 1 - P(1|w_1, w_2, ...w_n)$$

The goal function, which must be minimized, is:

$$\sum_{i}^{m} (1 - x_i)log(1 - P[1|w_1, w_2, ...w_n]) + x_ilog(P[(1|w_1, w_2, ...w_n])$$

In the multiclass case, the training algorithm uses the one-vs-rest. One-vs-the-rest is also known as one-vs-all. This strategy consists in fitting one classifier per class. For each classifier, the class is fitted against all the other classes. In addition to its computational efficiency (only n classes classifiers are needed), one advantage of this approach is its interpretability. Since each class is represented by one and one classifier only, it is possible to gain knowledge about the class by inspecting its corresponding classifier. This is the most commonly used strategy for multiclass classification and is a fair default choice [6].

However, it is also possible to use Multinomial logistic regression in the multiclass case. In natural language processing, multinomial LR classifiers are commonly used as an alternative to naive Bayes classifiers because they do not assume statistical independence of the random variables (commonly known as features) that serve as predictors. However, learning in such a model is slower than for a naive Bayes classifier, and thus may not be appropriate given a very large number of classes to learn. In particular, learning in a Naive Bayes classifier is a simple matter of counting up the number of co-occurrences of features and classes [14].

In comparison with Naive Bayes algorithm Logistic regression is better at Domains with many non-equally important features. Moreover, Logistic regression is as well robust to irrelevant features when regularization is properly set [3]. Besides, logistic regression is intrinsically simple, it has low variance and so is less prone to over-fitting. .

There are also drawbacks, for instance, Logistic regression is really fast algorithm, still Naive Bayes is faster. Another drawback is that logistic regression requires more data comparing to other classifiers to achieve significant prediction quality. Finally, logistic regression assumes linear dependencies between feature vector and label. A user has to specify non-linear features manually if linear dependency is not the case.

Chapter 3

Feature Extraction

3.1 Motivation

The entities, I need to classify, are text messages that are given in the form of strings. Raw strings are not convenient objects to handle by classifiers. Most machine learning algorithms can only classify numerical objects or otherwise require a distance metric or other measure of similarity between the objects. Moreover, string by itself doesn't represent a polarity of the message in any way. Therefore, set of features should be extracted. We have to convert all messages to numerical vectors called *feature vectors*, and then classify these vectors.

The simplest example of a feature vector is the vector of the numbers of occurrences of all the words in the dictionary in a message. This approach is fairly straightforward and usually provides good results. However, there are a few problems with this approach. When training set if large enough, there is possibly hundreds thousands of words. It is definitely possible to perform classification on matrix with such cardinality, yet apparently this procedure will require a lot of time. Moreover, a vast majority of the words in the training set will not contribute to success of the classifier. Using small set of words is also problematic, because important words are potentially missed.

Let us assume we solved the aforementioned problem and are able to extract proper significant words from the data. Another pitfall is that most of languages have different forms of the same word. Those languages are called fusional and English pertains to them to some extend. All the words must be transformed to their basic form somehow.

The last but no the least, there is a problem of negation. Occurance of the fantastic usually means that a review is positve. However, in the sentence "This product is not fantastic at all", using this word as a feature is not enough to classify it properly. Although this example might seem a little contrived, a point is clear, negation can cause a classifier to make mistakes.

To sum up, Feature Extraction is perhaps the most difficult task in SA. In the related literature, we did not find many works focusing on FE. In fact, the majority of the available works focus on sentiment classification, the problem which is well researched. There are plethora of machine learning algorithms which are suitable for review classification task. Choosing the best algorithm may improve a quality of prediction. In practice, however, it is much more important what features are chosen for classification than what classification algorithm is used. If the features are chosen badly, then any machine learning algorithm will fail to classify data correctly no matter how good it is. A reasonable choise of features can make classification both precise and quick.

3.2 Important Words Selection

As was mentioned in the previous section, the simplest approach to solve the problem of feature extraction is select a subset of significant words. There are other models though. One of them is using n-grams. In particular, studies show that 2-grams and 3-grams language models provide good performance [1]. Still the goal of my work is to combine reasonable prediction quality with fast training time. Taken into account that there are more than 100.000 unique words, using those and more sophisticated models will impose significant training time. Therefore, I have decided to use words as a features in this work.

Extraction of features usually means that some information from the original message is lost. On the other hand, the way feature vector is chosen is crucial for the performance of the filter. There is a trade-off between prediction quality and training speed. Usually, the difference between performance of a really detailed model and a reasonably simple model is not significant. Yet the training speed of a simple model is considerably swifter. Therefore, in most practical applications the most basic vector of word frequencies or its modification is used [5].

Now let us consider extracting a "small" dictionary. The problem is to decide which words to include not to lose important features. Of course, adjectives contribute to the opinion a lot. People tend to use them when they get emotional. I, however, consider using all parts of speech because words like "love" and "disappointment" also bear a strong polarity. Therefore, there will be no filtering based on parts of speech.

First step to take is removing punctuation. Usually, punctuation does not contribute to the meaning. However, emoticons, which can be parsed as punctuation, usually do. That is why I have decided to perform emoticons parsing before removing punctuation. People tend to use emoticons quite often and a good property of emoticons is that they are no subject to negation.

To eliminate words which does not contribute much to a polarity of a review, I am going try using so-called STOP words. It is a least of pronouns, determiners and specific words of other parts of speech which play solely functional role in a language. After that I have to consider ways of transforming words to their basic form. For instance, in English, the word "go" has other inflected forms such as "goes", "going", "went", and "gone". There are two common ways of solving this problem. They are Lemmatization and Stemming.

In linguistic morphology and information retrieval, stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form—generally a written word form. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. Algorithms for stemming have been studied in computer science since the 1960s. Many search engines treat words with the same stem as synonyms as a kind of query expansion, a process called conflation [2].

Lemmatisation (or lemmatization) in linguistics, is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. It is the algorithmic process of determining the lemma for a given word. Since the process may involve complex tasks such as understanding context and determining the part of speech of a word in a sentence (requiring, for example, knowledge of the grammar of a language) it can be a hard task to implement a lemmatiser for a new language [2].

Lemmatisation is closely related to stemming. The difference is that a stemmer operates on a single word without knowledge of the context, and therefore cannot discriminate between words which have different meanings depending on part of speech. However, stemmers are typically easier to implement and run faster, and the reduced accuracy may not matter for some applications [9].

I have decided to use lemmatization in my application because there are lots of them available for English language and because they improve a quality of text analysis. Before applying lemmatization though, I need to tag every word with part of speech. For this purpose I utilize The Brown Corpus.

The Brown Corpus is a carefully compiled selection of current American English, totaling about a million words drawn from a wide variety of sources. The Brown Corpus was the first million-word electronic corpus of English, created in 1961 at Brown University. This corpus contains text from 500 sources, and the sources have been categorized by genre, such as news, editorial, and so on. Every word is labeled with part of speech. Part-of-speech tags used in this corpus can be found on wikipedia.

After all those steps performed I suppose a reasonable thing to do is select top k used words. To do this selection I will form an inverted index, which is a dictionary of words and number of their occurrences in the data. I believe that selecting top k words makes sense, because it both allows to reduce cardinality of the training matrix and make it less sparse, so that there is a high probability that different feature representations of reviews have some features in common.

3.3 Other transformations

In the previous section I described in detail the process of words selection. However, before beginning classification there is still a work to be done. Consider the following example: "My baby likes this toy", "My baby doesn't like this toy". Using our previous approach there is no difference between those two sentences in a classification matrix. Nevertheless, obviously there is a significant difference – polarity is exactly the opposite. One solution would be to include "not" in the dictionary and hope that classifier will give it significant negative wage. The issue with this approach though is that in more complex reviews, a negation can bear positive sense. Consider an example "I really like this cradle. It looks nice and my baby does not cry as often as before". We can clearly see that there is nothing negative in this review, yet this particular review can be classified as negative. Therefore, another solution should be found.

One way to deal with this issue is to use antonyms. However, a list of antonyms should be defined. Even though it is certainly possible to find antonyms dictionary for lots of languages, this solution lacks generalization. Some domain specific words are probably not defined in such dictionaries. Besides, some words have many antonyms and it is quite ambiguous to decide which selection is the best.

I am going to use the solution proposed in [2]. Every word after negation will be prefixed with "NOT_" prefix. Because the punctuation is eliminated before text analysis, this transformation will affect only the closest words near a negation. Usually it is exactly what is required. Of course number of words is increased that way. Nevertheless, as described before, because I build vector based on top k used words, in the result there is no difference.

After words are selected there is natural choice between representing text as a list of word counts or as a Boolean vector. According to some studies in Sentiment Analysis the number of occurrences of the word in the text does not make much of a difference. Usually Binarized versions (occurrences clipped to 1) of the algorithms perform better than the ones that use multiple occurrences. Another reason to tone down words that appear often in a text is that a word that appears regularly is more likely to have a neutral sense. This is particularly true of nouns. In one example from our corpus, the words death, turmoil, and war each appear twice. A single use of any of these words might indicate a comment (e.g., I was bored to death), but repeated use suggests a descriptive narrative.

Chapter 4

Effectiveness of trained models

4.1 Configuration and comparison description

Usually algorithms classify opinion as either positive or negative. My sentiment analysis application is more precise and is able to measure polarity in range from 1 to 5.

First, I am going to compare different configurations of my algorithm on a subset of the data. There are different configurations. In particular:

Boolean configurations:

USE_STOP_WORDS

USE_EMOTICONS

USE_NEGATION

USE_BOOLEAN_REPRESENTATION

Numeric configurations:

NUMBER_OF_REVIEWS_TO_ANALYZE

NUMBER_OF_POPULAR_WORDS_TO_USE

In this section I will fixate *NUMBER_OF_REVIEWS_TO_ANALYZE* and *NUMBER_OF_POPULAR_WORDS_TO_USE* for training to complete in reasonable time. For every run the data is divided into training and test sets. In this section I will show performance for 80:20 ratio. In next sections, I will show performance for both 60:40 and 80:20 ratio.I consider confusion matrix and classification report as a performance metric. Classification report consists of precision, recall, f1-score for each class. Moreover, there are averages of this metrics, which I consider the most crucial part. I think of an algorithm A as a “better” algorithm than an algorithm B if its average f1-score is higher.

4.2 Configuration tuning comparison

In this section Naive Bayes is used. Data is devided into training and test sets with ratio 80:20.

Let us fixate:

$NUMBER\_OF\_REVIEWS\_TO\_ANALYZE = 10000$

$NUMBER\_OF\_POPULAR\_WORDS\_TO_USE = 1000$

Setup 1.1

Parameter	Value
USE_STOP_WORDS	False
$USE\_EMOTICONS$	False
$USE\_NEGATION$	False
$USE\_BOOLEAN_REPRESENTATION$	False

Confusion matrix:

$$\begin{bmatrix} 123 & 23 & 30 & 19 & 47 \\ 43 & 25 & 24 & 14 & 36 \\ 26 & 21 & 34 & 41 & 49 \\ 26 & 27 & 46 & 82 & 164 \\ 52 & 28 & 34 & 100 & 886 \end{bmatrix}$$

Classification report:

class	precision	recall	f1-score	support
1	0.46	0.51	0.48	242
2	0.20	0.18	0.19	142
3	0.20	0.20	0.20	171
4	0.32	0.24	0.27	345
5	0.75	0.81	0.78	1100
avg/total	0.55	0.57	0.56	2000

Setup 1.2

Parameter	Value
USE_STOP_WORDS	False
$USE\_EMOTICONS$	False
$USE\_NEGATION$	False
$USE\_BOOLEAN_REPRESENTATION$	True

Confusion matrix:

$$\begin{bmatrix} 132 & 19 & 21 & 18 & 52 \\ 44 & 24 & 20 & 17 & 37 \\ 28 & 9 & 38 & 32 & 64 \\ 26 & 20 & 35 & 83 & 181 \\ 43 & 17 & 24 & 82 & 934 \end{bmatrix}$$

Classification report:

class	precision	recall	f1-score	support
1	0.48	0.55	0.51	242
2	0.27	0.17	0.21	142
3	0.28	0.22	0.25	171
4	0.36	0.24	0.29	345
5	0.74	0.85	0.79	1100
avg/total	0.57	0.61	0.58	2000

Setup 1.3

Parameter	Value
USE_STOP_WORDS	False
$USE\_EMOTICONS$	False
$USE\_NEGATION$	True
$USE\_BOOLEAN_REPRESENTATION$	True

Confusion matrix:

$$\begin{bmatrix} 134 & 23 & 24 & 17 & 44 \\ 50 & 18 & 30 & 15 & 29 \\ 24 & 15 & 42 & 33 & 57 \\ 20 & 20 & 35 & 92 & 178 \\ 36 & 15 & 32 & 90 & 927 \end{bmatrix}$$

Classification report:

class	precision	recall	f1-score	support
1	0.51	0.55	0.53	242
2	0.20	0.13	0.15	142
3	0.26	0.25	0.25	171
4	0.37	0.27	0.31	345
5	0.75	0.84	0.79	1100
avg/total	0.57	0.61	0.59	2000

Setup 1.4

Parameter	Value
USE_STOP_WORDS	False
$USE\_EMOTICONS$	True
$USE\_NEGATION$	True
$USE\_BOOLEAN_REPRESENTATION$	True

Confusion matrix:

$$\begin{bmatrix} 136 & 24 & 20 & 18 & 44 \\ 49 & 19 & 30 & 14 & 30 \\ 25 & 15 & 43 & 32 & 56 \\ 21 & 20 & 32 & 92 & 180 \\ 35 & 15 & 29 & 97 & 924 \end{bmatrix}$$

Classification report:

class	precision	recall	f1-score	support
1	0.51	0.56	0.54	242
2	0.20	0.13	0.16	142
3	0.28	0.25	0.26	171
4	0.36	0.27	0.31	345
5	0.75	0.84	0.79	1100
avg/total	0.57	0.61	0.59	2000

Setup 1.5

Parameter	Value
USE_STOP_WORDS	True
$USE\_EMOTICONS$	True
$USE\_NEGATION$	True
$USE\_BOOLEAN_REPRESENTATION$	True

Confusion matrix:

$$\begin{bmatrix} 135 & 25 & 20 & 16 & 46 \\ 52 & 19 & 17 & 20 & 34 \\ 32 & 9 & 38 & 33 & 59 \\ 26 & 18 & 33 & 77 & 191 \\ 39 & 11 & 21 & 80 & 949 \end{bmatrix}$$

Classification report:

class	precision	recall	f1-score	support
1	0.48	0.56	0.51	242
2	0.23	0.13	0.17	142
3	0.29	0.22	0.25	171
4	0.34	0.22	0.27	345
5	0.74	0.86	0.80	1100
avg/total	0.57	0.61	0.58	2000

As we can see using stop words in sentiment analysis not only does not improve, but even harm the performance. Use of emoticons does not decrease the performance but the gain, if it is present,is too insignificant. This result is applicable only for a subset of the data with $NUMBER\_OF\_REVIEWS\_TO\_ANALYZE = 10000$. For bigger number of words results might be different. Yet we have seen that using boolean representation and negation clearly improve performance.

4.3 Predictors comparison

Using the results from the previous section I have decided to use the following configuration in the next comparison:

Parameter	Value
USE_STOP_WORDS	False
$USE\_EMOTICONS$	False
$USE\_NEGATION$	True
$USE\_BOOLEAN_REPRESENTATION$	True
$NUMBER\_OF\_REVIEWS\_TO\_ANALYZE$	10000
$NUMBER\_OF\_POPULAR\_WORDS\_TO\_USE$	1000

Let's compare performance of Naive Bayes, Logistic Regression and KNN. Let's start with training data to test data ratio 80:20.

Setup 2.1 Naive Bayes
Confusion matrix:

$$\begin{bmatrix} 134 & 23 & 24 & 17 & 44 \\ 50 & 18 & 30 & 15 & 29 \\ 24 & 15 & 42 & 33 & 57 \\ 20 & 20 & 35 & 92 & 178 \\ 36 & 15 & 32 & 90 & 927 \end{bmatrix}$$

Classification report:

class	precision	recall	f1-score	support
1	0.51	0.55	0.53	242
2	0.20	0.13	0.15	142
3	0.26	0.25	0.25	171
4	0.37	0.27	0.31	345
5	0.75	0.84	0.79	1100
avg/total	0.57	0.61	0.59	2000

Setup 2.2 Logistic regression

Confusion matrix:

$$\begin{bmatrix} 124 & 28 & 20 & 19 & 5 \\ 43 & 32 & 26 & 11 & 30 \\ 24 & 26 & 37 & 22 & 62 \\ 18 & 28 & 32 & 88 & 179 \\ 51 & 24 & 30 & 111 & 884 \end{bmatrix}$$

Classification report:

class	precision	recall	f1-score	support
1	0.48	0.51	0.49	242
2	0.23	0.23	0.23	142
3	0.26	0.22	0.23	171
4	0.35	0.26	0.30	345
5	0.73	0.80	0.77	1100
avg/total	0.56	0.58	0.57	2000

Setup 2.3 KNN

Confusion matrix:

$$\begin{bmatrix} 47 & 14 & 12 & 11 & 158 \\ 21 & 10 & 10 & 11 & 90 \\ 15 & 2 & 13 & 20 & 121 \\ 15 & 3 & 14 & 28 & 285 \\ 52 & 21 & 31 & 49 & 947 \end{bmatrix}$$

Classification report:

class	precision	recall	f1-score	support
1	0.31	0.19	0.24	242
2	0.20	0.07	0.10	142
3	0.16	0.08	0.10	171
4	0.24	0.08	0.12	345
5	0.59	0.86	0.70	1100
avg/total	0.43	0.52	0.45	2000

As we can see KNN performs poorly and there is no reason to use it in the future. Therefore, I don't include it in farther comparisons. Now let us demonstrate classification performance with training data to test data ratio 60:40.

Setup 2.4 Naive Bayes

Confusion matrix:

$$\begin{bmatrix} 271 & 35 & 42 & 41 & 82 \\ 104 & 34 & 43 & 46 & 65 \\ 50 & 30 & 68 & 82 & 111 \\ 41 & 36 & 62 & 191 & 366 \\ 84 & 37 & 48 & 196 & 1835 \end{bmatrix}$$

Classification report:

class	precision	recall	f1-score	support
1	0.49	0.58	0.53	471
2	0.20	0.12	0.15	292
3	0.26	0.20	0.23	341
4	0.34	0.27	0.31	696
5	0.75	0.83	0.79	2200
avg/total	0.56	0.60	0.58	4000

Setup 2.5 Logistic regression

Confusion matrix:

$$\begin{bmatrix} 230 & 66 & 56 & 39 & 80 \\ 91 & 53 & 44 & 40 & 64 \\ 41 & 56 & 75 & 69 & 100 \\ 39 & 49 & 81 & 205 & 322 \\ 102 & 62 & 85 & 309 & 1642 \end{bmatrix}$$

Classification report:

class	precision	recall	f1-score	support
1	0.46	0.49	0.47	471
2	0.19	0.18	0.18	292
3	0.22	0.22	0.22	341
4	0.31	0.29	0.30	696
5	0.74	0.75	0.75	2200
avg/total	0.55	0.55	0.55	4000

As can be seen, switching from 80:20 to 60:40 training to test set ratio does not reduce performance significantly which means that those two classifiers generalize well. Naive Bayes seems to perform slightly better than Logistic regression for data of this particular domain.

4.4 Comparison with other models

Let us now use most of the data to compare the quality of my solution with other solutions which apply to the review data I analyze.

For the setting $NUMBER\_OF\_REVIEWS\_TO\_ANALYZE = 100000$, there was no noticeable difference whether use stop words and emoticons or not. I used the following configuration:

Parameter	Value
USE_STOP_WORDS	True
$USE\_EMOTICONS$	False
$USE\_NEGATION$	True
$USE\_BOOLEAN_REPRESENTATION$	True
$NUMBER\_OF\_REVIEWS\_TO\_ANALYZE$	100000
$NUMBER\_OF\_POPULAR\_WORDS\_TO\_USE$	1000

Whenever applicable training data to test data ratio 80:20 is used.

Setup 3.1 Naive Bayes
Confusion matrix:

$$\begin{bmatrix} 1085 & 177 & 155 & 67 & 331 \\ 382 & 165 & 245 & 142 & 335 \\ 243 & 137 & 449 & 401 & 596 \\ 176 & 64 & 318 & 1166 & 1888 \\ 325 & 87 & 168 & 927 & 9971 \end{bmatrix}$$

Classification report:

class	precision	recall	f1-score	support
1	0.49	0.60	0.54	1815
2	0.26	0.13	0.17	1269
3	0.34	0.25	0.28	1826
4	0.43	0.32	0.37	3612
5	0.76	0.87	0.81	11478
avg/total	0.61	0.64	0.62	20000

Setup 3.2 Logistic regression

Confusion matrix:

$$\begin{bmatrix} 1067 & 157 & 117 & 42 & 432 \\ 394 & 169 & 200 & 96 & 410 \\ 214 & 166 & 405 & 310 & 731 \\ 88 & 79 & 274 & 922 & 2249 \\ 133 & 66 & 123 & 501 & 10655 \end{bmatrix}$$

Classification report:

class	precision	recall	f1-score	support
1	0.56	0.59	0.58	1815
2	0.27	0.13	0.18	1269
3	0.36	0.22	0.28	1826
4	0.49	0.26	0.34	3612
5	0.74	0.93	0.82	11478
avg/total	0.61	0.66	0.62	20000

Now let us compare my solution with predictor based on a prepared lexicon. In particular the lexicon of positive and negative words in English at [8] is used. The formula of prediction is $prediction = 1+round(4*(positivecount/(positivecount+negativecount))$. Now the $NUMBER\_OF\_REVIEWS\_TO\_ANALYZE$ is 100000. Since this alorithm does not require learning, the whole dataset is used as a test set. The results are bellow:

Setup 3.3 Positive and negative word count

Confusion matrix:

$$\begin{bmatrix} 1169 & 1818 & 3353 & 1683 & 897 \\ 413 & 807 & 2345 & 1762 & 997 \\ 381 & 707 & 2850 & 3015 & 2204 \\ 296 & 736 & 4224 & 7058 & 5977 \\ 707 & 1296 & 9185 & 18505 & 27615 \end{bmatrix}$$

Classification report:

class	precision	recall	f1-score	support
1	0.39	0.13	0.20	8920
2	0.15	0.13	0.14	6324
3	0.13	0.31	0.18	9157
4	0.22	0.39	0.28	18291
5	0.73	0.48	0.58	57308
avg/total	0.52	0.39	0.43	100000

For this dataset there is a list of words suggested for binary sentiment analysis. I have tried using it for this purpose and it produced fairly good results. Now let us compare how it performs at more accurate classification task and compare its performance with the performance of my solution.

Setup 3.4 Suggested words, Naive Bayes

Confusion matrix:

$$\begin{bmatrix} 511 & 9 & 10 & 0 & 1285 \\ 214 & 13 & 20 & 1 & 1021 \\ 148 & 8 & 20 & 4 & 1646 \\ 76 & 4 & 21 & 3 & 3508 \\ 153 & 7 & 25 & 7 & 11286 \end{bmatrix}$$

Classification report:

class	precision	recall	f1-score	support
1	0.46	0.28	0.35	1815
2	0.32	0.01	0.02	1269
3	0.21	0.01	0.02	1826
4	0.20	0.00	0.00	3612
5	0.60	0.98	0.75	11478
avg/total	0.46	0.59	0.46	20000

Setup 3.5 Suggested words, Logistic regression

Confusion matrix:

$$\begin{bmatrix} 510 & 9 & 4 & 4 & 1288 \\ 219 & 11 & 9 & 13 & 1017 \\ 144 & 7 & 14 & 10 & 1651 \\ 63 & 8 & 10 & 22 & 3509 \\ 141 & 2 & 5 & 18 & 11312 \end{bmatrix}$$

Classification report:

class	precision	recall	f1-score	support
1	0.47	0.28	0.35	1815
2	0.30	0.01	0.02	1269
3	0.33	0.01	0.01	1826
4	0.33	0.01	0.01	3612
5	0.60	0.99	0.75	11478
avg/total	0.50	0.59	0.47	20000

We can see that my algorithm performs significantly better than aforementioned sentiment analysis alorithms, which perform good on binary polarity classification problem.

Conclusion

Words.

Algorithms which suit well for binary polarization task, perform very poor on 5 degree classification.

Product reviews, etc are relatively easy

I learned a lot from this project. I'm so glad that I was able to pull together what I've learned in several different areas to work on one unified program.

Taking a look at my results, though, I have a few comments.

My classification results aren't that great. I had hoped to get much higher than a 60-65% correct classification rate. That's quite alright with me, though; it means I have plenty of room to improve this program! I could try different classification algorithms, different wordlists, even different training data. There's lots of space to work.

My code isn't particularly pretty, and I would like to change that. I've just started to read Google's Style Guide for R and I've noticed a few things I do wrong. I'll set out to fix those in later versions of this program if I choose to move forward with it.

References

[1] Manning, Chris and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. Cambridge [England]: MIT Press, 1999.

[2] Wake, Lisa. *NLP : principles in practice*. St. Albans, Herts [England]: Ecademy Press, 2010.

[3] Murphy, Kevin. *Machine Learning A Probabilistic Perspective*. Cambridge, Massachusetts [USA]: Massachusetts Institue of Technology, 2012.

[4] Leskovec, Jure, Anand Rajaraman, and Jeff Ullman. *Mining of Massive Datasets SECOND EDITION*. Cambridge [England]: Cambridge University Press, 2014.

[5] Witten, Ian, and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques THIRD EDITION*. Burlington [USA]: Morgan Kaufmann Publications, 2010.

[6] McKinney, Wes. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython 1st Edition*. Sebastopol [USA]: 2012.

[7] *Microsoft Naive Bayes Algorithm*. Microsoft, 2016. Web. 26 May 2016. <https://msdn.microsoft.com/en-us/library/ms174806.aspx>.

[8] *Opinion Mining, Sentiment Analysis, and Opinion Spam Detection*. University of Illinois at Chicago, Web. 26 May 2016. <http://www.cs.uic.edu/ liub/FBS/opinion-lexicon-English.rar>.

[9] *Sentiment Symposium Tutorial: Tokenizing*. Christopher Potts, 2011. Web. 26 May 2016. <http://sentiment.christopherpotts.net/tokenizing.html>.

[10] Minqing, Hu, and Liu Bing. *Mining and Summarizing Customer Reviews*. Seattle [USA]: 2004.

[11] Carlos Guestrin, Amazon Professor of Machine Learning. Web. 26 May 2016. <https://s3.amazonaws.com/static.dato.com/files/coursera/course-3/amazon_baby.csv.zip>.

[12] Turney, Peter. *Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews.* Philadelphia, Pennsylvania [USA]: 2002.

[13] Bing, Liu, Hu Minqing, and Junsheng Cheng. *Opinion Observer: Analyzing and Comparing Opinions on the Web.*. Chiba [Japan]: 2005.

[14] Bishop, Chrispopher. *Pattern Recognition and Machine Learning.* Singapore: Springer Science Business Media, 2006.