Report

Introduction

How to implement statistical classifiers which will perform the task of classifying documents. It should be able to classify in and out of domain sentiment as well as text categorization.

This report will present three different ways of doing the classification, Naive Bayes, Perceptron (and Average Perceptron) and k-Nearest Neighbors.

To validate our results we perform 10-fold cross-validation.

Naive Bayes

Bayes' theorem:

$$P(C \mid F_{1,\dots}, F_n) = \frac{P(C) * P(F_{1,\dots}, F_n \mid C)}{P(F_{1,\dots} F_n)} \quad \text{or} \quad posterior = \frac{prior * likelihood}{evidence}$$

We will calculate Naive Bayes to determine the chance that a document is of class C1 and the chance that it is C2. We will then compare the two posteriors to see which of them is higher. The higher posterior will be the probable class.

Prior and Evidence will be the same for all tests. Thus, they are not needed in the final comparison of the posteriors. This means that all we need to compare is the likelihood between the two classes. The likelihood will be calculated like this:

$$\text{likelihood} = \{ \quad P(C_{d_i} = c_i | d_i) = \prod_{\text{w in } d_i} \frac{\#\text{w in trainData of class } c_i}{\#\text{words in bag of class } c_i} \quad \}$$

Perceptron

The algorithm returns a hyperplane which separates the classes. It learns the hyperplane by iterating though all points and if it finds a point which is on the wrong side of the plane it adjusts the plane.

If the learning set is not linearly separable the perceptron will not terminate. Therefor we only iterate a specified number of times N.

Pseudo-code:

```
Perceptron(DOCUMENTS, LABLES):
      w = array with random values between 0 and 1 of size one DOCUMENT
      N times do
            for i in DOCUMENTS:
                  if  sign(w * DOCUMENTS(i)) != LABLES(i):
                        w = w + alpha * LABLES(i) * DOCUMENTS(i)
      return w
```

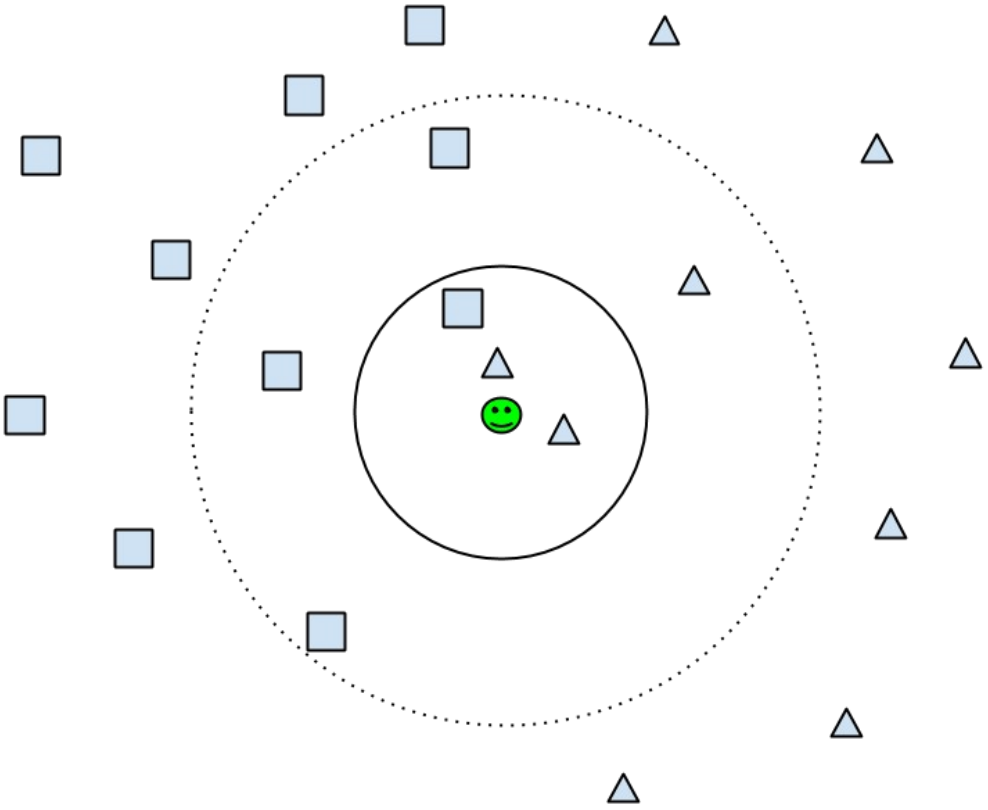
Average Perceptron

One problem the normal Perceptron has is that it is biased toward the latest of the documents in the learning set it has processed. To avoid that we also use a Average Perceptron which saves all vectors and returns the average weight vector over all the ones it has seen instead of just the last one.

k-Nearest Neighbors

As the third algorithm we are using the k-Nearest Neighbors algorithm which only locally looks at the neighbors of the document in a radius predefined by the user. It assumes that the document is of the same category as the highest number of neighbors within this radius.

In the following figure you can see that depending if the user choses k = 3, as showed by the solid line, the algorithm will conclude that the document in the center (green smiley) is of the type triangle because most of this three neighbors are triangles. If on the other hand the user choses k = 7, as showed by the dotted line, then the amount of neighbors which are rectangles is greater as the amount of neighbors which are triangles, so it concludes that the smiley is of type rectangle.



k-fold cross-validation

We are using 10 fold cross-validation for every test we are doing which means that we use 90% of the

available data for the learning algorithms and then the remaining 10% to test how they perform. We repeat this ten times until all data has been used for testing once.

Architecture

Representation

The data is represented in a matrix, where each row represents a given document and each column represents a word. The value in matrix cell (x,y) represents the number of occurrences of the word y in the document x.

In order to extract our data, we will iterate each document and build a vocabulary consisting of all unique words. In order to avoid duplicate words (for example “dont” and “don’t”), we trim each word – removing characters that are not letters. We will also remove all words that are represented in a list of stop words that we downloaded from a MySQL database.

When this is done, we filter the remaining data into three different features.

- Binary feature – here we use a sign(count) to fill our matrix with 0 for no occurrence and 1 for any amount of occurrences.
- Normal feature – here we simply count the number of occurrences of each given word.
- TF-IDF feature – We filter the data using TF-IDF, removing all words below a certain weight.

For Binary and Normal feature we remove all words used less than 5 times, in order to keep the data reasonably small and computable.

Categorization

A categorization algorithm is an algorithm f that given input data x calculates the type of the input $y = f(x)$.

Naive Bayes

As described in the introduction all we need to calculate is the likelihood for each class.

likelihood = { $P(C_{d_i} = c_i | d_i) = \prod_{\text{w in } d_i} \frac{\#\text{w in trainData of class } c_i}{\#\text{words in bag of class } c_i}$ }

Because the product $\prod_{\text{w in } d_i}$ will become a very small number, we will log the numerator and the denominator and instead use the formula:

$$P(C_{d_i} = c_i | d_i) = \sum_{\text{w in } d_i} \log(\#\text{w in trainData of class } c_i) - \log(\#\text{words in bag of class } c_i)$$

After doing this, we can compare $P(C_{d_i} = c_1 | d_i)$ and $P(C_{d_i} = c_2 | d_i)$ and by doing so, we find our most probable classification.

Perceptron

Definition of terms:

$D = \{d_1, d_2, \ldots, d_s\}$ Are the documents

$L = \{l_1, l_2, \ldots, l_s\}$ Are the documents labels (1 or -1)
$d_j = \{x_1, x_2, \ldots, x_p\}$ A documents consists of frequencies of words
$w = \{w_1, w_2, \ldots, w_s\}$ Is a weight vector
α Is the chosen learning rate
i_{max} is the maximal number of iterations
i is the current iteration

Algorithm:

The algorithm for the normal perceptron uses this learning rule:

$$w_j \leftarrow w_j + \alpha\, x_j\, l$$

The algorithm for the averaged perceptron also uses this learning rule:

$$wa_j \leftarrow wa_j + \alpha\, x_j\, l\, \frac{i_{max}|\vec{x}| - i}{i_{max}|\vec{x}|}$$

The learning rules are applied only if $sign(w_j\, x_{dj}) = -l_d$

It iterates over all words, all documents i_{max} times every time updating $\alpha = \frac{i_{max} - i}{i_{max}}$.

K-nearest neighbor

The K-nearest neighbor algorithm classifies the document according to its nearest k neighbors.

We calculate the euclidean distance between the document and all the other documents in the n-dimensional space. We sort them and select the k smallest neighbors.

So if the majority of those neighbors are for example positive reviews then it assumes that this document is also positive.

Instead of implementing it ourselves we used the implementation we found build in in Matlab.

Results

In-domain sentiment analysis

Category	Perceptron	Perceptron avrg.	Naive Bayes	k-Nearest
Books	66,55%	63,85%	78,22%	51,85%
DVDs	68,90%	67,25%	79,19%	50,35%
Music	65,57%	65,45%	78,22%	52,20%
Health	68,40%	66,75%	79,33%	56,40%
Software	76,22%	73,83%	82,67%	51,22%
Camera	76,55%	74,40%	85,39%	54,00%

Table 1: Using the binary feature

Category	Perceptron	Perceptron avrg.	Naive Bayes	k-Nearest
Books	65,05%	64,05%	77,33%	55,00%
DVDs	67,35%	67,15%	77,61%	54,00%
Music	66,35%	65,70%	77,56%	53,35%
Health	68,90%	68,60%	79,00%	58,65%
Software	75,28%	74,11%	82,33%	57,94%
Camera	76,05%	74,10%	84,28%	57.35%

Table 2: Using the frequency feature

Category	Perceptron	Perceptron avrg.	Naive Bayes	k-Nearest
Books	51,90%	51,40%	63,28%	55,50%
DVDs	54,95%	55,00%	64,39%	51,85%
Music	53,75%	52,95%	65,67%	54,35%
Health	56,85%	55,55%	64,61%	56,15%
Software	67,44%	66,17%	70,56%	59,83%
Camera	56,90%	64,50%	73,11%	59,60%

Table 3: Using the tf-idf feature

Out of domain sentiment analysis

Learn category	Test category	Perceptron	Perceptron avrg.	Naive Bayes	k-Nearest
Books	Health	57,00%	57.80%		
Health	Books	58,50%	56.75%		

Table 4: Using the binary feature

Learn category	Test category	Perceptron	Perceptron avrg.	Naive Bayes	k-Nearest
Books	Health	56,35%	59.95%		
Health	Books	47,40%	55.75%		

Table 5: Using the frequency feature

Learn category	Test category	Perceptron	Perceptron avrg.	Naive Bayes	k-Nearest
Books	Health	49,70%	47.75%		
Health	Books	46,30%	49.35%		

Table 6: Using the tf-idf feature

Category analysis

Naive Bayes

We test each column against each row, the value in each cell represents the correctness rate of the classification.

	Book	Camera	DVD	Health	Music	Software
Book	-	98.86%	93.83%	97.86%	97.92%	96.08%
Camera	98.86%	-	98.11%	95.22%	99.03%	97.28%
DVD	93.83%	98.11%	-	97.39%	94.61%	96.83%
Health	97.86%	95.22%	97.39%	-	98.58%	97.08%
Music	97.92%	99.03%	94.61%	98.58%	-	98.06%
Software	96.08%	97.28%	96.83%	97.08%	98.06%	-

Table 7: Using the binary feature

	Book	Camera	DVD	Health	Music	Software
Book	-	99.06%	93.92%	97.94%	98.08%	96.31%
Camera	99.06%	-	98.39%	95.31%	99.06%	97.36%
DVD	93.92%	98.39%	-	97.44%	94.92%	96.94%
Health	97.94%	95.31%	97.44%	-	98.78%	96.94%
Music	98.08%	99.06%	94.92%	98.78%	-	98.06%
Software	96.31%	97.36%	96.94%	96.94%	98.06%	-

Table 8: Using the frequency feature

	Book	Camera	DVD	Health	Music	Software
Book	-	98.14%	92.92%	95.36%	96.68%	95.17%
Camera	98.14%	-	97.58%	93.11%	98.08%	96.19%
DVD	92.92%	97.58%	-	95.25%	93.97%	95.89%
Health	95.36%	93.11%	95.25%	-	95.94%	94.06%
Music	96.67%	98.08%	93.97%	95.94%	-	96.75%
Software	95.17%	96.19%	95.89%	94.06%	96.75%	-

Table 9: Using the tf-idf feature

Perceptron

The columns represent how the texts were categorized, and the rows what was categorized. For example 1,58% of DVDs have been categorized as Books.

	Books	DVD	Camera	Music	Health	Software
Books	84,21%	0%	3,16%	2,11%	2,63%	2,63%
DVD	0%	82,11%	0	5,79%	0	6,84%
Camera	3,16%	0,53%	80,53%	2,63%	4,21%	3,68%
Music	3,16%	2,63%	2,63%	77,37%	0	8,95%
Health	0,53%	0%	2,63%	2,11%	86,84%	2,63%
Software	0%	2,11%	0	3,16%	1,05%	88,42%

Table 10: Using the binary feature

	Books	DVD	Camera	Music	Health	Software
Books	84,368%	1,05%	0%	1,58%	4,21%	9,47%
DVD	1,58%	78,42%	1,58%	2,63%	5,79%	10%
Camera	0%	1,05%	84,74%	0%	5,79%	8,42%
Music	0%	3,68%	1,05%	85,79%	3,16%	6,32%
Health	0,53%	0,53%	6,32%	0%	77,3%	15,26%
Software	0%	0,53%	1,58%	1,05%	2,11%	94,74%

Table 11: Using the frequency feature

	Books	DVD	Camera	Music	Health	Software
Books	80,53%	0%	4,74%	2,63%	1,05%	5,79%
DVD	0%	83,68%	0,53%	7,37%	0%	3,16%
Camera	2,63%	2,11%	81,58%	1,58%	3,68%	3,16%
Music	2,11%	3,68%	2,63%	77,37%	1,58%	7,37%
Health	0,53%	0%	2,63%	2,63%	86,32%	2,63%
Software	0%	2,11%	0	2,11%	1,05%	89,47%

Table 12: Using the tf-idf feature

Perceptron avrg.

The columns represent how the texts were categorized, and the rows what was categorized. For example 2,11% of DVDs have been categorized as Books.

	Books	DVD	Camera	Music	Health	Software
Books	86.84%	2.26%	0.53%	0.53%	1.58%	2.63%
DVD	2.11%	82.11%	1.58%	2.63%	2.63%	3.68%
Camera	0.00%	1.58%	82.63%	0.00%	5.79%	4.75%
Music	0.53%	3.16%	0.53%	83.68%	3.16%	3.68%
Health	0.53%	2.11%	7.37%	0.00%	80.53%	4.21%
Software	0.00%	1.05%	3.68%	1.05%	2.63%	86.32%

Table 13: Using the binary feature

	Books	DVD	Camera	Music	Health	Software
Books	86.84%	2.11%	0.53%	0.53%	1.58%	3.16%
DVD	3.16%	81.58%	1.05%	3.68%	2.63%	2.63%
Camera	1.05%	0.53%	86.35%	0.00%	3.68%	3.16%
Music	0.53%	3.16%	0.53%	84.20%	3.68%	2.63%
Health	0.00%	1.05%	5.79%	0.53%	82.63%	4.74%
Software	0.53%	0.53%	1.58%	0.53%	2.11%	89.47%

Table 14: Using the frequency feature

	Books	DVD	Camera	Music	Health	Software
Books	82.11%	5.26%	0.00%	1.05%	2.63%	3.68%
DVD	2.63%	81.58%	0.53%	1.58%	4.21%	4.21%
Camera	0.00%	0.00%	82.63%	0.53%	7.37%	4.21%
Music	0.00%	4.21%	0.00%	85.79%	2.11%	2.63%
Health	1.05%	2.11%	4.74%	2.63%	75.26%	8.95%
Software	0.00%	0.00%	1.05%	1.05%	1.05%	91.58%

Table 15: Using the tf-idf feature

K-nearest neighbor

	Books	DVD	Camera	Music	Health	Software
Books						
DVD						
Camera						
Music						
Health						
Software						

Table 16: Using the binary feature

	Books	DVD	Camera	Music	Health	Software
Books						
DVD						
Camera						
Music						
Health						
Software						

Table 17: Using the frequency feature

	Books	DVD	Camera	Music	Health	Software
Books						
DVD						
Camera						
Music						
Health						
Software						

Table 18: Using the tf-idf feature

Conclusion

Naive Bayes

As our results show, all of the features are performing well for category analysis. Even though the frequency-, and binary features are slightly higher but the tf-idf feature is more than twice as fast due to a smaller vocabulary. Therefore, we feel that depending on the situation, it might be worth to sacrifice some of the correctness rate in favor of speed when doing category analysis.

For in domain sentiment analysis, however, the tf-idf does not perform well compared to the two other features. This is likely because it removes words that are frequently used in different categories, such as words that describe a good or bad product without specifying the type of the product. In these cases, it is probably not worth the loss of correctness in favor of speed. Especially when considering that the amount of data we feed the algorithm with is already half the size compared to category analysis (In sentiment analysis we feed only positive and negative data for one category, while for category analysis we need to send positive and negative data for both the categories used).

The out of domain sentiment has not yet been tested...

We understand that we can not draw any conclusions until we have finished all the tests. This is just an example how we want to present our conclusion.

Perceptron

Looking at the results for the Perceptron we see that all of the features perform well for category analysis, there are basically no significant differences. Due to the smaller vocabulary the tf-idf feature runs much faster so because of the fact we recommend using the tf-idf feature when the Perceptron is used for categorization.

For in domain sentiment analysis we see that the binary feature performs best, tight behind it is the frequency feature and the tf-idf feature obviously performs worst because it filters out all the important words which express the sentiment before the data is passed to the Perceptron.

We also see that the Perceptron algorithm performs about 10% worse then the Naive Bayes algorithm in almost all tests.

Average Perceptron

Over all the average perceptron performs slightly worse for in domain analysis then the normal one. For out of domain and category analysis it performs better but still not as good as the Naive Bayes algorithm.

k-Nearest Neighbor

The k-Nearest Neighbor performs really bad for in domain sentiment analysis, with around 50% getting it right it is about as good as random chance. We think it is due to the fact that there is not a big gap in between the positive and negative documents but instead all of them are overlapping quite much like you would expect when categorizing natural language texts.

Category