DIT867/DAT340 Assignment 3

Calvin Smith
University of Gothenburg
gussmica@student.gu.se

Bragadesh Bharatwaj Sundararaman
Chalmers University
brabha@student.chalmers.se

Amogha Udayakumar
Chalmers University
amogha@student.chalmers.se

Abstract

The project aims at classifying the restaurant reviews crowdsourced from different websites by the students in this course as positive or negative using a supervised machine learning approach.

1 Introduction

In this report we will investigate the use of a naive Bayes classifier on a dataset consisting of restaurant reviews collected and annotated by students at Chalmers and labeled as either positive, negative or indecisive. In Chapter 2, we give a brief theoretical background on the naive Bayes classifier and our evaluation method. In Chapter 3, we describe the method of cleaning and processing our data as well as feature processing and model selection. Finally, the results are presented in Chapter 4 and the following discussion in Chapter 5.

2 Theory

2.1 naive Bayes classifier

The naive Bayes classifier is a machine learning algorithm commonly used in text classification. The algorithm is simple and fast to implement while often outperforming more complex algorithms. Given an observation, the algorithm assigns the most likely class based on its feature vector. The features are assumed to be independent given class, which is why the classifier is referred to as *naive*. Given a class y and features $x_1, x_2, ..., x_n$, Bayes theorem gives us that the probability of the class given the features is

$$P(y|x_1, ..., x_n) = \frac{P(y)P(x_1, ..., x_n|y)}{P(x_1, ..., x_n)},$$

where $P(y)$ is the prior probability of class y, $P(x_1, ..., x_n|y)$ is the conditional probability of the features given the class and $P(x_1, ..., x_n)$ is the joint probability of the features. Using the conditional independence assumption we can expand the numerator into separate conditional probabilities, and since the joint probability of the features is not dependent on the class, we can use proportionality to obtain

$$P(y|x_1, ..., x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y),$$

and the classification rule becomes,

$$\text{class}(\hat{y}) = \arg\max P(y) \prod_{i=1}^{n} P(x_i|y),$$

that is, $\hat{y}$ is assigned to the class that produces the highest probability.

2.2 Evaluation method

To evaluate and analyze the performance of our model we will be utilizing the confusion matrix together with the accuracy. The simplest form of a confusion matrix is a 2×2 matrix with columns corresponding to the true values and the rows corresponding to the predicted values:

$$\begin{array}{cc|c|c|}
 & & \multicolumn{2}{c}{\textbf{True}} \\
 & & 1 & 0 \\
\cline{3-4}
\textbf{Pred} & 1 & \text{TP} & \text{FP} \\
\cline{3-4}
 & 0 & \text{FN} & \text{TN} \\
\cline{3-4}
\end{array}$$

The accuracy is defined as the fraction of correctly predicted observations (regardless of class) divided by the total number of observations:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \qquad (1)$$

Since our data is more or less balanced between classes and that we value true positives and true negatives equally, the accuracy is a suitable metric for evaluating the performance of our models. During the model selection we also kept track of the per class accuracy's (recall and specificity) in case any of the models would show bias towards a certain class.

3 Method

3.1 Data

The data consists of restaurant reviews that have been annotated twice by two independent annotators. The annotators have classified each review as either positive (1), negative (0) or indecisive (-1) if the annotator cannot decide whether a review is positive or negative. The data has been separated into a training set consisting of 7018 reviews and a testing set of 1751 reviews. The testing set has not been annotated twice and only contains the "true" label of either 0 or 1.

Further, out of the 7018 reviews in the training set, 396 of them are reviews where annotators disagree. As a first pre-processing step, these reviews are removed from the training data. The training set now consists of 6622 reviews, out of which 47% are negative reviews and 53% are positive reviews. After removing the data with disagreements we obtain a clean and reliable dataset. The testing set consists of 1751 reviews with equally many positive as negative reviews. The final step before moving forward with the classification task is to split the training data into two new sets, a new training set and a validation set. These two sets will be used for building our model before moving on to evaluating on the test set. The two new sets are created by randomly splitting 80% of the data into the new training set and 20% of the data into the validation set, while preserving the class balance of the full training set.

3.2 Feature processing

After cleaning the data and doing the preprocessing, we proceeded with extracting the features for the naive Bayes classifier. Since the naive Bayes classifier is an algorithm that works well with numerical data in the form of matrices we convert our text document data into a TF-IDF matrix. For this we utilized Scikit learn's TfidfVectorizer. The TfidfVectorizer is a feature extractor that computes the feature values by multiplying the term frequency (the number of times a term appears in a document) with the inverse document frequency. Here the output of the method is a matrix with the number of rows equal to number of reviews and the columns are the unique words in the case of unigrams or also unique word combinations in the case of bigrams and so on. Thus, based on the above mentioned calculation the text document is converted to a numerical matrix. This gives a better feature representation than CountVectorizer, thus it was the reason we chose the former. We perform fit and transform and obtain the features for the training set, while we perform transform to obtain the features for the test set. These are the features we input to the naive Bayes classifier to train the model and then predict on the test set.

3.3 Implementation

There are different versions of the naive Bayes classifier that are more or less suitable depending on the task. In this project we will be looking at the multinomial naive Bayes classifier since we are representing our features as a numerical matrix and because it outperformed other versions (like bernoulli) during preliminary model exploration. The multinomial naive Bayes classifier assumes that the features are multinomially distributed. The classifier takes our TF-

IDF matrix Z as input, where each row represents a review and its corresponding class label (0 or 1) and each column represents a feature n-gram. Each element z_{ij} represents the TF-IDF score of n-gram j from review i. We have two classes, positive and negative belonging to the set $C = \{positive, negative\}$. Let c represent an arbitrary class in our data. The probability of n-gram j belonging to class c is denoted by π_{cj} and estimated by:

$$\hat{\pi}_{cj} = \frac{K_{cj} + \alpha}{K_c + \alpha N}$$

where:

- $K_{cj} = \sum_{z_{ij} \in c} z_{ij}$ is the total TF-IDF score of n-gram j in reviews belonging to class c.
- $K_c = \sum_{j=1}^{N}$ is the sum of all TF-IDF scores for class c
- N is the total number of n-grams
- α is a smoothing parameter for n-grams that are not present in the training data i.e preventing zero probabilities. Often $\alpha = 1$ but this is a parameter that can be fine tuned.

Further let x_j, $j = 1, 2, ..., N$ represent the feature n-grams and let y_i, $i = 1, 2, ..., R$ represent each review, where R is the total number of reviews. The probability that review y_i belongs to class c given the feature n-grams x_j is then estimated by:

$$P(y_i \in c | x_j) = \hat{\theta}_c \prod_{j=1}^{N} \hat{\pi}_{cj}$$

where $\hat{\theta}_c = \dfrac{\sum_{y_i \in c} y_i}{\sum y_i}$ is the fraction of reviews belonging to class c. Further, the class that maximizes the probability is chosen:

$$\text{class}(y_i) = \arg\max_{c \in C} \ \hat{\theta}_c \prod_{j=1}^{N} \hat{\pi}_{cj}$$

3.4 Model selection

The features used in our naive Bayes classifier are from the TF-IDF matrix. The TF-IDF matrix is in turn based on n-gram counts, which is defined as a sequence of n words (or tokens). Depending on the length of the n-gram different names are used, a sequence of one word is called a unigram, two words is called a bigram, three words trigram and so on. When converting the data into a TF-IDF matrix the choice of how many different n-grams to use is a hyperparameter that can be tuned. For example, we can use only unigrams to construct the TF-IDF matrix features, or we can use both unigrams and bigrams, or even unigrams, bigrams and trigrams.

To obtain a good model we performed a parameter search over a specified range of n-grams and training the naive Bayes classifier on the temporary training set and evaluating on the validation set. To minimize computation times the range of n in n-grams was limited to four. In a similar way, the smoothing parameter α is fine tuned using a set of different values.

4 Results

4.1 Best model

The n-gram range that produced the highest validation accuracy was $(1, 2)$, meaning that the features consist of both unigrams and bigrams, the validation accuracy obtained was 0.967. Further, Figure 1 is a plot of the validation accuracy as a function of the smoothing parameter α. As we can see the highest validation accuracy is obtained with $\alpha = 0.2$. Thus, the best model based on the validation accuracy is a multinomial naive Bayes classifier with $\alpha = 0.2$ and n-gram range $= (1, 2)$. The best validation accuracy using this model is 0.97.

4.2 Final evaluation

In the final evaluation of our model, we train it on the entire training set and evaluate on the testing set. The accuracy obtained was 0.966 and the per class accuracies are 0.965

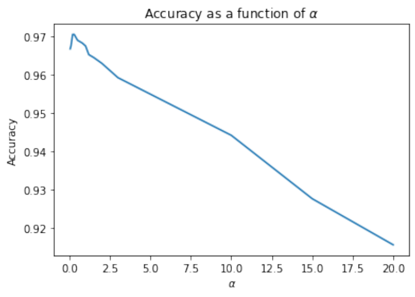

Figure 1: Accuracy as a function of α

for positive reviews and 0.968 for negative reviews. Figure 2 is a plot of the confusion matrix.

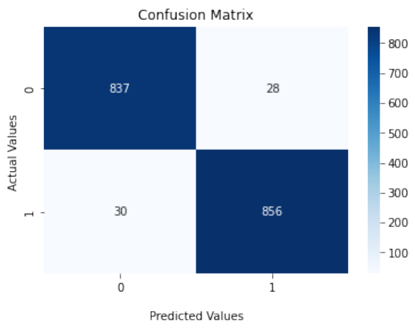

Figure 2: Confusion matrix

5 Discussion and conclusion

Overall, our naive Bayes classifier works well. Considering that the algorithm is quite simple, meaning that we are essentially only using frequencies of n-grams as features, obtaining an accuracy of 0.96 is impressive. When we analyze the confusion matrix, we could find that our model made some errors. So, to understand why it made those mistakes we fetched all the reviews that were misclassified and analyzed some of them. A few examples are shown here:

```
Actual Value: 1 Prediction: 0

Text: Great burgers. I heard
some pretty bad press about
this place, but everything
from food to staff and
interior was more than
good enough.
```

```
Actual Value: 1 Prediction: 0

Text: Certainly not for Asian
food gourmets. Tasteless
soups, uncomfortable seating,
but great decor and sound
track reproducing Tokyo
suburb cheap joint.

Actual Value: 1 Prediction: 0

Text: This used to be a
Priority-go-to even though
their pesto and meatballs
contained pine nuts. The
staff were always very
helpful and knowledgeable
and we felt comfortable
there. They also double
check all of the orders with
the chef. On our last visit
we noticed some changes to
the menu and an addition of
Mortadella Di Bolognawith
with pistachios. This caused
some discomfort for several
reasons. Have not been back
since.
```

When looking at the misclassified reviews, a common theme is that our model has a hard time classifying reviews containing both positive and negative sentiments. These are reviews that even a human would have a hard time classifying as certainly positive or certainly negative. However, a review can be overall positive but still contain some negative, or vice versa. This is something that naive Bayes has a hard time capturing. Some of the misclassified reviews should probably have been labelled as indecisive. However, a more sophisticated language model that uses word embeddings to capture semantics and context might have been able to accurately predict an 'uncertain' review.