

An Implementation of a Naive Bayes Classifier on Text Classification

Jarrett Decker* and Jon David*

Abstract—Abstract goes here. Do we really need an abstract? Keep this example here so we know how to cite references [1]. Keep this here so we know how to do embedded math \LaTeX like $\mathcal{O}(\log_v n)$ where v is blah blah blah.

I. INTRODUCTION

Nave Bayes classifiers are a powerful, yet relatively simple, method of doing classification. By leveraging Bayes theorem, the difficult problem of predicting a class based off of a group of attributes can be transformed into the easier task of estimating the chance of those attributes appearing, given a specific class. This is the heart of the Nave Bayes classification algorithm.

Compared to humans, computers can parse documents incredibly fast, but knowing what the documents are about has traditionally been something only humans can do reliably. With the progress made in machine learning computers can now classify documents at reasonable accuracies. The task of this project was to classify documents into one of twenty newsgroups using a Nave Bayes classifier. Our model did not take word position into account and treated documents as a bag of words.

Nave Bayes is a good choice for this type of problem because the simplicity of classification allows us to easily handle large data sets, such as all words in the English language. Nave Bayes classifiers can also be tweaked through changing beta values in the MAP estimation, or manipulating the data set through things like deleting certain stop words, which could confuse the classifier. This allows you the flexibility to tune your classifier to your problem, making this algorithm useful for many applications, including document classification.

II. DESIGN AND IMPLEMENTATION

The main features of the Nave Bayes classifier are the MLE, MAP, and classification functions. Our design modularized these functions through object oriented programming methods, as well as utility functions.

III. DATA STRUCTURES

Our data necessitated the use of pythons numpy arrays to represent our calculated probabilities. By representing our data in this way, the classification function became a simple addition and matrix multiplication. Along with numpy arrays we also saved our MLE and MAP values to .model files, so we could capture these values and not have to retrain the system every time a classification was run.

A. Naive Bayes Training

Our design implemented a Trainer class that was in charge of reading in the data and labels and then creating the MLE and MAP models that our classifier uses. This class only has to train once to create the model files and those files are available for classification without having to be retrained.

The first function of training is to calculate the MLE of the training labels. The purpose of this is to determine the probability of any label appearing for an unknown document. This probability has nothing to do with what words are associated with what labels, just the rate at which labels appear in the training set. The MLE equation is defined in 1

$$P(Y_k) = \frac{\text{number.of.documents.labeled}.Y_k}{\text{total.number.of.documents}} \quad (1)$$

The second function of training is to determine the MAP of the attributes of the training set. The MAP function allows you to adjust your results to either more resemble what you found in the training set, or to a prior estimation that you can set through the beta value. This beta value is very important, it makes it possible to classify with attributes that you may not have seen for any class before. Usually if you see an attribute that has never been seen belonging to a class, the probability of the unknown class equaling that class will now be zero, but by using the prior beta value instead of zeroing out the probability you can just set it to something small. Since chances are high that classifying combinations of attributes that were not present in the training set, having a beta protects you from having a classification saying there is a zero percent chance of the attributes belonging to any class.

The MAP equation is defined in Figure 2.

$$P(X_i|Y_k) = \frac{(\text{count.of}.X_i.\text{in}.Y_k) + (\alpha - 1)}{(\text{words.in}.Y_k) + ((\alpha - 1)(\text{length.of.vocab}))} \quad (2)$$

Where $\alpha = 1 + \beta$.

B. Naive Bayes Classification

We implemented a NaiveBayesClassifier class that was able to use the MLE and MAP from the trainer to do the calculations needed to classify new data. The classifier writes its classifications to a text file in order to be compared to the truth data for the determination of accuracy. We use Equation 3 to classify words as belonging to one of the newsgroups.

$$Y_{new} = \text{argmax}(\log_2(\vec{a}) + \vec{b} \times \log_2(C)) \quad (3)$$

Where \vec{a} is an MLE vector, \vec{b} is a vector of new words to classify, and C is a matrix produced by calculating MAP over the set of pairs of all words and newsgroups.

C. Confuion Matrix and Accuracy

A confusion matrix is useful for visualizing the results of classification. Accuracy can also be determined from the confusion matrix. We created a ConfusionMatrix class that can read the predicted values that the classifier determined, and the true values from the data set and then increments an element in the confusion matrix associated with those values. For example, if the predicted label was 4 and the true label was 5, the element indexed by [4][5] would be incremented. It follows that elements lying along the diagonal are all accurate predictions while elements off of the diagonal are incorrect. By totaling the values along the diagonal and dividing by the sum of all the values in the matrix the total accuracy of the predictions can be found.

D. Utility Clases

To help with the other parts of the program, we added some utility classes. The Vocabulary class was used to store all possible words in our vocabulary and was capable of producing words associated with word IDs and vice versa. The NewsGroups class was similar, except instead of storing information about the words, it stored information about the possible labels, including the size of the set of labels and the names and IDs.

E. Misc. ~~BTX~~code

This section contains misc. stuff we might to copy/modify later if we want to use equations, algorithms, and tables.

MAKE SURE THIS SECTION GETS DELETED BEFORE SUBMITTING

Algorithm 1 calculate information gain(A, S)

```

    ▷ %Input: attribute A, dataset S%
    entropybefore ← Entropy(S)
    entropyafter ← 0.0
    for v ∈ vals(A) do
        Let  $S_v \subseteq S$ , such that  $S[A] = v$ 
        localentropy ←  $\frac{|S_v|}{|S|} \times \text{Entropy}(S_v)$ 
        entropyafter ← entropyafter + localentropy
    end for
    gain ← entropybefore - entropyafter
    return gain

```

IV. EXPERIMENTS

A. Data

To classify text documents, both training and test files are provided. Definition files are also provided.

1) *Definition Files:* The vocabulary.txt is a definition file that defines that set of all words that appear in the data set. It is a simple text file where each line represents a word, whose word ID is the line number in that file. The newsgroups.txt file is a definition file that defines the set of all news groups that appear in the data set. It is a simple text file where each line represents a news group, whose news group ID is the line number in that file.

2) *Data Files:* There are two data files: train.data, and test.data. The train.data file is used for training the model and the test.data file is used to evaluate the performance and accuracy of the model. Both files have the same format. These are text files where each line consists of a document ID, word ID, and word count. Each line represents the number of instances a particular word appears in a particular document. This means the word counts for each word in a document are distributed throughout multiple lines.

3) *Label Files:* There are two label files: train.label, and test.label. The train.label file is used for training the model and the test.data file is used to evaluate the performance accuracy of the model. Both files have the same format. These are text files where each line consists of a single integer representing a newsgroup ID. The line number represents the document ID. The file is a mapping between document IDs and news group IDs. For each line, the document is classified as a type of news group.

B. Methods

Naive Bayes classifiers are built by calculating the maximum likelihood estimate (MLE), and the maximum a priori (MAP). This calculation is described in the implementation section. This experiment builds several naive bayes classifier models with various settings: (1) various beta-value in the range [0.00001, 1.0], and (2) stop-word inclusion or exclusion. [Hmm include a sentence that explains what the beta-value is and what it's for?]

Ten models of various beta values are generated and their performance is compared against each other. Two models using the same beta value, but differing only in the inclusion or exclusion of stop-words, are also generated and compared against each other.

C. Results

1) *The effects of various beta-values:* Figure 1 shows performance of Naive Bayes Classifier over different values of beta. For small values of beta (0.00001), the accuracy is less than .785. As beta values increase, accuracy increases until reaching beta value of around 0.1. At beta value 0.1, accuracy peaks at 0.80. From this point, any increase in beta rapidly decreases accuracy. At beta value of 1.0, accuracy drops below 0.785.

This behavior may be explained by analyzing β 's rule in Equation 2, the MAP equation. Looking at the β -value in the MAP equation, we can see that for small counts of X_i , β becomes significant. For small counts of X_i , if we decrease β the numerator becomes very small, as seen in Figure 1. Similarly, when β approaches 1.0, β plays a larger

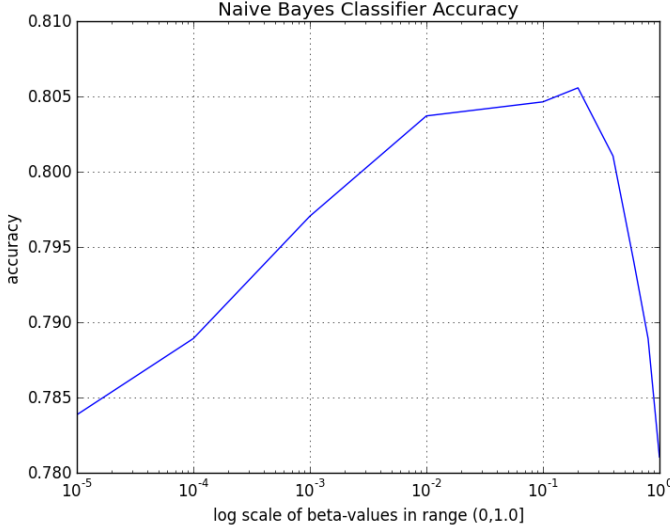


Fig. 1. Performance of classifier with different beta-values

role in the denominator. This is because as β increases, the denominator increases more rapidly than the numerator. When the denominator is much larger than the numerator, result is a very small number. As seen in Figure 1, when β approaches 1.0 accuracy drops significantly.

Two models with the same beta-value are also compared. One model diminishes the weight of English stop-words, and the other treats English stop-words the same as every other word in the vocabulary. The model that diminishes the weight of English stop-words produces an accuracy of 0.80 over the test set. The other model produces an accuracy of 0.78 over the test set. [So what? what is the significance of that?]

Table I is the confusion matrix produced for generated model with beta-value 0.1, excluding stop-words. The confusion matrix is a useful tool to help visualize what types of errors occur in the model. The confusion matrix's two axis are the predicted labels(columns), and actual labels(rows). The number in each cell[x][y] represents the number of times a prediction y was made when the true value was x. The main diagonal represents the correct prediction. All other values outside the main diagonal represent incorrect prediction.

The first two, and possible third, columns of this confusion matrix produce more errors than the other columns. Column one represents the prediction of the alt.atheism newsgroup, column two represents the prediction of the comp.graphics newsgroup, and column three represents the prediction of comp.os.ms-windows.misc.

V. DISCUSSION

Introduce discussion...

A. Comparing Model Options

Comparison...

B. Explanation of Results

Explanation...

C. Explain why it would be difficult to accurately estimate the parameters of this model on a reasonable set of documents.

The model given is:

$$(P(Y|X_1..X_d) \propto P(X_1..X_d|Y)*P(Y) = P(Y)*\prod(P(X_i|Y)) \quad (4)$$

It would be difficult to accurately estimate the parameters of this model on a reasonable set of documents because there are too many (how many?) parameters. For larger data sets estimating the parameters of this model becomes intractable. Answers go here.

D. Report your overall testing accuracy and print out the confusion matrix

The accuracy for our model with beta-value $\frac{1}{size(V)}$ is 0.785. This confusion matrix can be found in Table I in the Appendix. It is in the Appendix because the table is too large to fit in a two-column page.

E. Are there any newsgroups that the algorithm confuses more often than others? Why do you think this is?

asdf

F. Retrain your Naive Bayes classifier for values of beta between 0.00001 and 1 and report the accuracy over the test set for each value of B. Create a plot with values of beta on the x-axis and accuracy on the y-axis. Use a logarithmic scale for the x-axis. Explain in a few sentences why accuracy drops for both small and large values of beta.

The naive bayes classifier models with beta-values ranging from 0.00001 and 1.0 produce accuracies, over the test set, shown in Figure 1.

Figure 1 shows performance of Naive Bayes Classifier over different values of beta. For small values of beta (0.00001), the accuracy is less than .785. As beta values increase, accuracy increases until reaching beta value of around 0.1. At beta value 0.1, accuracy peaks at 0.80. From this point, any increase in beta rapidly decreases accuracy. At beta value of 1.0, accuracy drops below 0.785.

This behavior may be explained by analyzing β 's rule in Equation 2, the MAP equation. Looking at the β -value in the MAP equation, we can see that for small counts of X_i , β becomes significant. For small counts of X_i , if we decrease β the numerator becomes very small, as seen in Figure 1. Similarly, when β approaches 1.0, β plays a larger role in the denominator. This is because as β increases, the denominator increases more rapidly than the numerator. When the denominator is much larger than the numerator, result is a very small number. As seen in Figure 1, when β approaches 1.0 accuracy drops significantly.

G. Propose a method for ranking the words in the dataset based on how much the classifier 'relies on' them when performing its classification. It should give high scores to those words that appear frequently in one or a few of the newsgroups but not in other ones. Words that are used frequently in general English should have lower scores, as well as words that only appear extremely rarely throughout the whole dataset.

Answers go here.

H. Implement your method, set beta back to $1/V$ and print out the 100 words with the highest measure.

Answers go here.

I. If the points in the training dataset were not sampled independently at random from the same distribution of data we plan to classify in the future, we might call that training set biased. Dataset bias is a problem because the performance of a classifier on a biased dataset will not accurately reflect its future performance in the real world. Look again at the words your classifier is 'relying on'. Do you see any signs of dataset bias?

Answers go here.

VI. CONCLUSIONS

Hello, conclusions.

REFERENCES

- [1] T. M. Mitchell *et al.*, *Machine learning*. wcb, 1997.

APPENDIX

A confusion matrix is more descriptive of a model's performance than accuracy. A confusion matrix shows the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). From this accuracy and misclassification can be calculated. Accuracy is $(TP+TN)/(p+n)$ and Misclassification is $1.0 - \text{Accuracy}$.

TABLE I
CONFUSION MATRIX

Here are some of the more significant words and their word IDs.

TABLE II
SELECTED WORD ID AND WORD MAPPING

Word	
1	a
2	an
3	the