MiniProject 2: Classification of Textual Data

David Schrier 260827890

Theo Janson 260868223

Jacob McConnell 260706620

Abstract

Computer Algorithms have evolved over many decades of research and development. Today, supervised machine learning algorithms are used to make predictions on new data using previous labelled data. These techniques are revolutionizing many fields including retail, advertisement, healthcare, engineering, etc. Machine learning techniques in Natural Language Processing can be used to classify topics and sentiments from textual data. This assignment explores the use of Logistic Regression and Multinomial Naive Bayes for classification on two text-based datasets. Our results suggest that Logistic Regression performs moderately better than Naive Bayes for binary classification problems, while Multinomial Naive Bayes performs moderately better than Logistic Regression on multi-class classification.

Keywords: Naive Bayes, Logistic Regression, K-Fold Cross-Validation, Gradient Descent

1. Introduction

Text classification has many applications in the commercial field. Machine Learning models allow for grouping large texts into different categories and topics, or even sentiments and emotions. For instance, a text classifier can be used detect and filter spam emails, or sentiment analysis can be used to label product reviews or social media posts as either positive or negative, or even classify these texts by emotions such as anger and empathy. As a product can have thousands of reviews, conclusions cannot practically be drawn by hand selecting, and reading a few reviews. Text-based machine learning techniques allow for huge amounts of text to be reviewed and classified statistically. The task of this project is to apply Naive Bayes and Logistic Regression to two specific datasets to classify one dataset by sentiment and a second dataset into categories. The datasets used in this report are 20 News Groups from Sci-kit Learn and IMDB movie reviews from Stanford University. Results from other papers have shown that Logistic Regression performed much better compared to Multinomial Naive Bayes on the two datasets [1].

2. Model Descriptions

This section will detail both machine learning methods and their background.

2.1 Logistic Regression

Logistic Regression model maps a logistic function to a dataset. While typically used for binary text classification, this machine learning model can handle multi-class labelling as well. $-w^\top x$ represent the best fit weights on the particular dataset. The regression model is defined as:

$$f_w(x) = \frac{1}{1 + e^{-w^\top x}} \tag{1}$$

The logistic regression model can be fitting by solving for the weights using gradient descent techniques. We used the Limited-memory-Broyden–Fletcher–Goldfarb–Shanno Algorithm, a gradient descent technique which approximates the Hessian Matrix, and performs better on smaller datasets. The documentation indicated that this solver produces better performance on multiclass classification. We also used Stochastic Average Gradient, which performs better on larger datasets. We varied regularization, and obtained the best regularization constant at the value for which the average accuracy on the validation sets was highest. Finally, we also varied the maximum number of iterations of gradient descent.

2.2 Naive Bayes

The Naive Bayes model uses a prior probability distribution for each class and a likelihood to give a posterior probability for that given class c. Naive Bayes works by applying Bayes Theorem and it assumes that there is independence between the features. Even though this might not be true the resulting classifier can still work reasonably well. Depending on the kind of classification you are doing your Naive Bayes classifier has to be designed with the kind of probability distribution you expect your features to have. For example because our features were counting the number times each word showed up in each entry we used a multinomial probability distribution. This posterior probability is defined by Bayes' Rule:

$$p(y = c|x) = \frac{p(c)p(x|c)}{p(x)} \tag{2}$$

When classifying new text there is the possibility that it would contain a word which had never been recorded in its correct class. Even though other words would indicate its correct class a probability of zero would dominate so Bayes' rule would give a zero posterior probability, given that the conditional probability of the correct class given the observed feature is initialized to zero. To circumvent this issue we apply Laplace smoothing such that it adds a pseudo-count to a word so it is initialized to a non-zero prior probability, regardless of whether or not the word class pair appears in the training set. This psuedo-count is the hyper-parameter that we vary in this project.

3. Datasets

The first step to processing a piece of text for classification by a machine learning is to perform basic preprocessing to reduce the number of features. We remove 'stop words' from the data. Stop words, such as (at, is, a, etc.) don't add value for classification as they are contained in most text examples, regardless of the text's label. We also converted all text to lower case and removed accents as 'uppercased' words have the same meaning as their 'lowercased' counterparts. The second step to processing a piece of text for classification by a machine learning is to tokenise it into words, or grams. We used an n-grams model, and tested unigrams and bigrams. Unigrams tokenise a text by separating each word. The result is essentially a bag of words. Bigrams tokenise a text using sequences of two consecutive words in order to maintain some element of structure to the data, resulting in double the number of features. The structure can have value for classification. The second step for processing is to vectorize the text data. Naturally, machine learning models are trained

on numerical features and not text ones. Vectorizing removes all linguistic value from the words, replacing each word with a number. We used a count vectorizer and a frequency vectorizer TF-IDF. Both vectorizers transforms a corpus of text (the training set) into a numerical vector. Count vectorizer transforms a test word into the number of times it appears in the text. Similarly, TF-IDF vectorizing transforms a test word into a number obtained from the function below:

TF-IDF = (Number of times word appears in text) / (Total number of words in text) x log_{10}(Number of texts / Number of texts containing words).

3.1 20 News Groups Dataset

The 20 newsgroups training set comprises around 11314 newsgroups posts on 20 topics obtained from Scikit-Learn. Naturally, there are 20 distinct labels. Usually, we use a test set size which is 20% that of the training set. To speed up calculations, we capped the test set size to 1000 and the validation set size to 500. We performed a test using the complete test set and complete validation sets to make sure that the results are similar. The dataset is fairly balanced - 600 examples in the most common category, and 377 examples in the least common category.

3.2 Movie Reviews Dataset

The IMDB Reviews dataset is obtained from Stanford University. Its training and test sets are comprised of 25000 examples each. The labels are a binary sentiment score. We also capped the test set size to 1000 and the validation set size to 500.

4. Results

We performed 5-Fold Cross Validation on both datasets on a series of hyperparameters to obtain the hyperparameter which produces the highest accuracy. We also tested using unigrams versus bigrams and count vectorizer versus frequency vectorizer. We assessed the datasets' and the models' performances using accuracy, precision, recall, F1, and time taken to train and test the model. Accuracy was used to tune hyperparameters. Test descriptions and results can be found in the results appendix. The tests with the highest performance are listed in the tables below.

4.1 Logistic Regression

Tests were ran to determine the best solver, regularization constant and max number of iterations.

Dataset	Vectorizer	Solver	N-Grams	Regularization	Max Iteration	Accuracy
Movie Review	CV	SAGA	1	65	450	87.00
Movie Review	CV	SAGA	2	65	350	85.80
Movie Review	FV	SAGA	1	65	350	85.45
20 News Set	CV	LBFS	1	0.27	100	63.70

4.2 Naive Bayes

Dataset	Vectorizer	N-Grams	Alpha	Accuracy
Movie Review	CV	2	3.549	83.46
Movie Review	CV	1	3.549	85.72
Movie Review	FV	1	3.549	83.40
20 News Set	CV	1	0.01	70.80

To determine the ideal multinomial distribution for our posterior probability, we varied the optimal Laplace smoothing value α, and obtained the best constant producing the highest average accuracy on the validation sets.

4.3 Varying Training Set Size

For both models, the hyperparameters and solvers that produced the highest accuracy were obtained, and the models were trained on datasets of varying sizes (by proportion of available data). The test set sizes were held constant.

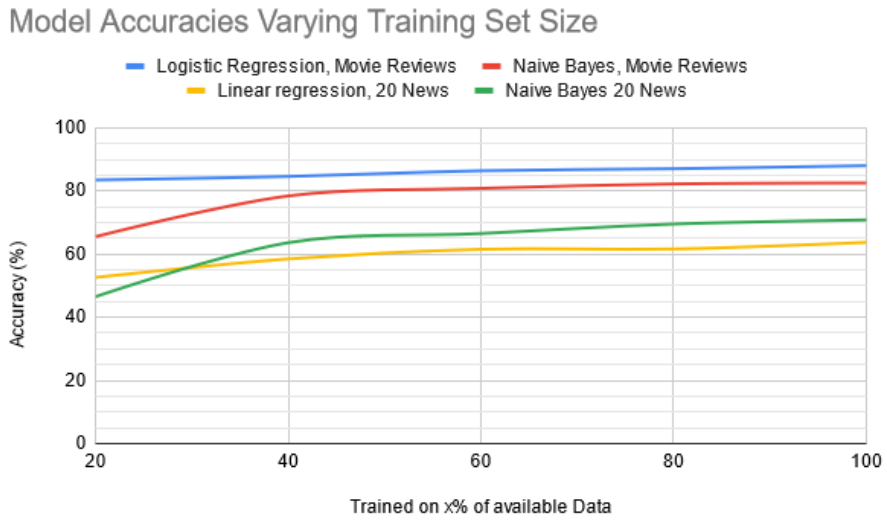

Figure 1: Model Accuracies for Both Models and Datasets

5. Discussion and Conclusion

Both logistic regression and Naive Bayes showed similar accuracy, recall and precision performance for data vectorized on unigrams and on bigrams of both datasets. This result suggests maintaining structure in the data does not improve predictive performance for basic classification. Moreover, unigrams are preferred to bigram as the require half the features, reducing computational costs. Time taken to train and test was obtained for these tests. Generally, using unigrams resulted in a lower training time, but we cannot derive meaningful conclusions from the results as the tests were done on a hosted runtime on Google Colab, which has variable run times.

Similarly, the count vectorizer and the TF-IDF vectorizer produced similar performance. This might indicate that the vocabulary in the corpus is 'relatively' small, as many of the same words are used in its composing texts. If TF-IDF were to perform better, it would indicate a large vocabulary, and words that occur more rarely would have a greater value for classification. If the size of each entry is roughly held constant than the count is roughly proportional to the frequency. In that case you wouldn't gain much predictive power by looking at frequency over counts. As the multiclass 20 News dataset had much higher fitting and predicting times than the binary movie reviews case, no tests were done using bigrams and the TD-IDF vectorizer.

On the movie reviews dataset, the stochastic average gradient performed similarly to Broyden–Fletcher–Goldfarb–Shanno Algorithm, but was considerably faster. A regularization constant of 65 and a maximum of 350-450 gradient iterations also produced the highest average accuracy on the validation sets. The later increased fitting and prediction time. Max gradient iterations above 200 did not significantly increase the accuracy. The 20 News dataset showed worse performance than the binary dataset. Accuracies around 30% were obtained using stochastic average gradient solver. Gradient Decent Algorithms might not converge to a global minimum for Logistic Regression algorithms because it is multi-class, as compared to binary classification where Gradient Descent is performed on a cost function that is convex and has only one global minimum, so this may be the explanation for the performance. Conversely, using the LBFGS yielded accuracies around 70% for regularization values on the order of 0.1.

The Laplace smoothing factor was varied when testing Naive Bayes. Tests using unigrams and count vectorization also produced similar results to those using bigrams and TF-IDF vectorizer. Naive Bayes had strong average performance on the movie reviews dataset - around 85% – performing slightly worse than logistic regression. Conversely, Naive Bayes performed better, on average, than logistic regression for the multiclass dataset. This may be because the optimized parameters for the weights for Logistic Regression for Gradient Descent are bottle-necked by the text data being multi-classed. Gradient Descent might find a local minimum but not the global minimum for a loss function that is multi-classed, so it is likely that Naive Bayes performed better for multi-class labels. Moreover, the ideal smoothing factor was greater for multiclass Naive Bayes classification than for binary classification. We suppose this is because the count feature word associated with each class are lower as they spread out over many classes.

Finally, the training set size was varied relative to the number of available training examples. Naive Bayes' performance depends considerably on the training set size. When fitted using less than 40% of available data, the performance dropped by roughly 20% for both datasets. The logistic regression performance on the the multiclass news dataset also dropped by roughly 20% when the model is trained using less than 30% of available data. Conversely, logistic regression performance on the binary movie reviews dataset is minimally affected by training set size. Binary classification regardless of the training set size should always form a similar convex loss function for Gradient Descent.

In conclusion, Logistic Regression performed better than Naive Bayes on binary sentiment analysis, while Multinomial Naive Bayes performed better for 20-class classification. As mentioned before, Logistic Regression's loss function forms convex shape such that gradient descent will find a global minimum for binary classification, so when the loss function maps to complex convex shape it might not find a global minimum for multi-class. This aligns well with the literature.

6. Statement of Contribution

The distribution of work is as follows: Theo Janson wrote all the classes other than Naive Bayes', helped debug Naive Bayes, ran tests on the models, and contributed to the report. David Schrier worked on the beginning of the Naive Bayes and contributed to the report in Overleaf. Jacob McConnell implemented Naive Bayes, ran tests on the models and contributed to the report.

References

[1] Jaiswal, Manish, et al. "Prediction and Analysis of Movie Success with Machine Learning Approach."