CSPC 204/224

MACHINE LEARNING PROJECT

Sentiment Prediction expressed in IMDB reviews

Table of Contents:

Title

Sentiment Prediction expressed in IMDB reviews

Abstract

Sentiment Analysis or opinion mining is one of the most sought after topics of both Machine Learning and Deep Learning. Sentiment refers to the sequence of words and is usually associated with an opinion or emotion. Analysis refers to the process of looking at the data and making inferences; in this case using Machine Learning to predict whether a movie review is positive or negative. In this study, I develop a classifier that will take as input a user generated movie review and automatically classify that review as positive or negative. To this end I experimented with a wide variety of models including Naïve Bayes, Logistic Regression and Artificial Neural Network.

Keywords

Natural language Processing (NLP), Bag of Words Pipeline, Naïve Bayes, Sentiment Analysis, Binary Classification, TF-IDF, Logistic Regression, Artificial Neural Network

Introduction and Motivation

Ever since Data Mining and Machine Learning techniques were popularized, people applied the use on a wide range of topics ranging from classification of insect sounds to recognition of pen strokes on a handwritten assignment of students. In the area of Natural Language Processing while some use it to perform tasks such as information extraction; some use it for the classification of sentiment orientation.

Sentiment is an attitude, thought, or judgment prompted by feeling. Sentiment analysis, which is also known as opinion mining, studies people’s sentiments towards certain entities. Internet is a resourceful place with respect to sentiment information. From a user’s perspective, people are able to post their own content through various social media, such as forums, micro-blogs, or online social networking sites.

However, those types of online data have several flaws that potentially hinder the process of sentiment analysis. The first flaw is that since people can freely post their own content, the quality of their opinions cannot be guaranteed. For example, instead of sharing topic-related opinions, online spammers post spam on forums. Some spams are meaningless at all, while others have irrelevant opinions also known as fake opinions. The second flaw is that ground truth of such online data is not always available. A ground truth is more like a tag of a certain opinion, indicating whether the opinion is positive, negative, or neutral.

Previous Works

The original work on this dataset was done by researchers at Stanford University wherein they used unsupervised learning to cluster the words with close semantics and created word vectors. They ran various classification models on these word vectors to understand the polarity of the reviews. This approach is particularly useful in cases when the data has rich sentiment content and is prone to subjectivity in the semantic affinity of the words and their intended meanings.

Apart from the above, a lot of work has been done by Bo Pang and Peter Turnkey towards polarity detection of movie reviews and product reviews. They have also worked on creating a multi-class classification of the review and predicting the reviewer rating of the movie/product.

These works discussed the use of Random Forest classifier and SVMs for the classification of reviews and also on the use of various feature extraction techniques. One major point to be noted in these papers was exclusion of a neutral category in classification under the assumption that neutral texts lie close to the boundary of the binary classifiers and are disproportionately hard to classify.

Project Scope and Contributions

Sentiment Analysis is one of the most important topics in Machine Learning used to analyze the behaviour and emotions of the users about a certain topic or issue. This can be easily used to understand the response of the users about anything ranging from a political issue to even a new product released in the market. I have applied different models to do so on Movie Review Prediction System. This can be used further to understand the emotions and reviews of any new movie released in the future as the vocabulary created in the training set is most optimal to the use of Movie Review Sentiment Analysis.

There are many sentiment analysis tools and software existing today that are available for free or under commercial license. With the advent of micro blogging, sentiment analysis is being widely used to analyze the general public sentiments and draw inferences out of these. One famous application was use of Twitter to understand the political sentiment of the people in context of German Federal elections.

Dataset and Features

The dataset used for this task was collected from Large Movie Review Dataset which was used by the AI department of Stanford University for the associated publication. The dataset contains 50,000 training examples collected from IMDB where each review is labeled with the rating of the movie on scale of 1-10. As sentiments are usually bipolar like like/dislike, I categorized these ratings as either 1 (like) or 0 (dislike) based on the ratings. If the rating was above 5, I deduced that the person liked the movie otherwise he did not.

For preprocessing, I first created a copy of the dataset, but in lowercase. I then created several variations of the lowercase dataset using different combinations of preprocessing methods. The four preprocessing methods I used were removing punctuation, tokenization, stemming, and removing stop words.

1. The punctuation removal step was a straightforward step; removing all punctuation characters from the strings.
2. Tokenization used a regex expression to split the string by non-alphanumeric characters (the difference from removing punctuation here is mainly in contractions- removing punctuation changes "what's" to "whats", whereas tokenization changes "what's" to "what s").
3. For stemming I used the popular snowball stemmer from NLTK. This transformed words into their stems- i.e. "fishing" becomes "fish".
4. For stop words removal I similarly utilized NLTK's provided corpus to remove common English 'filler' words like "and", "an", and "the" from the dataset.

These methods were used in various combinations to create the dataset variations. Specifically, I tried using all four, excluding one at a time (i.e. Punctuation, Stemming, Stop word removal), just tokenization and stop word removal, and tokenization and stemming (6 variants in total).

All 6 of these dataset variants were then converted into a word count vector for use by the Naive Bayes algorithm, and a tf-idf (a statistic combining term frequency and inverse document frequency) vector for Naive Bayes, Logistic Regression and ANN. Lastly, for every model that I ran with every dataset, I split this dataset 60/20/20, with 60% becoming the training set, 20% becoming the validation set, and the last 20% becoming the test set.

Model Architecture/Design

1. Naive Bayes

Naive Bayes is a generative learning model that takes the distribution of words given labels from the training set to compute the probability of a test set statement having a positive or negative label. The distribution from the training set is usually modified somewhat using Laplace Smoothing, which deals with unseen vocabulary in the training set. The exact equations for calculating the distributions are as follows:

$$\phi_{j|y=1} = \frac{1 + \sum_{i=1}^{n} 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\}}{2 + \sum_{i=1}^{n} 1\{y^{(i)} = 1\}}$$

$$\phi_{j|y=0} = \frac{1 + \sum_{i=1}^{n} 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\}}{2 + \sum_{i=1}^{n} 1\{y^{(i)} = 0\}}$$

$$\phi_y = \frac{\sum_{i=1}^{n} 1\{y^{(i)} = 1\}}{n}$$

I also read that a Naive Bayes could theoretically run with fractional counts, so I also ran a variant using tf-idf scores instead of word counts.

2. *Logistic Regression*

Logistic Regression is an algorithm that learns a set of weights, theta, which gets multiplied with the features and fed into the sigmoid function in order to get the hypothesis:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

The output of the sigmoid is then used to make a prediction of the label, and the goal of logistic regression is to find the weight that minimizes the log-loss.

$$\sum_{i=1}^{n} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))$$

To find the right set of weights, I run gradient descent on the training set:

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$$

3. *Artificial Neural Networks*

Artificial Neural Networks is an algorithm that learns a set of weights and bias for each layer l to get the hypothesis as:

$$z^{[\ell]} = W^{[\ell]} a^{[\ell-1]} + b^{[\ell]}$$

$$a^{[\ell]} = g^{[\ell]}(z^{[\ell]})$$

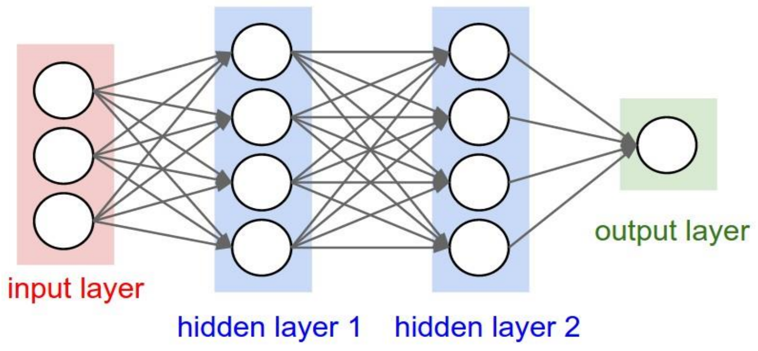
For the training part backpropagation algorithm is used as:

$$\delta^{[\ell]} = (W^{[\ell+1]\top} \delta^{[\ell+1]}]) \circ g'(z^{[\ell]})$$

$$\nabla_{W^{[\ell]}} J(W, b) = \delta^{[\ell]} a^{[\ell-1]\top}$$

$$\nabla_{b^{[\ell]}} J(W, b) = \delta^{[\ell]}$$

In this model, I have trained a neural network having two hidden layers with neurons in input layers= length of dictionary, neurons in first hidden layers=16, neurons in second hidden layer=16 and neurons in output layer=1. I used ReLU as activation function in both the hidden layers and Sigmoid function as the activation function of the final output layer.



Implementation and Results

1. Naive Bayes

One of the first models I experimented with was the normal Naive Bayes with word counts and Laplace smoothing. I ran the algorithm with all six preprocessed dataset variations, and found pretty minimal differences in performance, with F1 scores varying from 0.836 to 0.864, and accuracies varying from 0.841 to 0.866. The best performing dataset here (based on F1 score) was the dataset that did not have stemming performed, with an F1 score of 0.864 and an accuracy of 0.866.

I also tried a variant of Naive Bayes with Tf-idf scores instead of word counts. Again I ran with all six preprocessed dataset variations, and again I found minimal performance differences, and while accuracies improved marginally to a range of 0.851 to 0.874, the F1 scores belonged to a range of 0.847 to 0.872. The (marginally) best performing dataset here was the one where I did not run the stemming step, with an accuracy of 0.874 and an F1 score of 0.872.

2. Logistic Regression

Similarly to Naive Bayes, the first thing I did was run Logistic Regression with all our different preprocessed datasets. I utilized scikit-learn’s implementation, leaving most hyper parameters at a default, but I varied the regularization factor, C, using the values 0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 100. This was chosen based on the advice of tuning with values increasing approximately by a factor of 3. Surprisingly, across all datasets, the best regularization factor based on the cross validation set turned out to be 3, and using this regularization and the tokenized and stemmed dataset, I were able to achieve an F1 score between 0.892 and 0.902 and accuracy between 0.891 and 0.901. The (marginally) best performing dataset here was the one where I did not run the stemming step, with an accuracy of 0.901 and an F1 score of 0.902. The worst performing dataset was the one where only tokenization and stemming was used without using any punctuation and stopword removal with an accuracy of 0.891 and F1 score of 0.892.

3. Artificial Neural Network

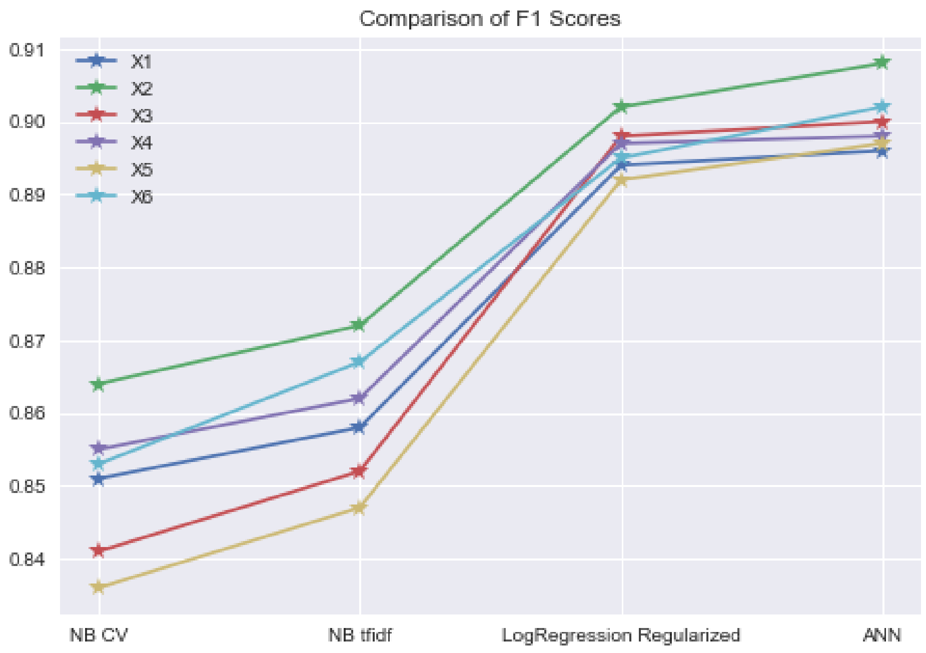
To implement Artificial Neural Network, I used Dense Layer implementation of keras with activation as ReLU in both the hidden layers and sigmoid activation in the final output layer. I first trained the model till 10 epochs with a batch size of 512 and analyzed the epoch up to which the validation accuracy was increasing. Then I re-initialized and re-trained the model from beginning but now only up to the epoch till which validation accuracy was not decreasing. This step is crucial to avoid Overfitting Issues of ANN.

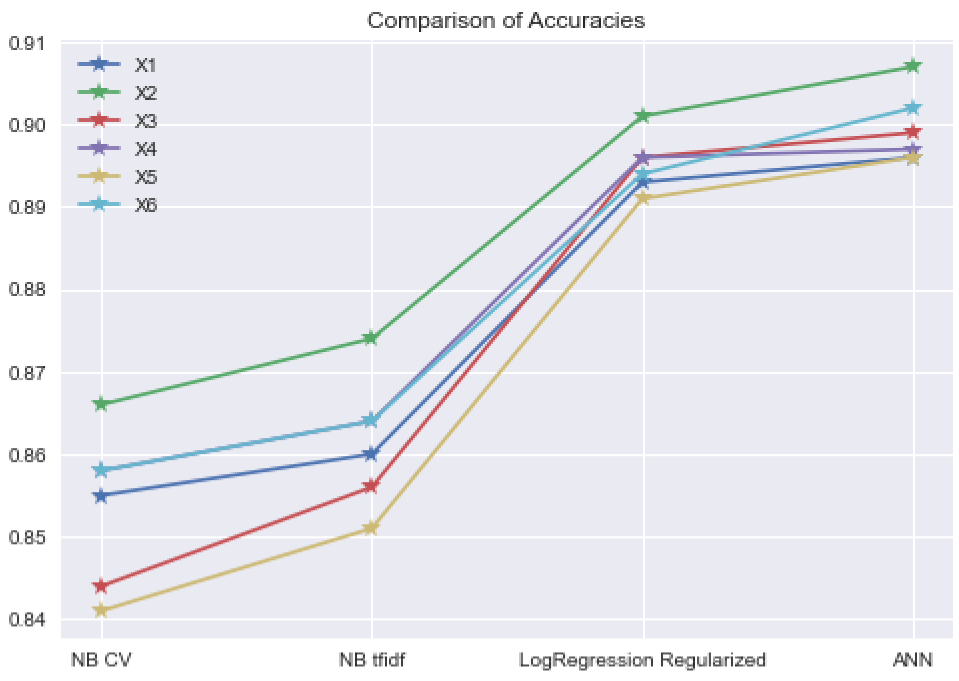
I ran the algorithm with all six preprocessed dataset variations, and found pretty minimal differences in performance, with F1 scores varying from 0.896 to 0.908, and accuracies varying from 0.896 to 0.907. The best performing dataset here (based on F1 score) was the dataset that did not have stemming performed, with an F1 score of 0.896 and an accuracy of 0.907. The worst performing dataset was the one where all the four steps i.e. tokenization, stemming, punctuation and stopword removal was used with an accuracy of 0.891 and F1 score of 0.892.

Comparison

	Accuracy	F1	Precision	Recall	TN	FP	FN	TP
Naive Bayes (w/ tf-idf)	0.874	0.872	0.891	0.854	4431	528	737	4304
Naive Bayes (w word counts)	0.866	0.864	0.880	0.848	4378	581	764	4227
Logistic with regularization	0.901	0.902	0.895	0.909	4424	535	457	4584
ANN	0.907	0.908	0.906	0.909	4486	473	458	4583

To draw the comparison graphs among different preprocessed datasets for different algorithms used let us define the pipeline of all the datasets used.
X1 is prepared through (Punctuation, Tokenization, Stemming and Stop word Removal)
X2 is prepared through (Punctuation, Tokenization and Stop word Removal)
X3 is prepared through (Punctuation, Tokenization, and Stemming)
X4 is prepared through (Tokenization, Stemming and Stop word Removal)
X5 is prepared through (Tokenization and Stemming)
X6 is prepared through (Tokenization and Stop word Removal)





Conclusion and Future Work

From the results above, I can infer that for our problem statement Artificial Neural Networks performed much better as compared to other models both in terms of F1 Scores and Accuracy on Test Data for all the six different preprocessed datasets. Even Logistic Regression with Regularization works better than Naïve Bayes Algorithm. It is generally seen that Logistic Regression<Naïve Bayes < Logistic Regression with Regularization. Also, in case of Naïve Bayes Algorithm, tf-idf vectorization works better than count vectorization on our dataset.

If I were to continue this project in the future, there are a variety of things I would like to try. On the preprocessing side of techniques, I would like to try using other embedding schemes than tf-idf like Word2Vec and perhaps some Text Normalization. I would like to apply RNNs on this dataset and check their performance. Also provided more time and computing resources, I would like to apply Google's pre-trained BERT model on this dataset.

I would like to extend the classification scheme from 'positive' and 'negative' to 'positive' , 'neutral' and 'negative' as well.

References

- Chu, Chun-Han & Wang, Chen-Ann & Chang, Yung-Chun & Wu, Ying-Wei & Hsieh, Yu-Lun & Hsu, Wen-Lian. (2016). Sentiment Analysis on Chinese Movie Review with Distributed Keyword Vector Representation. 10.1109/TAAI.2016.7880169.
- Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). "Learning Word Vectors for Sentiment Analysis." *The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011)*.
- Maas AL, Daly RE, Peter PT, Huang D, Ng YA, Potts Ch (2011) Learning word vectors for sentiment analysis. In: Proceeding of 49th ACL, pp 142–150

- IMDB Dataset Link : http://ai.stanford.edu/~amaas/data/sentiment/
- NLTK Stopwords Corpus: http://www.nltk.org/book/ch02.html
- Punctuation vector: https://www.kaggle.com/c/quora-insincere-questions-classification/discussion/74518
- Sentiment Analysis- Wikipedia: https://en.wikipedia.org/wiki/Sentiment_analysis