CIS520 Machine Learning

Sentiment Analysis of Tweets

The SentiMentalists

Sai Krishnan Chandrasekhar	Sakthivel Sivaraman	Vivek Venkatram
saikrish@seas.upenn.edu	sakt@seas.upenn.edu	vivekv@seas.upenn.edu

Project Overview

The goal of the project is to apply machine learning algorithms to analyze the sentiment of tweets, as joy or sad, given tweets and their associated images. The data consists of 4500 labelled and unlabelled tweets. The components of the data given are raw tweets, word counts of the top 10000 words occurring in all tweets, raw images, features corresponding to the last layer of a Convolutional Neural Network, object/scene probabilities and color features.

Methodology

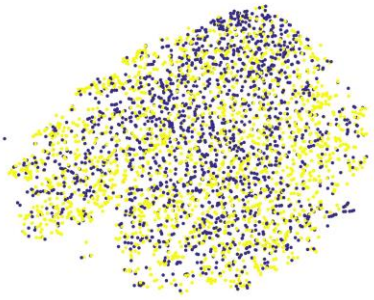
Preprocessing

The goal of preprocessing was to format the data into a form from which we could obtain meaningful features. To do this, we separated out words, words with hashtags, emoticons and Unicode characters. The methods that were tried out are Word2vec[3] and obtaining scores for words using lists like SentiWordNet[4] and AFINN (Finn Årup Nielsen)[5]. We also used our own way of assigning positive and negative scores to the words based on the data given. Based on these scores, the neutral words/features were eliminated. This method didn't give much boost as it was very data specific.

Feature Selection and Dimensionality Reduction

NLP Tools:

- **Word2vec:** Word2vec represents the given words in a continuous vector space where semantically similar words are grouped together. We used a matlab package to represent the words in the tweets as vectors. Though the vector representation gives more meaning to the tweets, it didn't work quite well in this case due to the following reasons:



Data Visualization

1. Word2Vec generates a 300 dimensional vector for each word, which can be then normalized and averaged to obtain the features of a tweet. Averaging the word vectors leads to loss of information.
2. The averaged word vectors, for each tweet, when visualized using t-Distributed Stochastic Neighbor Embedding (t-SNE)[1], we can see (as shown in the fig), that the happy tweets and the sad tweets aren't seperated well enough in the feature space. It gave around an **average accuracy of 76%** which was much lower compared to the accuracies on word counts.

- **Sentiwordnet:** Sentiwordnet is a lexical resource for opinion mining. This was used to obtain three scores for a given word: positive, negative and objective. This method was used to predict the overall sentiment of a tweet as a combination of the individual sentiments of each word based on their POS tag. This method was not successful due to following reasons:

1. The structure of the data we had. Most tweets had words removed hence POS tagging did not work. Many words had multiple scores based on their POS. The correct scores could not be obtained since their POS was not known to us.
2. No scores could be obtained for non-dictionary words.
3. The overall sentiment of the tweet could not be judged accurately based on the scores of each word on its own.
4. Most words did not have a positive, negative or objective score.

- **Sentiment Score Assignment based on Data:** We also used our own way of assigning positive and negative scores to the words based on the data given. Based on these scores, the neutral words/features were eliminated. This method didn't give much boost as it was very data specific.

Top Negative Words

Top Positive Words

- **Stepwise Regression:**
  Stepwise regression was used as a method to determine the coefficient estimate values of each feature. MATLAB's stepwisefit() was used to achieve this. This method was used mainly during ensembling to assign weights to each of the methods.

Algorithms

- **Generative:**
  - **Naive Bayes: Test Accuracy: 80.06%**
    The Naive Bayes classifier is based on the Bayes' theorem with independence assumptions between predictors and is dependent on the bag of words representation of the data [11]. The model assumed that the position of the words do not matter. As a result, Naive Bayes could consider only unigrams and not bigrams. But on the whole, it proved to be a dependable classifier that could be further used in ensembling.
  - **K-means: Test Accuracy: 68.34%**
    K-means was used to divide the data into different clusters based on euclidean distances between two points. However, the algorithm did not provide with a higher accuracy due to the high dimensionality of the data and sparsity of the features. It became difficult to understand and interpret the clusters as well as determine the right number of clusters since K-means includes all the features and gave intricate clustering models.

- **Discriminative -**
  - **Logistic Regression: Test Accuracy: 80.69%**
    Logistic Regression was used to extract a set of weighted features from the data, combine them up linearly using their weights and applying the logistic function to this combination. An L2-regularized Logistic Regression gave the highest accuracy among all the methods used since it could appropriately weigh the effect of each of the features on the outcome variable i.e. features that could contribute more to a certain outcome (joy or sad) are weighted higher than features that do not.

- **Support Vector Machines: Test Accuracy: 80.4%**
  Usually, SVM and Logistic Regression give comparable results but Logistic Regression performed slightly better than SVM for the given data. Although SVMs are efficient in high dimensional spaces [8], it might failed to give an accuracy as good as Logistic Regression as SVMs only consider points near the margin i.e the support vectors. It doesn't consider points that are away from the separating hyperplane.

We observed that the accuracy of the SVM was comparable to that of Naive Bayes and hence we used it as a classifier for ensembling.

Instance-Based

- **K Nearest Neighbours: Test Accuracy: 62.6667% for 10 Nearest Neighbors**
  KNN performed the worst amongst all the methods we tried. This might be so because the nearest neighbors ended up being quite far owing to the high dimensionality of the data. It was also computationally intensive to find the nearest neighbors due to the same. We observed that the accuracy increased when the neighbors increased from 4 to 10 but decreased thereafter.

Regularization

The confusion matrix of each algorithm was used to determine the accuracy with which each algorithm predicted positive labels and negative labels. These were called positive weights and negative weights respectively. While ensembling the different methods, we multiplied the predicted label of each label with its corresponding positive or negative weight. This was, in effect, the probability of that method predicting that observation to be positive or negative. The graphs are shown in the ensembling methods section.

We also experimented with assigning zero weights to observations which had a "neutral" score to it (using the positive and negative scores obtained from sentiwordnet). This didn't improve the accuracy significantly.

Semi-supervised Learning:

- **Eigenwords (PCAed Logistic): Test Accuracy - 78.9%**
  SVD was used to reduce the dimensionality of the data. The left singular vectors (word embeddings) were used to predict the labels. A logistic regression was run on the reduced data. Through cross validation, it was determined that 232 left singular vectors gave the best accuracy for our data.
- **Semi-Supervised Support Vector Machine : Test Accuracy - 79.77%**
  Cost sensitive semi-supervised support vector machine (cs4vm) package [2] was used to label the validation set. This algorithm is very useful when there are at least half the data labelled and different misclassification errors associated with unequal costs. This cost was tuned with the labeled data. This returned labels for our unlabeled data. We trained a logistic regression on the entire dataset (9000 observations) and used this on the leaderboard.

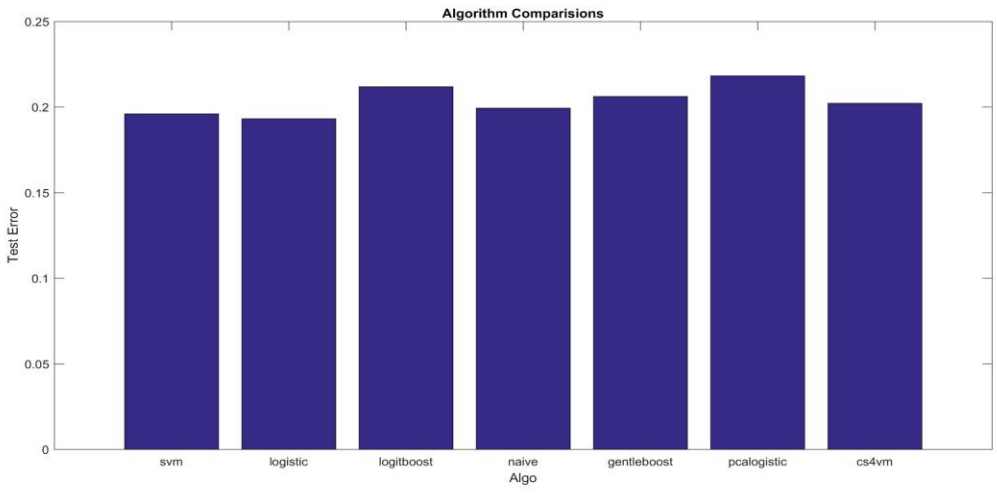
- **Ensemble Methods as part of core models:**

  From the below mentioned methods, we used **Logitboost and GentleBoost as part of our core models.**

  - **Adaboost[7]: Test Accuracy: 77.45%, 450 stumps:**
    We experimented with AdaboostM1 method. The algorithm trains learners sequentially. For every learner with index *t*, computes the weighted classification error. AdaBoostM1 then increases weights for observations misclassified by learner *t* and reduces weights for observations correctly classified by learner *t*. The next learner *t* + 1 is then trained on the data with updated weights. We discarded this model because it didn't give a great accuracy.
  - **LogitBoost[10]: Test Accuracy: 78.8%:**
    LogitBoost works similarly to AdaBoost, except it minimizes binomial deviance. Binomial deviance assigns less weight to badly misclassified observations (observations with large negative values of ynf(xn)). LogitBoost can give better average accuracy than AdaBoost for data with poorly separable classes.
  - **GentleBoost[9]: Test Accuracy: 79.37%:**
    Gentle AdaBoost combines features of AdaBoost and LogitBoost. GentleBoost minimizes the exponential loss. Like LogitBoost, every weak learner fits a regression model to response values yn ∈ {–1,+1}. This makes GentleBoost another good candidate for binary classification of data with multilevel categorical predictors.
  - **Random Forest[6]: Test Accuracy: 79%, 1800 trees, max number of splits, 200**
    We experimented with Bootstrap-aggregated (*bagged*) decision trees. These combine the results of many decision trees, which reduces the effects of overfitting and improves generalization. We discarded this method despite it's good accuracy because of its prediction time. It took close to 13 minutes to predict.

TEST ACCURACIES

SVM	Logistic	Logitboost	Gentleboost	PCAed Logistic	Cs4VM	Naïve Bayes
80.4	80.69	78.8	79.37	78.9	79.77	80.06

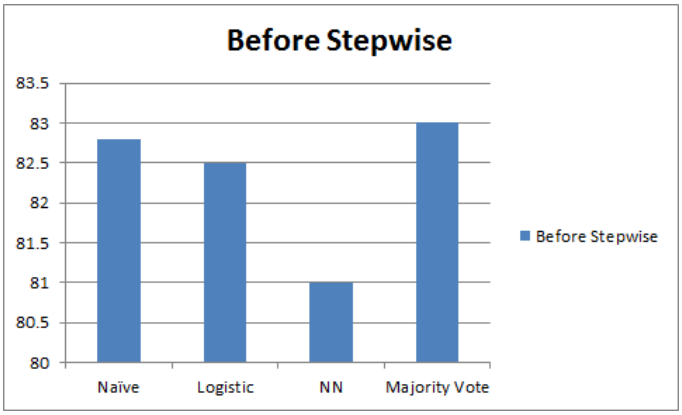


- # Ensemble Methods:

We used three methods to ensemble our 7 core models. We combined the predicted labels of our 7 core models into a matrix of 7 columns and ran different ensemble methods on them. We also used stepwise regression to determine weights of each model and whether or not to consider the labels obtained from a model:

Before Stepwise:

- **Naive Bayes: Test Accuracy: 82.8%**
  We used naive bayes with a "Normal Distribution" as one of our ensembling models.
- **Logistic Regression: Test Accuracy: 82.5%**
  We used L2 regularized Logistic regression as one of our ensembling models.
- **Neural Networks: Test Accuracy: 81%**
  We used a well-tuned neural network of 2 layers consisting of 7 neurons in the first layer and 8 neurons in the second.
- **Majority Vote: Test Accuracy: 83%**

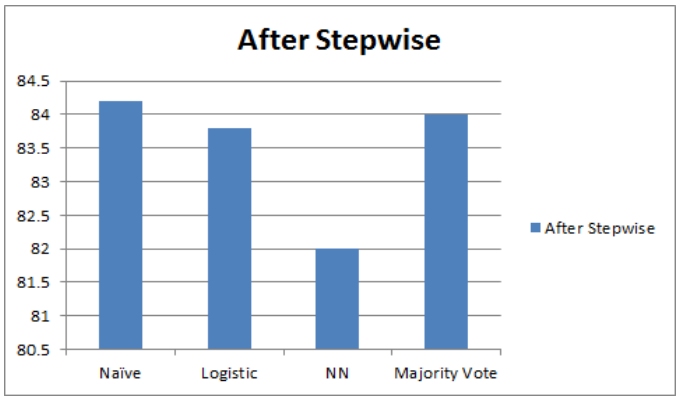


Stepwise Regression Coefficients

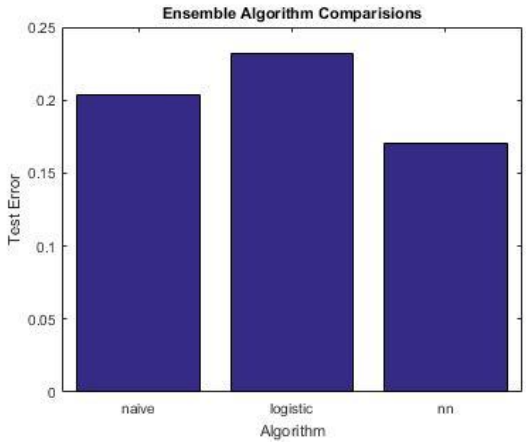
Method	Coeff	Std. Error	Status	P
Naive Bayes	0.2885	0.0324	In	1.8637e-18
PCAed Logistic	0.1479	0.0336	In	1.1345e-05
Logistic Regression	0.1204	0.0353	In	6.713e-04
LogitBoost	0.0409	0.0525	Out	0.4366
Semi Supervised SVM	0.1875	0.0399	In	2.9279e-06
Gentleboost	-0.0024	0.0494	Out	0.9609
SVM	-0.0035	0.0523	Out	0.8792

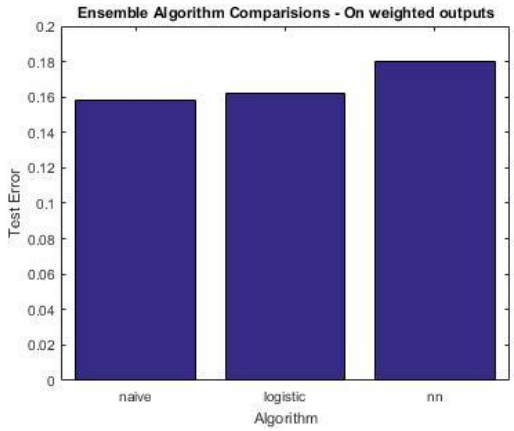
After Stepwise:

- **Naive Bayes: Normal Distribution: Test Accuracy: 84.2%**
- **Logistic Regression: Test Accuracy: 83.8%**
- **Neural Networks: 2 layers - Test Accuracy: 82%**



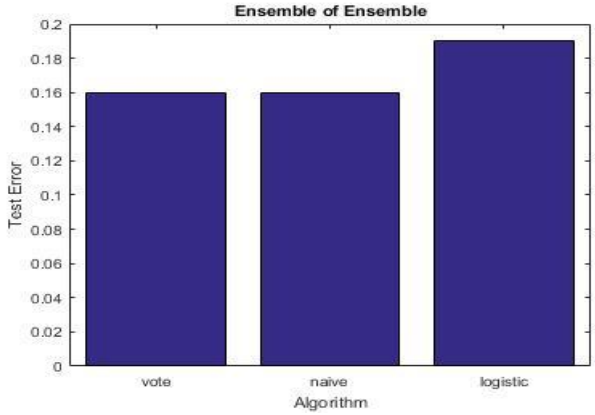
- **Comparison of accuracies before & after using P Weights and N Weights**





- **Ensemble of Ensembles:**

We now had 3 predicted labels from each of the ensemble methods. We took a majority vote of them to return our final 'Y_hat' vector. This returned a **leaderboard accuracy of 81.44%.** This returned a **test accuracy of 84%.** We also did **Naive Bayes (Test accuracy: 84%) and logistic regression (Test accuracy: 81%).**



REFERENCES:

[1] https://lvdmaaten.github.io/tsne/
[2] http://lamda.nju.edu.cn/code_CS4VM.ashx
[3] https://github.com/chrisjmccormick/word2vec_matlab
[4] http://sentiwordnet.isti.cnr.it/
[5] http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=6010
[6] https://in.mathworks.com/help/stats/treebagger.html
[7] https://in.mathworks.com/help/stats/ensemble-methods.html#bsw8aue
[8] http://stats.stackexchange.com/questions/95340/comparing-svm-and-logistic-regression
[9] https://in.mathworks.com/help/stats/ensemble-methods.html#bsw8aue
[10] https://in.mathworks.com/help/stats/ensemble-methods.html#bsw8aue
[11] https://web.stanford.edu/class/cs124/lec/naivebayes.pdf