Comp 135 Intro ML: Project 2

Thomas Klimek

11/20/2018

1. Data Preparation

The data used for this project consists of 3000 reviews each containing a sentence and a sentiment label. To prep this data for our ML algorithms, I used two document vectorization techniques known as *bag of words* and *word embedding*. The data was first removed of numbers, punctuation, and non-lowercase characters to simplify our feature space.

For *bag of words* I represent a sentence as a feature vector over the vocabulary of all words used. I also use the *tf-idf* technique (term frequency-inverse document frequency) to assign different weights to more and less indicative words. This was achieved in python using pandas for reading in the input data, and using scikit learn to build the vectorization and perform the term frequency weighting. Calling the bag_of_words(data) function performs the vectorization on the training and testing data.

I also implement *word embedding* used a dataset of pretrained GloVe embeddings. To use this corpora in my code I utilized the gensim package using glove2vec2word function to read in the glove file to a model, and model.word_vec to get the vector for a word. The key algorithm was to create an array storing each word of a document. Then for each word find the glove embedded weight vector and add it to the total weighting. This is called Sum of Word Embeddings (SOWE).

2. Learning Algorithms & Hyper-parameters

SVM

First a Support Vector Machine algorithm was used to learn the sentiment labels. The SVM was validated on three different types of kernels: sigmoid, rbf, and linear. The hyper-parameters of the sigmoid and rbf kernels are *gamma.* The *gamma* parameter defines how far the influence of a training example reaches. It can be seen as the inverse of the radius of influence of samples selected for the support vector. All 3 kernels also have a *C* parameter which behaves as a regularization parameter in the SVM. The parameter trades off correct classification of training examples against maximization of the decision function's margin. These parameters where tested with crossfold validation of k=10 and the configuration that scored the highest average accuracy was found to be: `{'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}` scoring an accuracy of 77.6% on the testing data. For SVM 72 combinations of hyper parameters where tested from the set `{'kernel': ['rbf'], 'gamma': [1e-2, 1e-3, 1e-4, 1e-5],'C': [0.001, 0.10, 0.1, 10, 25, 50, 100, 1000]}, {'kernel': ['sigmoid'], 'gamma': [1e-2, 1e-3, 1e-4, 1e-5], 'C': [0.001, 0.10, 0.1, 10, 25, 50, 100, 1000]}, {'kernel': ['linear'], 'C': [0.001, 0.10, 0.1, 10, 25, 50, 100, 1000]}` and not all the results were graphed.

Logistic Regression

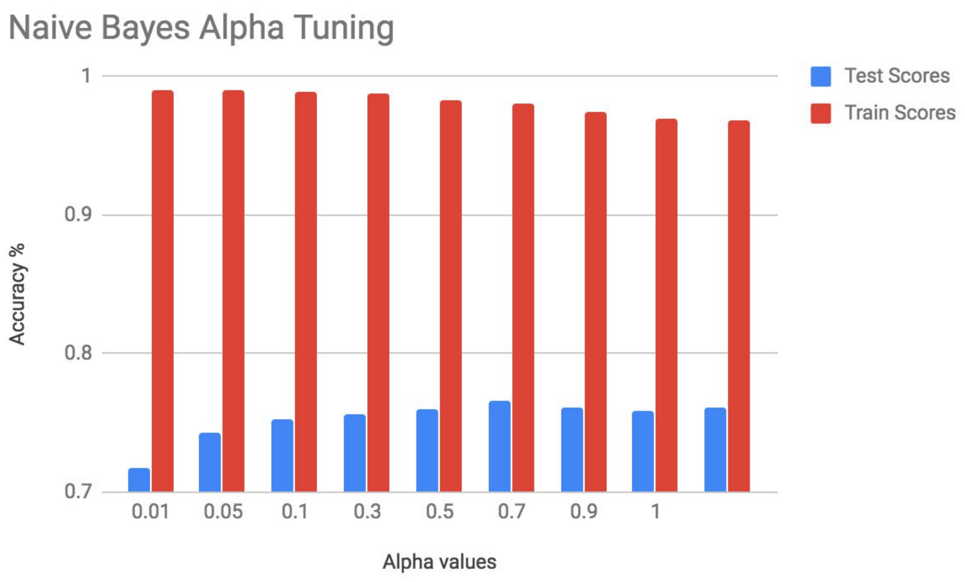
The hyper-parameter tuned for logistic regression was also the C parameter. This parameter affects the regularizer on the regression, however in an inverse manner then the lambda parameter used in class. Small values of C constrain the model while higher values provide more freedom in the margin described previously. We tested on values of C = [0.001, 0.01, 0.1, 1, 10, 100, 1000] using *l2* loss. The best accuracy was achieved with parameters `{'C': 10, 'penalty': 'l2'}` achieving accuracy of 78.6%. This was the highest accuracy achieved on the training set. I was able to iterate through these hyper-parameters with 10-fold validation using the scikit learn module gridsearchcv. I graphed the different training and test errors on different hyperparameter tunings in the graph section.

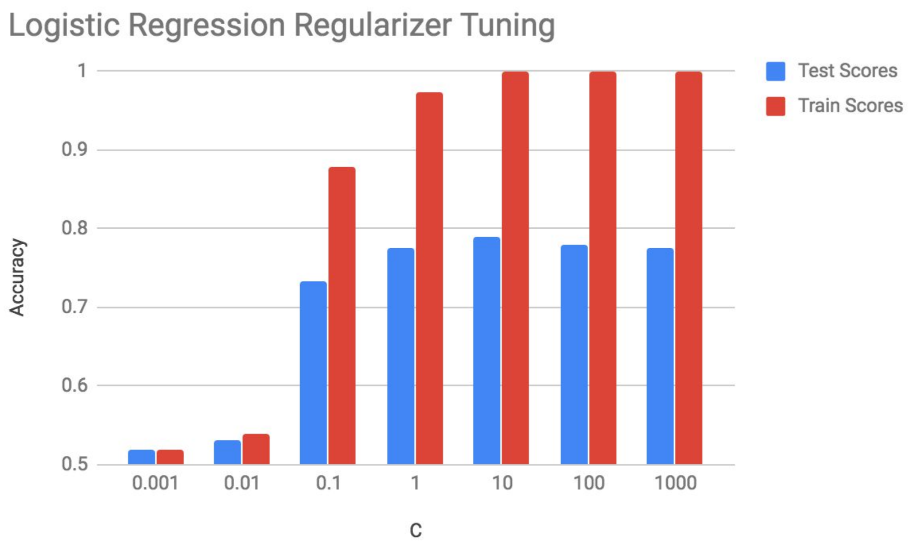
Naive Bayes

For Naive Bayes we tuned the hyperparameter alpha over different values. In scikit learn the parameter alpha = 1 is used to indicate Laplace smoothing, and alpha < 1 is used for Lidstone smoothing. The values tested were `{'alpha': (0, 0.01, 0.05, 0.1, 0.3, 0.5, 0.7, 0.9, 1)}` with the highest accuracy achieved at alpha = 0.5 with an accuracy of 80.5%. I also graphed the performance of the different hyperparameter settings in the graph section.

3. Graphs

These accuracies and figures were generated using the bag of words model, gridsearchcv, scikit classifier functions, and a 70:30 training test split.





4. Final Model

The final model was generated using the function bayes_print(). This function takes the naive bayes model and optimizes the alpha parameter on all the training data, then prints a prediction to the file predicted-labels.txt. I chose naive bayes because it resulted in the highest test accuracy, however I also considered logistic regression as it was able to achieve much higher training accuracy. The hyper parameter settings are `{'alpha': 0.3}` and the predicted accuracy is 80.5% from the cross validation of k=10.