Sentiment Classification

Project Final Report

Name: Tirumal Reddy Ramidi
UID: u1414933
Mail: u1414933@umail.utah.edu, tirumalramidi@gmail.com

Overview of the Project:

The concept of our project is to determine whether some text has a positive or negative sentiment. In other words, we classify any statement into positive or negative contexts. Usually, these classifiers are used to analyze texts to measure the response of a particular tweet or a movie or to capture public perception about the release of certain products into the market. Analyzing the tweets, responses/reviews/feedback to measure the product's effectiveness in the market gives us insights into the improvements we can make to that product.

NLP natural language processing tools are good at classifying these kinds of texts. Some of the applications of these kinds of classifiers are to automatically ascertain whether the reviewer likes the movie/product or not, news or social media text about famous people or companies (to ascertain how public opinion changes about them), and psychotherapy transcripts (to track how the patient's affective state is changing during a therapy session).

In this project, we worked on the well-studied Movie Review Dataset, which consists of a collection of reviews from Rotten Tomatoes.

While working on this project, I have explored different classifiers that predict the sentiment of a featurized movie review. The instances for classification are movie reviews, and the labels are either positive (denoted by 1) or negative (denoted by 0, sometimes -1). While working on this project, I used all three different feature representations of the data for different learning algorithms we discussed in class. The data provided to us is only the feature representations instead of the real-world raw text of the reviews.

I have used five algorithms discussed in the class on a combination of datasets. Overall the max accuracy for this project was given by the neural network library with an accuracy of 83.289

Important Ideas explored:

I have used around five algorithms. Some algorithms don't provide the optimal answers or accuracies in their standard form. For example, algorithms like perceptron standard perceptron are not good enough to get a good answer, so I have used a slightly modified version of the average perceptron in my project.

And also, in algorithms like SVM where the standard SVM or hard SVM doesn’t make sense if the data is not linearly separable, I have used an extended version of this: the Stochastic sub-gradient version of the SVM algorithm, where the basic concept is the same. Still, the way in which we have derived the gradient is an important factor in its implementation. Another important idea I have explored is using cross-validation to find the best hyperparameters that optimize the output.

The logistic regression algorithm uses a sigmoid function. Understanding and implementing the objective function with both the regularization functions as well as the loss function was very satisfying. Implementation-wise, these algorithms are very similar to perceptron algorithms.

Ideas from class:

Based on the lectures on neural networks, I have learned a lot about this exciting set of algorithms. Having multiple layers feeding into its adjacent layer is very attractive. Its very similarity with our human brain is fascinating. Even though there are not enough practical connections between the working of our neural networks algotihms to the human brain, it still is very interesting.

I have used a machine learning library to implement this algorithm. Building a neural network on our own from scratch is very time-consuming. Moreover, many efficient neural networks do the job for us. But I have a firm grip on the concepts discussed in the class for training a classifier.

Concepts like forward pass and backpropagation to calculate the output of a neural network using the sub gradients and sub-sb gradients, and so on, are very interesting. To avoid overfitting and, in turn, reduce the variance of our learners, I have used a concept called dropout. This is a concept on which we had a brief discussion in class.

Basically, what this does is that it ignores a certain number of random examples from our instance space for every epoch and moves forward. In the next epoch, some other random set of a certain number of examples will be ignored. This way, there would be random shuffling, and even the complex neural networks or learners with less training datasets can avoid overfitting and generalize well on future examples.

What did you learn:

One of the important things I learned while working on this project is that one algorithm is not a solution to all types of problems. Even similar algorithms(linear classifiers) produce different outputs on different datasets.

A lot of this also depends on the features we have in our datasets. I have learned how to use some exciting Python libraries to implement complex algorithms like neural networks with multiple layers and use its methods to implement features like dropout, etc. I learned to implement some of the linear classifiers discussed in class, like perceptron, logistic regression, stochastic sub-gradient SVMs, and even decision trees, as an optional exercise.

A summary and discussion of results:

Milestone_1:

Perceptron Algorithm on Roberta features set:

For the first milestone, I have used the Roberta features set among the three feature sets provided. I got the best results using this feature set to train my model. Also, the Average perceptron would give the maximum accuracy; hence I have used this algorithm. I have developed a linear classifier using the perceptron variation using average weights.

For this algorithm to find the only hyperparameter used in this algorithm, I have passed all three values to the algorithm to get the best learning rate that gives the highest accuracy. The hyperparameter value of the learning rate with the highest accuracy of 79.98 is 0.1. Using this best learning rate, I've trained a linear classifier where the epoch is 20. I retained the best weights with good accuracy on the test dataset. With this hyperparameter, I have passed the average weights learned from the training dataset to the prediction function to predict accuracy.

Then, the eval dataset values were read into a pandas data frame, passed the eval_x, and averaged weights learned earlier to the eval_predict function. This predicts the labels on the evaluation dataset. Evaluation accuracy using the Roberta feature set is 80.921

Algorithm	Hyper-Parameter	Features Set	Accuracy
Average_Perceptron	Lr: 0.1	Roberta	80.921%

Lr: learning_rate

I followed a similar process for all the other algorithms to get the best accuracy for the other two milestones.

Milestone_2:

Perceptron on tfidf dataset:

For the second milestone, I used the tfidf features set among the three features provided to train an average perceptron algorithm. The average perceptron would give the maximum accuracy among all variants of the perceptron algorithm. Hence I have used this algorithm. The hyperparameter value of the learning rate with the highest accuracy of 75.934054 is 1. Evaluation accuracy using the Roberta feature set is 75.131

Algorithm	Hyper-Parameter	Feature Set	Accuracy
Average_Perceptron	Lr: 1	TFIDF	75.131%

Lr: learning_rate

ML Library: Neural Networks Algorithm on Roberta:

I have used third-party libraries to train my module for my next submission. I have used the TensorFlow library to train and test the dataset.

```
  tf.keras.layers.Dense(256, activation='elu'),
  tf.keras.layers.Dropout(0.2)
```

I checked the algorithm's performance for multiple dense values and changed the dropout values.

The TensorFlow neural network machine learning module gave me the highest accuracy so far in my project. The Highest accuracy I got after playing around with various dense and dropout values for the algorithm is 83.289

Algorithm	Dense values:	Feature Set	Accuracy
Neural Networks	16, 32, 64, 128, 256	Roberta	83.289%

Milestone_3:

Logistic Regression on tfidf and spacy features set:

For the third milestone, I made 4 submissions, of which I have selected 3 for my final submissions list. I have used the Logistic Regression algorithm on two datasets, a Stochastic sub-gradient SVM algorithm on one dataset, and a decision tree algorithm on one dataset.

I have implemented the logistic regression based on our discussions in class. I followed the same procedures to calculate appropriate loss and regularization function values to update my weight vectors for every epoch. I have used a sigmoid function to calculate the loss values. I have trained, tested, and evaluated this algorithm on tfidf and spacy datasets.

The best hyperparameters on the tfidf set for the logistic regression algorithm are: Learning rate is: 0.001 and best tradeoff_parameter: 1000, and the evaluation accuracy using the tfidf dataset is 71.184.
The best hyperparameters on the spacy set for the logistic regression algorithm are: Best Learning rate is: 0.001 and tradeoff_parameter: 1000 and evaluation accuracy using the spacy dataset, and the best hyperparameters is: 73.421

Algorithm	Hyperparameter values	Feature Set	Accuracy
Logistic Regression	Lr: 0.001, tradeoff: 1000	TFIDF	71.184%
Logistic Regression	Lr: 0.001, tradeoff: 1000	Spacy	73.421

Lr: Learning Rate

SGD SVM on spacy features set:

I have implemented the Stochastic sub-gradient SVM algorithm using the spacy dataset. Unlike the standard SVM (hard SVM), where we update weights for every epoch without any conditional statements, the SGD SVM has conditional statements to update weight vectors for every epoch based on whether a certain example in a particular epoch is linearly separable or not. In SVM (hard), which returns the penalty as either 0,1 for every example, but here for every epoch, we update the weight vectors by calculating the penalty.

The best hyperparameters on the spacy set for the SGD SVM algorithm are: Best Learning rate is: 0.001, the best regularization or loss/tradeoff_parameter: is 10, and evaluation accuracy using the spacy dataset, and the best hyperparameters are: 73.289

Algorithm	HyperParameter values	Feature Set	Accuracy
SGD SVM	Lr: 0.001, loss tradeoff: 10	Spacy	73.289

Lr: Learning Rate

Decision Tree on Roberta Features set:

As my optional submission, I implemented a decision tree algorithm; since the datasets are huge, I could not do cross-validations for this algorithm; instead, I chose the hyperparameter for the decision tree, which is the max_depth as 11. Even running this algorithm once with such huge datasets took me hours.
Because of this time complexity, I couldn't work much more on this, and the accuracy I got for this algorithm for the hyperparameter: max_depth as 11 is 55.

Algorithm	Hyperparameter values	Feature Set	Accuracy
Decision_Tree	Depth: 11	Roberta	55%

Future works:

I would like to understand more about the features used in different datasets in this project. I want to learn more about feature transformations and representations of these features. I want to explore online articles to learn more about these concepts.

I would also like to spend more time implementing neural networks from scratch instead of using Libraries. I would also like to work on implementing Naive Bayes classifiers and AdaBoost algorithms for this project.

I also want to implement all of these algorithms using their corresponding Python libraries and compare these accuracies with my implementations.

The data provided to us is only the feature representations instead of the real-world raw text of the reviews. So I want to focus mainly on feature transformations and how to convert raw text to feature representations(into values).

Conclusion:

Overall, the project is very interesting. Learned a lot by working on this project. Trained different classifiers, converting theoretical knowledge to practical knowledge. The datasets provided are very interpretable, and one can easily use these sets for different algorithms.