Sentiment Analysis with Spark

Anushka Hebbar*, Shree Thavarekere*, Vibha Masti*, Shruvi D*
*Department of Computer Science and Engineering, PES University,
Bengaluru, India
{anushkahebbar, PES1UG19CS460, vibha, PES1UG19CS474}@pesu.pes.edu

***Abstract*—This project explored different sentiment analysis techniques on streaming twitter data using Python and Streaming Spark. Various incremental learning clsssification and clustering algorithms were tried out on batchwise streaming data of tweets. The target sentiment classes were binary in nature, with a class 0 representing negative and a class 4 representing positive.**

***Index Terms*—spark, sentiment analysis, streaming spark, n-grams, hashing vectorizer, logistic regression, Naive Bayes, passive aggressive classifier, k-means clustering, sklearn, incremental learning**

I. Introduction

Social networking sites are very vast sources of information from users all across the world. With the advancement of big data and cloud infrastructure, terabytes of data are constantly being generated. Twitter is one of the most popular social networking sites, as it allows users to freely and publicly express themselves.

Twitter data analysis is a highly useful and popular tool for businesses, as they can use it to understand how their products are received by their users.

Sentiment analysis of tweets is a very popular and useful tool and has gained a lot of traction. In this project we explore different techniques of conducting sentiment analysis on streaming twitter data.

II. Related Work

There is existing literature on performing sentiment analysis with streaming twitter data. In the work done by Elzayady et. al [1], the authors use Spark MLLib to compare the performance of Naive Bayes, logistic regression and decision tree classifiers on increasing dataset sizes. They concluded that NB and LR show the best improvement in performance with an increase in dataset size. The authors Al-Saqqa et. al [2] performed sentiment analysis on the Amazon review polarity dataset, comparing the accuracies of Naive Bayes, SVM and logistic regression. They found that SVM classifiers work the best, followed shortly by Naive Bayes classifiers.

III. Dataset

The dataset chosen is the Sentiment Analysis dataset from the Machine Learning with Spark Steaming [3] project. The data is streamed through a socket every 5 seconds in `json` format using the `stream.py` file. While training, every RDD is preprocessed through a pipeline and passed on to the various models, where their accuracies are compared.

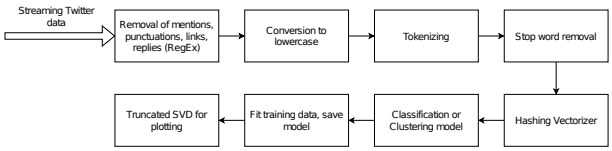

Fig. 1. Preprocessing and Training Pipeline for Sentiment Analysis

IV. Main Files

Two files called `train.py` and `test.py` have been created. These act as the main file for training and testing functionalities respectively. This has been done so that the training and testing aspects of the project are kept separate.

V. Preprocessing

The entire preprocessing and training pipeline is summarized in Fig. 1. The data are preprocessed through a pipeline with the following stages

1) Removal of username mentions, punctuations, links, replied tweets (regex removals)
2) Conversion to lowercase
3) Tokenizing - extraction of words
4) Stop word removal

The data is then further transformed using a Hashing Vectorizer. A Count Vectorizer was experimented with, but it did not yield satisfactory results.

A. Hashing Vectorizer

This has been chosen for 2 reasons.

1) It doesn't store the entire vocabulary in memory. This increases efficiency, especially since the dataset is extremely large.
2) The size of the HashingVectorizer object remains constant, which means the number of features given to the models for training remains constant with each batch. The input parameter `alternate_sign` has been set to 'False' in order to facilitate input to the Multinomial Naive Bayes classifier, since it cannot take negative values.

The Hash Vectorizer has been fixed to a hash table length of 2^{16}, as anything larger than that might be too large for memory to handle, and most collisions should be taken care of within that range. The default size of 2^{20} was too large for spark to run.

B. Count Vectorizer

A Count Vectorizer was also tried for preprocessing, with a custom-written `partial_fit()` function. However, it did not yield good results. This is because when count vectorizer is applied on the input batch, the output vector has a different (increased) size in every increment. To counteract this, PCA was used to restrict the number of features. This worked, but yielded very poor accuracy. In addition, Count Vectorizer stores the entire vocabulary in memory, constantly adding to it at every iteration, which results in hugely reduced efficiency. That is why Count Vectorizer has not been used in the end project.

VI. Classification Models

The following classification models are used, and they have been chosen since they are the most commonly used models for dealing with training of textual data. They have been shown to yield the best results on datasets similar to the one used in this project. All the below models were trained using both Hash Vectorized and PCA reduced data. Of the two, Hash Vectorized data yielded better results. So the end version uses this data.

A. Stochastic Gradient Descent

Scikit-Learn's SGD (Stochastic Gradient Descent) classifier is used, and the `partial_fit()` function is used for incremental learning. The classes are specified as 0 and 4 in accordance with the dataset. 3 variations of ths SDG classifier were trained with, and the results were compared:

1) Logistic regression: loss function - `log`
2) Support vector machine: loss function - `hinge`
3) Perceptron: loss function - `perceptron`

B. Multinomial Naive Bayes

A multinomial naive bayes classifier has been used, and the hash vectorized data is given as input. Scikit-Learn's MultinomialNB classifier has been used here. Again, the `partial_fit()` function is used for incremental learning. Three versions of the model were trained, with the hyperparameter `alpha` (Laplace smoothing parameter) set to one of the following:

1) 1
2) 0.5
3) 0.7

C. Passive Aggressive Model

The passive-aggressive algorithms are a family of algorithms for large-scale learning. They are online learning algorithms, making them perfect for training streaming data. The meaning of the two parts, passive and aggressive, are:

- Passive: if the prediction is correct, keep the model as is (the data in the training example does not change the model)
- Aggressive: if the prediction is incorrect, make changes to the model

The PAC used here fits the hashing vectorized data. Scikit-Learn's PassiveAggressive classifier has been used here. The `partial_fit()` function is used for incremental learning. Three versions of the model were trained, with the hyperparameter `C` (regularization parameter - penalization due to incorrect prediction) set to one of the following:

1) 0.2
2) 0.5
3) 1

VII. Clustering Models

A. KMeans

For clustering, MiniBatchKMeans of Scikit-Learn has been used. The number of clusters has been set to the number of classes in the dataset (two). Here also the hash vectorized data is fed as input to the model, but since this is unsupervised learning, the class labels are not given to the model. The clusters are plotted using matplotlib. The figure is saved to a new file for every batch, which means that a new image is created with each increment. This is done in order to visualize the cluster formation. In order to plot the data in two-dimensional space, PCA with 2 components is applied on the input data.

VIII. Results

We used accuracy as a measure to compare the different models. Accuracy is defined as

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{1}$$

where TP and TN are the number of true positives and negatives respectively, and FP and FN are the number of false positives and negatives respectively.

Each model was tested on different batch sizes - 2000, 3000, 4000, 5000 - and with different hyperparameters/loss functions. The cluster formation is visualized using the plots created for each batch.

A. Stochastic Gradient Descent

The average accuracies (over all test batches) of three SGD models - logistic regression, support vector classifier, perpectron - are summarized in Table I, with the best accuracies of each batch size highlighted.

The accuracies of each batch of the test data on the models trained with a batch size of 5000 are shown in Fig. 2. The accuracies of the best-performing SGD model - SVM - are shown in Fig. 3.

TABLE I
Accuracies of SGD Models

Batch sizes \ Loss	Log	Hinge	Perceptron
2000	0.7432	**0.7455**	0.6872
3000	0.7431	**0.7453**	0.6867
4000	0.7432	**0.7456**	0.6983
5000	0.7432	**0.7456**	0.6909

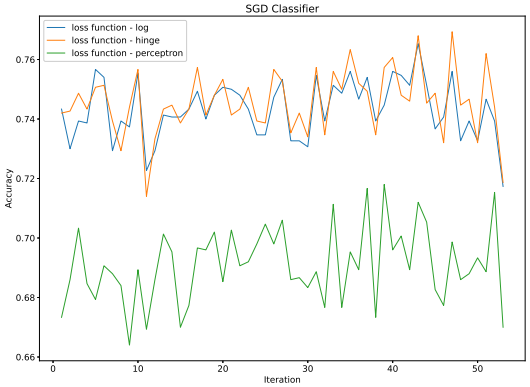

Fig. 2. SGD model accuracies vs test batch iteration

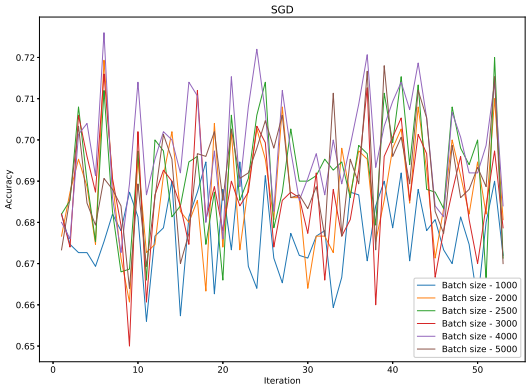

Fig. 3. SVM accuracies vs test batch iteration

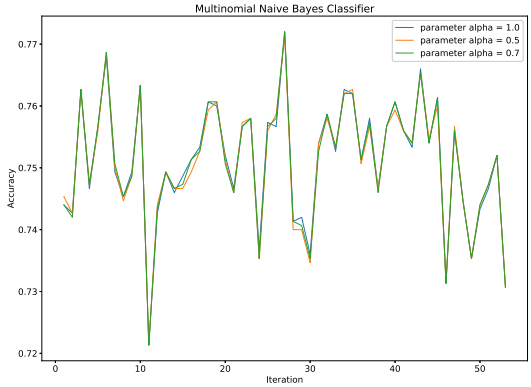

Fig. 4. MNB model accuracies vs test batch iteration

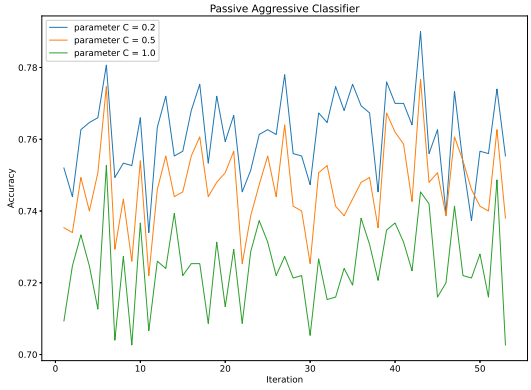

Fig. 5. PAC model accuracies vs test batch iteration

B. Multinomial Naive Bayes

The accuracies of the MNB models are summarized in Table II, with the best accuracies of each batch size highlighted. The effect of the hyperparameter `alpha` (Laplace smoothing parameter) is shown on the different batch sizes.

The accuracies of each batch of the test data with different `alpha` values on the model trained with a batch size of 5000 are shown in Fig. 4.

TABLE II
ACCURACIES OF MNB MODELS

Batch sizes \ Alpha	1.0	0.5	0.7
2000	**0.7511**	0.7509	0.7501
3000	**0.7511**	0.7508	0.7509
4000	**0.7511**	0.7509	**0.7511**
5000	**0.7511**	0.7509	**0.7511**

From the results, we see that tuning the hyperparameter `alpha` has almost no significant effect on the accuracies achieved by the models.

C. Passive Aggressive Classifier

The accuracies of the PAC models are summarized in Table III, with the best accuracies of each batch size highlighted. The effect of the hyperparameter `C` (regularization parameter) is shown on the different batch sizes.

The accuracies of each batch of the test data with different `C` values on the model trained with a batch size of 5000 are shown in Fig. 5. An ROC curve of the best PAC model - `C` = 0.2 - tested on the model trained with a batch size of 5000 is shown in Fig. 6.

TABLE III
ACCURACIES OF PAC MODELS

Batch sizes \ Alpha	0.2	0.5	1.0
2000	**0.7624**	0.7463	0.7289
3000	**0.7623**	0.7461	0.7231
4000	**0.7624**	0.7461	0.7247
5000	**0.7614**	0.7470	0.7246

Models with `C` set to 0.2 perform slighly better than the other two models.

D. KMeans Clustering

The test data were visualised in cluster after performing dimensionality reduction using Scikit-Learn's Truncated SVD class. The clusters formed by a single batch (size 1500) of test data using the learned cluster centroids from the 5000 batch size model are shown in Fig. 7.

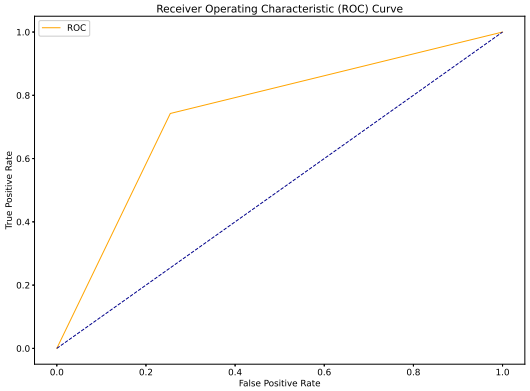

Fig. 6. ROC curve of PAC

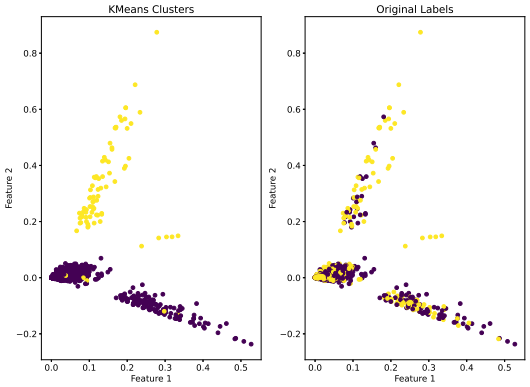

Fig. 7. Cluster formation on training data

The clusters formed did not correctly group the positive and neagtive samples, and it performed no better than a random guesser (accuracy close to 50%).

IX. Conclusion

1) Beyond a batch size of 1000, increasing the batch size during training had little to no effect on the testing accuracy obtained
2) Logistic regression and SVM models outperformed Perceptron learners (weak learner)
3) Tuning the hyperparameter `alpha` had little effect on the accuracy
4) PAC models with `C` set to 0.2 outperformed models with a higher `C` value
5) The best accuracy obtained with large batch sizes for all the weel-performing models is around 0.75

Improvements in accuracy can be made by training with lower batch sizes. Other possible areas of improvement may be to use more than 2-grams in the Hashing Vectorizer to create the TF matrix. In addition, the hash table size may be increased to reduce the occurrences of collisions.

References

[1] H. Elzayady, K. M. Badran, and G. I. Salama, "Sentiment analysis on twitter data using apache spark framework," in *2018 13th International Conference on Computer Engineering and Systems (ICCES)*, 2018, pp. 171–176.

[2] S. Al-Saqqa, G. Al-Naymat, and A. Awajan, "A large-scale sentiment data classification for online reviews under apache spark," *Procedia Computer Science*, vol. 141, pp. 183–189, 2018, the 9th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2018) / The 8th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2018) / Affiliated Workshops. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050918318167

[3] B. D. Teaching Assistants and K. V. Subramanian, "Machine learning with spark streaming project datasets," https://drive.google.com/drive/folders/1hKe06r4TYxqQOwEOUrk6i9e15Vt2EZGC.