Big D|a|t|a

Project Report

Machine Learning with Spark

Sentiment Analysis

Team ID: BD_031_059_231_587

Team Members:

Aditi Killedar	PES1UG19CS031
Anchal Sharma	PES1UG19CS059
Krithika Ragothaman	PES1UG19CS231
Gautham YS	PES1UG19CS587

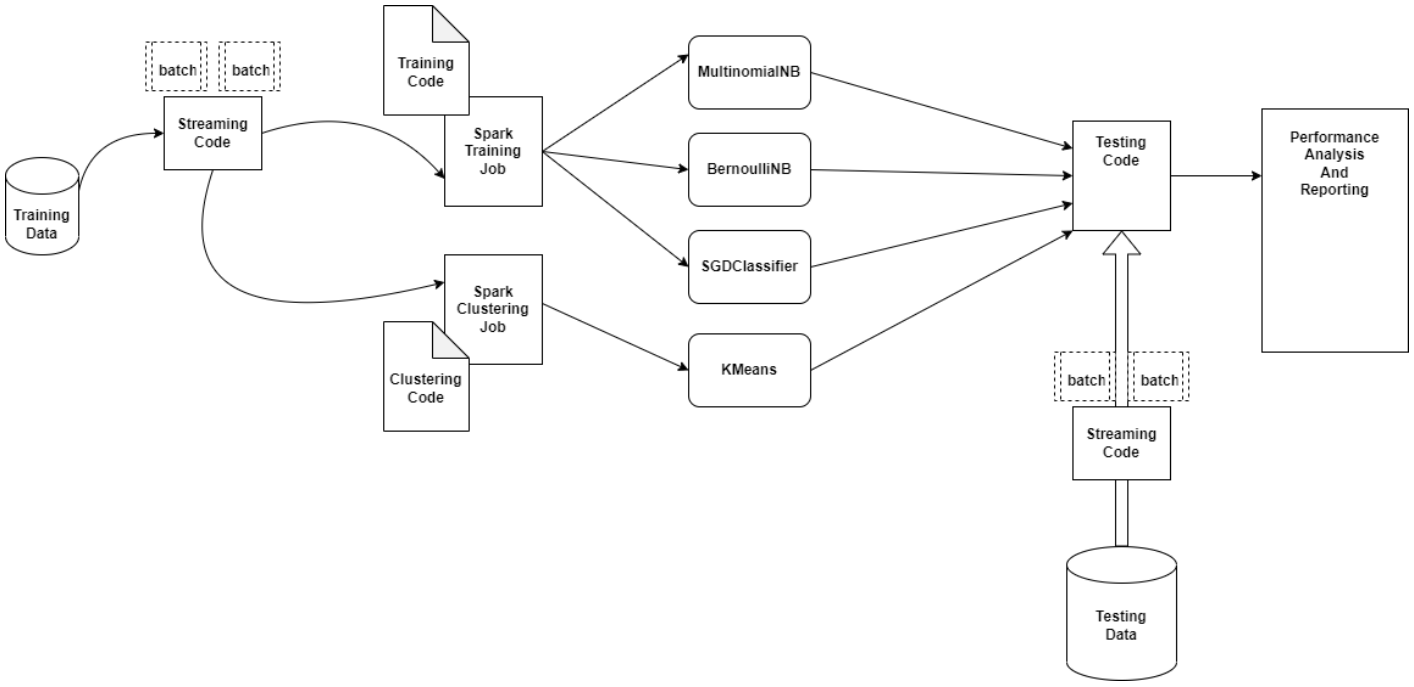


Fig. Design Overview

STREAMING

Data was streamed over a TCP socket stream in JSON format, and a 'spark streaming context' was initialised in our 'client.py' file to stream the training data, and similarly at 'test.py' to capture a stream of test data. **Training data was streamed such that our models could learn incrementally batch by batch.**

PRE-PROCESSING AND TRANSFORMATIONS:

We pre-processed the data from our DStream and converted it into a format that could be passed to our models. The pre-processing stage included removal of stopwords like ('and', 'or', etc) from the tweets. Stop-words were extracted from "nltk.stopwords". Removal of all non-alphabetical characters, like '#', '$', '%' etc from it was also done. We also removed the 'http' links and split each tweet into a list of words. We then lemmatized our words **to make it easier to analyse the context behind each word**. Lemmatization is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form.

We also transformed our DStream into a Spark DataFrame, **to facilitate the ease of its use with MLib**.

Feature Extraction:
After performing the aforementioned preprocessing techniques, which gave us a BOW, we used a HashingTF to map words to word frequencies as a vector of size 1000, this **allowed us to pass this feature vector to our classifiers and clustering algorithms**, which we will discuss below.

MODELS USED:

1. **MultinomialNB** - A multinomial naive bayes classifier
2. **BernoulliNB** - A bernoulli naive bayes classifier
3. **Stochastic Gradient Descent (SGD) Classifier** - an iterative method of gradient descent

We made use of the *partial_fit* function from the scikit-learn library **to support incremental learning**.

HYPERPARAMETER TUNING:

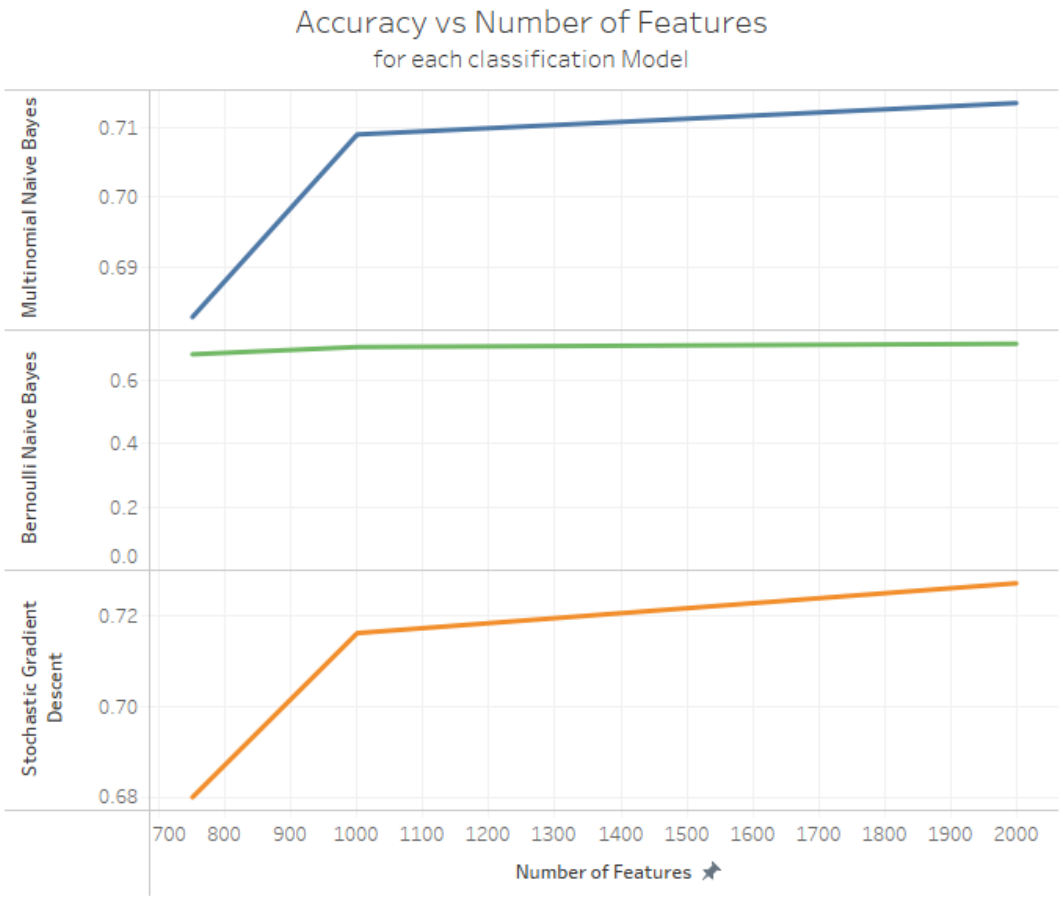
The parameters were tuned using GridSearchCV from scikit-learn
Multinomial Naive Bayes was tuned for alpha value.
Bernoulli Naive Bayes was tuned for alpha, binarizer values.
Stochastic Gradient Descent was tuned for alpha and epsilon values.

VECTORISER

The HashingTF function from pyspark.ml.feature library was used for vectorizing. This function **maps a sequence of term frequencies using the hashing trick**. The vectorizer has a parameter - numFeatures which stands for the number of features the data will be vectorized into. For our data, we compared the different accuracies we achieved from the model by altering the value of numFeatures. When the default value of numFeatures was used (262144), the program crashed due to high memory requirements (19.6GB). The parameter was then changed to a 100, which resulted in a poor accuracy of close to 0.5. The parameter was then subsequently changed to *500, 750, 1000, 1500 and 2000*. The **increase in accuracy slowed down after *1000***, but the

processing time and memory requirement continued to grow. There was only a minor difference when *1000* and *2000* features were taken, and hence we went with numFeatures as *1000*.

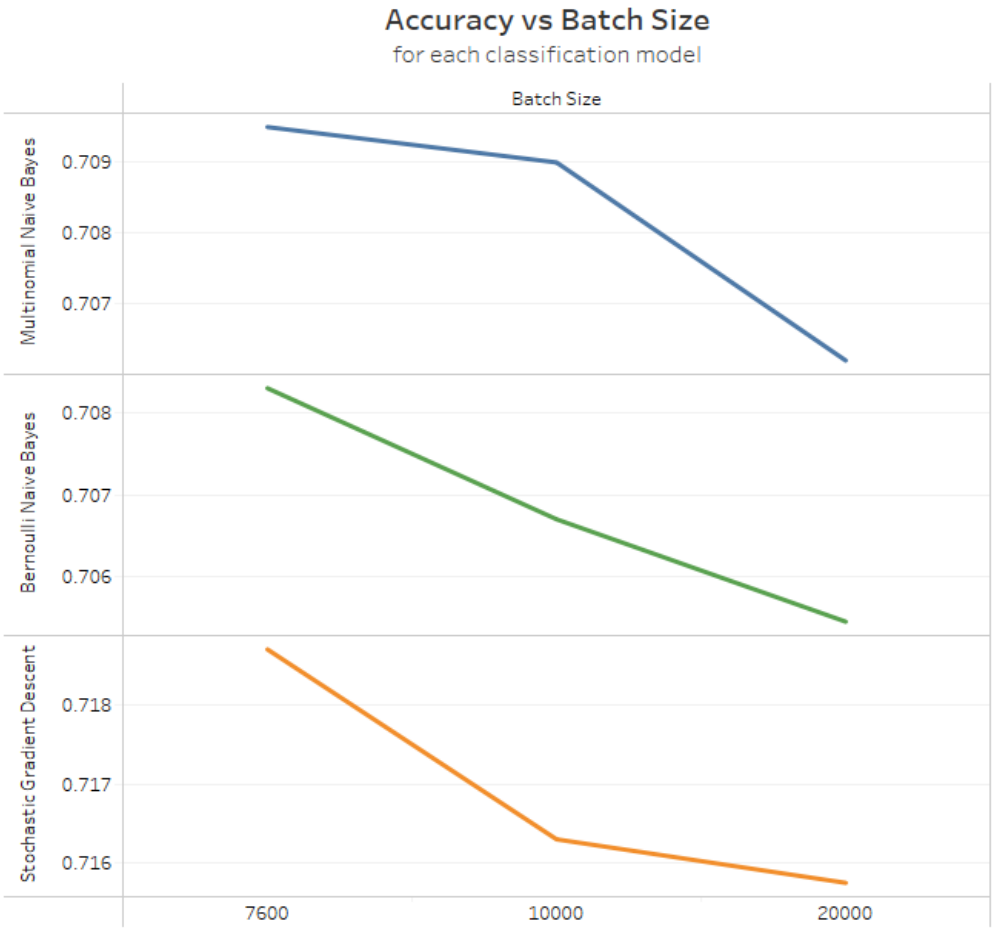
The graph below shows the change in accuracy with tuning of numFeatures. We have selected ***1000*** as the number of features.



BATCH SIZE

We looked at the performance of the models, with batch sizes of *5000, 7600, 10000 and 20000*. Keeping batch size as *5000* (not in graph) gave us worse performance than *7600*. The **batch size of *7600* had the highest accuracy, even when compared with accuracy from batch sizes *10000 and 20000***. The graph below represents the same.

We thus fix our batch size to 7600.

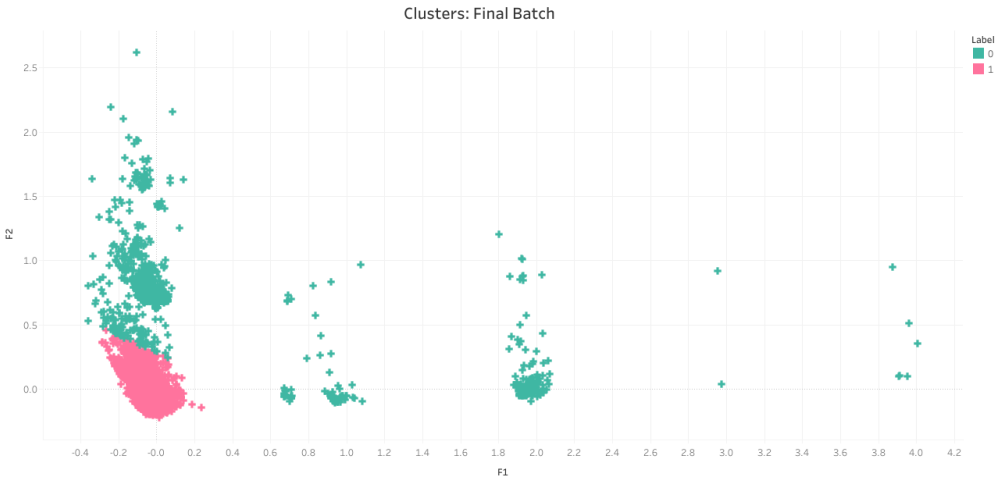


Clustering:

We have used sklearn's miniBatchKmeans to model our data using K-Means. On the train data, the accuracy score function returned **0.4761**. On the test data, the function returned a value of **0.4654**

This model was considered due its **incremental learning** functionality. In each iteration, the batch is used to update the clusters.

The graphs below show the clusters formed in the first and last batches of our incremental clustering algorithm.



Evaluation:

Performance Metrics:

1. BEFORE Parameter Tuning, default parameter values were used for each of the models.

MULTINOMIAL NAIVE BAYES

Label	Precision	Recall	F1 Score	Support
0	0.70	0.71	0.70	4930
4	0.71	0.70	0.70	5070
Accuracy	-	-	0.70	10000

Macro Avg	0.70	0.70	0.70	10000
Weighted Avg	0.70	0.70	0.70	10000

BERNOULLI NAIVE BAYES

Label	Precision	Recall	F1 Score	Support
0	0.70	0.70	0.70	4930
4	0.71	0.71	0.71	5070
Accuracy	-	-	0.70	10000
Macro avg	0.70	0.70	0.70	10000
Weighted avg	0.70	0.70	0.70	10000

STOCHASTIC GRADIENT DESCENT

Label	Precision	Recall	F1 Score	Support
0	0.70	0.71	0.71	4930
4	0.71	0.71	0.71	5070
Accuracy	-	-	0.71	10000
Macro avg	0.71	0.71	0.71	10000
Weighted avg	0.71	0.71	0.71	10000

Confusion Matrices:

MULTINOMIAL NAIVE BAYES

	Predicted	
Actual	0	4
0	3507	1423

4	1583	3532

BERNOULLI NAIVE BAYES

	Predicted	
Actual	0	4
0	3507	1423
4	1583	3532

STOCHASTIC GRADIENT DESCENT

	Predicted	
Actual	0	4
0	3484	1446
4	1459	3611

2. AFTER Parameter Tuning, optimal parameter values were used for the models.

MULTINOMIAL NAIVE BAYES

Label	Precision	Recall	F1 Score	Support
0	0.66	0.7	0.68	4930
4	0.69	0.65	0.67	5070
Accuracy	-	-	0.68	10000
Macro Avg	0.68	0.68	0.68	10000
Weighted Avg	0.68	0.68	0.68	10000

BERNOULLI NAIVE BAYES

Label	Precision	Recall	F1 Score	Support
0	0.66	0.70	0.68	4930
4	0.7169	0.65	0.67	5070
Accuracy	-	-	0.68	10000
Macro avg	0.68	0.68	0.68	10000
Weighted avg	0.68	0.68	0.68	10000

STOCHASTIC GRADIENT DESCENT

Label	Precision	Recall	F1 Score	Support
0	0.70	0.69	0.69	4930
4	0.70	0.71	0.70	5070
Accuracy	-	-	0.70	10000
Macro avg	0.70	0.70	0.70	10000
Weighted avg	0.70	0.70	0.70	10000

Confusion Matrices:

MULTINOMIAL NAIVE BAYES

	Predicted	
Actual	0	4
0	3460	1470
4	1769	3301

BERNOULLI NAIVE BAYES

	Predicted

Actual	0	4
0	3460	1470
4	1769	3301

STOCHASTIC GRADIENT DESCENT

	Predicted	
Actual	0	4
0	3407	1523
4	1482	3588

We can infer that the performance was better before hyper-parameter tuning, this could be due to the fact that there is overfitting.

Takeaways

It was interesting to see how ML algorithms could be implemented on distributed systems using Spark. We learnt how to stream data over TCP, which was something we were not familiar with earlier. We learnt to preprocess and model text-data to classify sentiments in the Tweets, which was also something we were unfamiliar with before this project. Another key takeaway from this, was how we were able to incrementally build our models batch by batch. We observed how tuning the hyper-parameters, and the batch sizes affected the performance of the models, by experimenting with both of them.

We would like to thank KVS Sir and our TAs for their guidance and support and for the opportunity to do this project. Doing this project enabled us to understand working with Spark and using it for Machine Learning tasks with Big Data.