Machine Learning using Spark Streaming - Sentiment Analysis

Team Number: BD_090_102_563_577

Team Members:

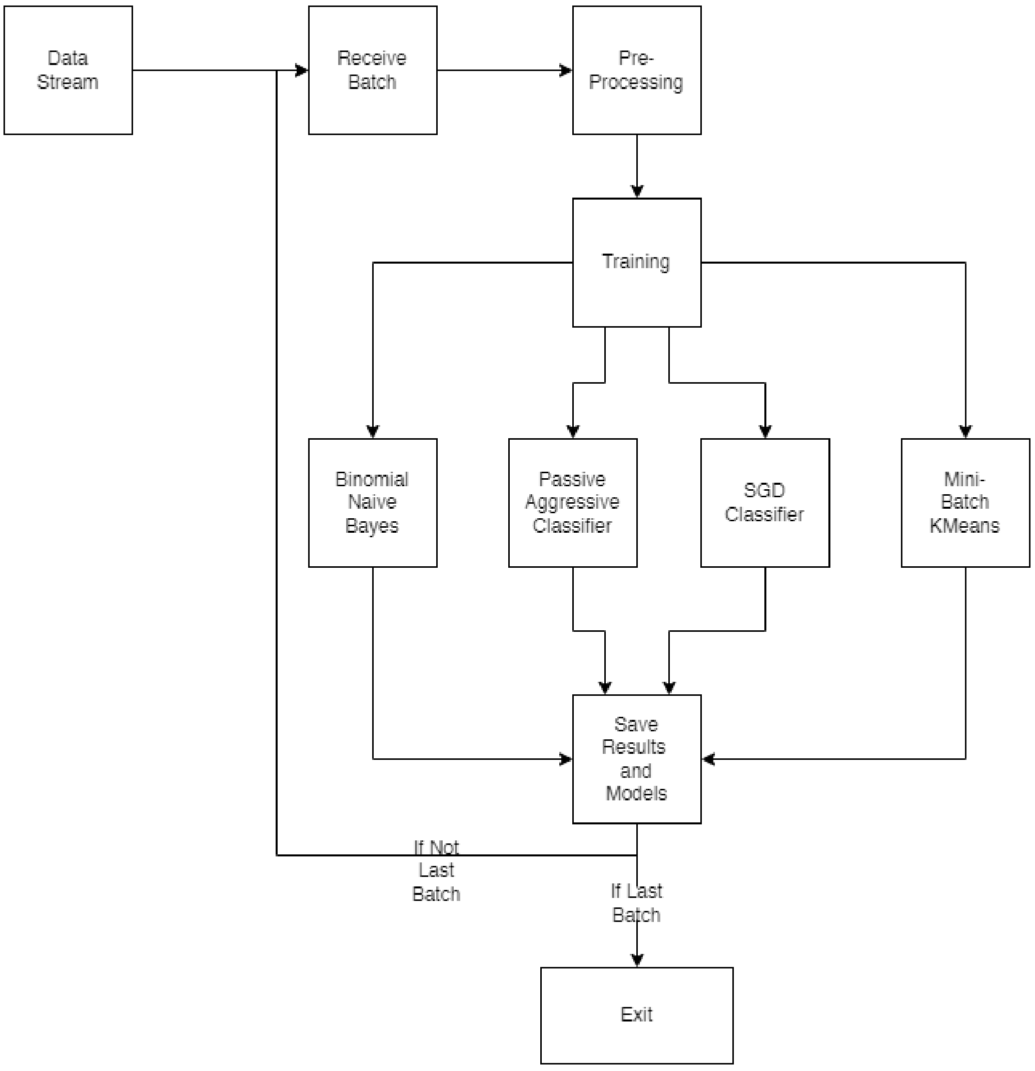
1. Arvind Krishna - PES1UG19CS090
2. Sachin Shankar - PES1UG19CS102
3. Suhas Thalanki - PES1UG19CS563
4. Vishruth Veerendranath - PES1UG19CS577

Project Repository: https://github.com/Big-Data-Gang/Machine-Learning-with-Spark-Streaming

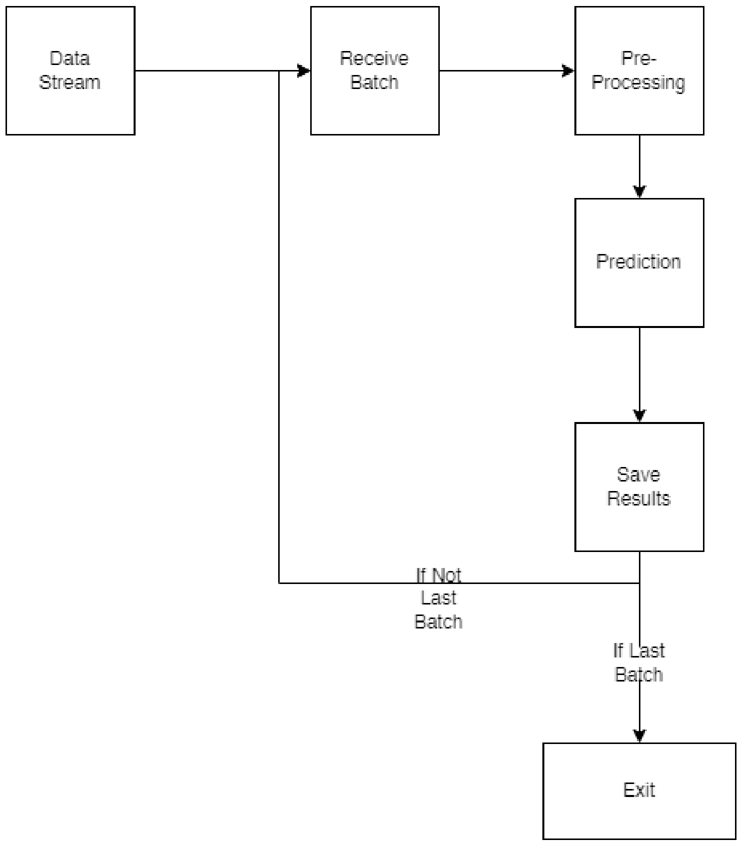
Project title chosen: Machine Learning Using Spark Streaming Sentiment Analysis

Flow of the System

Training Flow



Test Flow



Surface Level Implementation

Receiving Data

- Data Streams were received in `app.py` over a **TCP** Connection.
- DStream was operated on using `flatmap` to split it into individual tweets.
- This was then converted to an RDD and used for pre-processing.

Preprocessing

A pre-processing Pipeline was built with the following steps: - **documentAssembler** - Prepares data into a format that is processable by Spark NLP. - **Tokenizer** - Prepares data into a format that is processable by Spark NLP. - **Normalizer** - Prepares data into a format that is processable by Spark NLP.Prepares data into a format that is processable by Spark NLP. - **Stopwords Cleaner** - This takes a sequence of strings and drops all the stop words from the input sequences. - **Stemmer** - Returns hard-stems out of words with the objective of retrieving the meaningful part of the word. - **Lemmatizer** - This will generate the lemmas out of words with the objective of returning a base dictionary word. - **Finisher** - Converts annotation results into a format that easier to use - **HashingVectorizer** - Convert a collection of text documents to a matrix of token occurrences

Training

1. Classification:

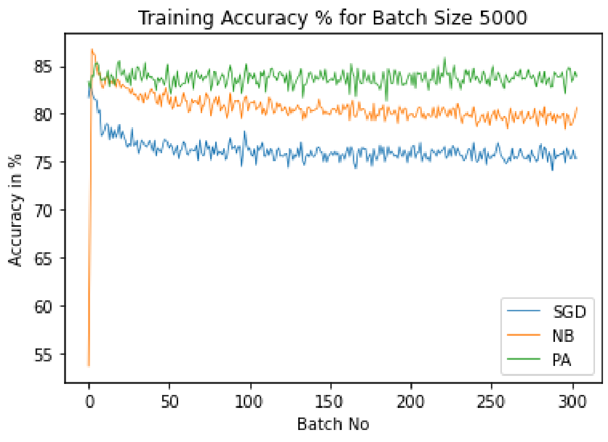
- **SGDClassifier** - Linear classifiers (SVM, logistic regression, etc.) with SGD training.
- **BernoulliNB** - Naive Bayes classifier for multivariate Bernoulli models.
- **PassiveAggresiveClassifier** - Passive Aggressive Classifier works by responding as passive for correct classifications and responding as aggressive for any miscalculation

2. Clustering:

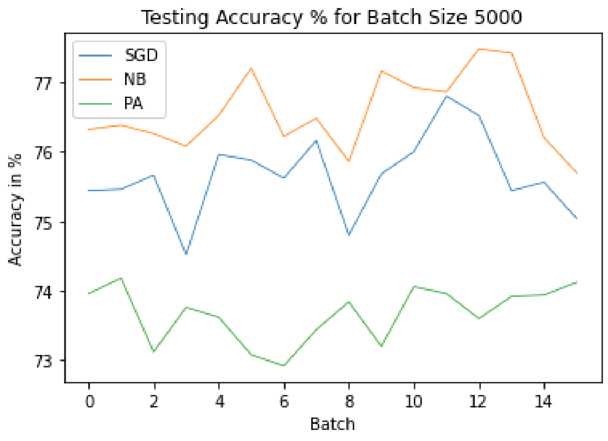
- **MiniBatchKMeans** - Mini Batch K-means algorithm works by using small random batches of data of a fixed size

Plotting

The tested model with its insights like accuracy, recall, score, precision & F1 was later plotted corresponding to batch numbers and size to evaluate the effectiveness of incremental learning. The plots of different models were also plotted to draw conclusions on effectiveness of the same



As seen in the above graph, the `Passive Aggresive` model performed the best in **training**. This trend was observed across other **Training** batch sizes as well.

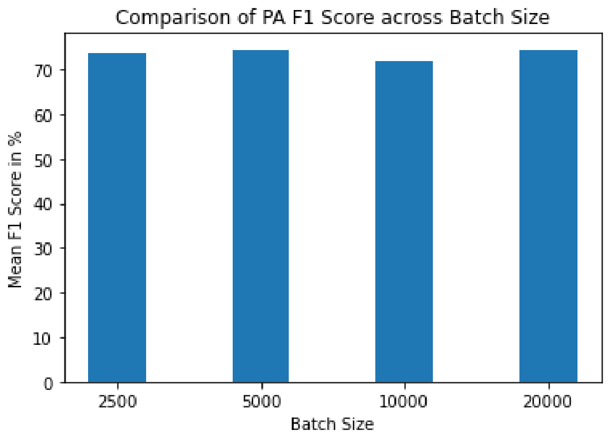


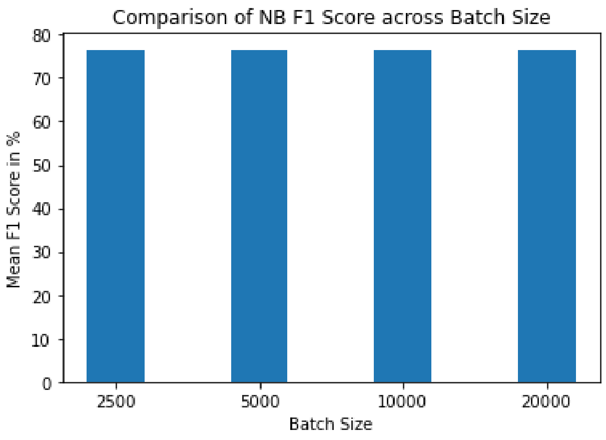
As seen in the above graph, the `BernoulliNB` model performed the best in **testing**. This trend was observed across other **Testing** batch sizes as well.

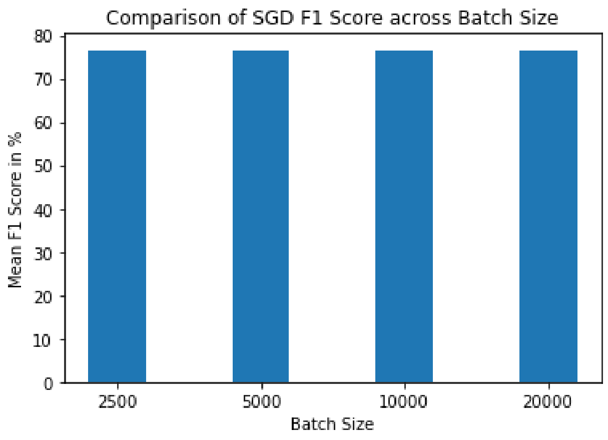
From the above two graphs, it can be inferred that the Passive Aggresive Classifier while having a higher Training Accuracy, had a worse Test Accuracy which is an indicator of **overfitting**.
Due to this, we conclude that **BernoulliNB** is the most suitable model.

Effect of Training Batch Size on Accuracy of models

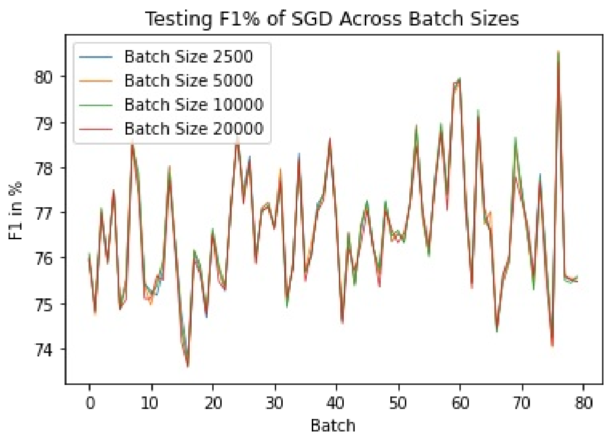
Models trained over various Batch Sizes were tested on the Test Set with a Batch Size of `1000` in order to get a standard accuracy unaffected by testing batch size. The only variable after this is done is the training batch size that the models were trained on.



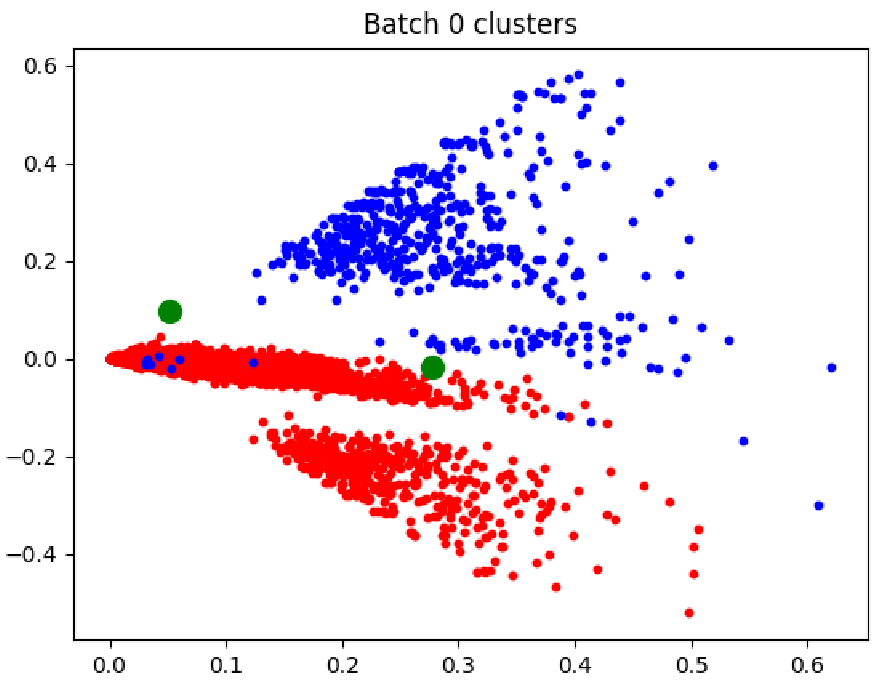


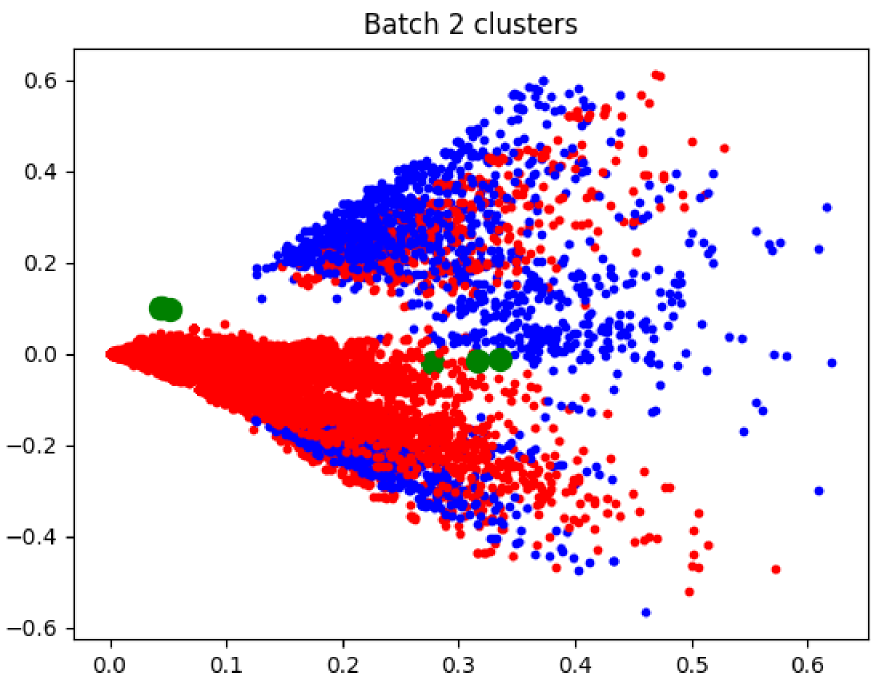


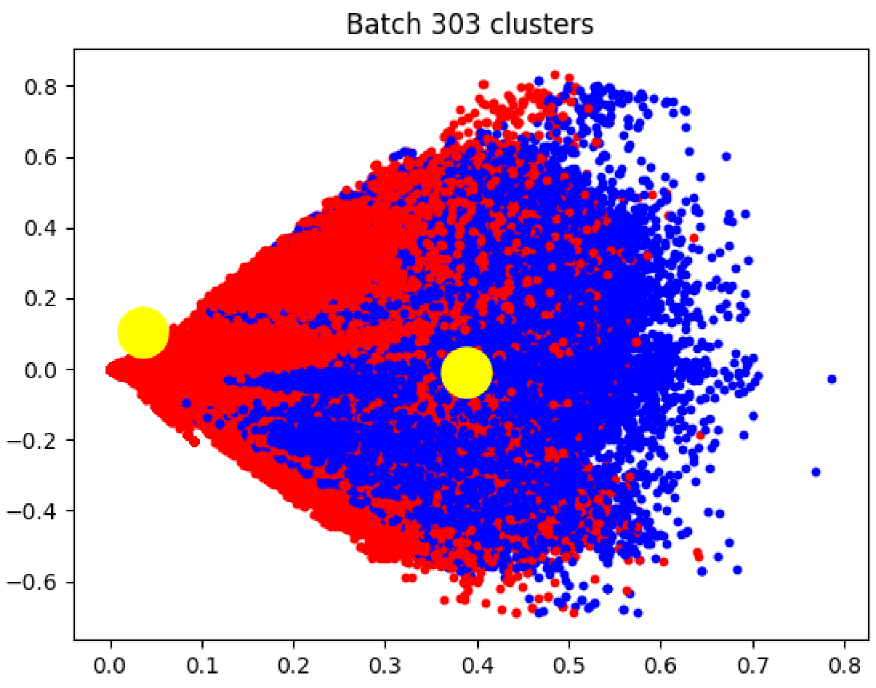
- From the above graphs we can see that the batch size doesn't really affect the accuracy of the model, it just changes the time of training and memory consumption. Also the PA classifier is slightly more affected by the batch size compared to the other models.



  - While considering KMeans plotting, we can see the trend of movement of centroids in the plane with the addition of each batch of data. At the end of the streaming, i.e. in the final batch, we can see that the points always cluster around the centroid for a given label.

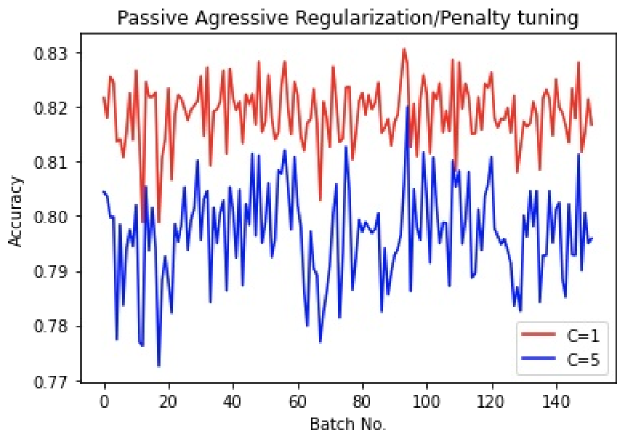




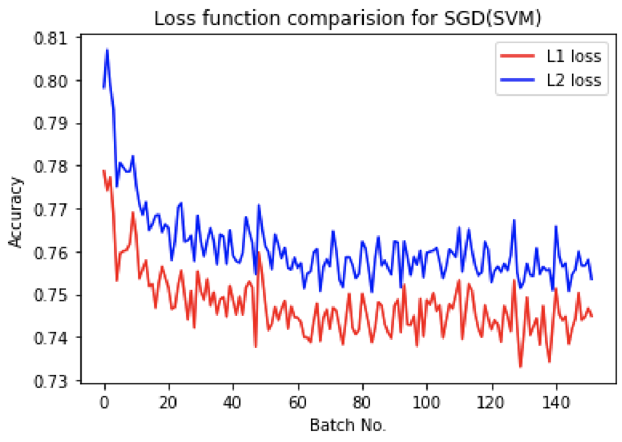


Hyper-Parameter Optimisation:

We have searched through a few hyperparameters for each of the classifiers, keeping everything else the same. The plots for PA and SGD classifiers are below.



The **c parameter (max step size) in PA** being tuned clearly gives a better model overall i.e, *c = 1 gives a better model than c = 5.*



By comparing the **penalty functions in SGD** - L1 & L2 and we can conclude that *L2 is better hyperparameter/penalty than L1.*

Drawing Conclusions:

- Considering the above plots and further analysis, we can conclude that the Naive Bayes is the best classifier for the given dataset and our device capabilities. It doesn't suffer from either overfitting or underfitting too much.
- From the graphs, it's obvious that clustering is not a good idea for the given dataset due the huge size of the vectorized text and its dimensionality. Even in the graphs we are losing so much information due to the dimensionality reduction techniques used(PCA).
- Also given the Time-Space tradeoff, a batch size of 5000 was found to be best for our device configuration and available memory space. It best utilized the memory while not being too slow either.

Reason behind design decisions:

- The Hashing vectorizer was used since it is not necessary to store the vocabulary dictionary in memory, for large data sets it is scalable in very little memory thus causing it to be highly efficient
- SGDClassifier was selected because It is easier to fit in the memory due to a single training example being processed by the network.
- Bernoulli Classifier was chosen since its the one of the few NaiveBayes classifiers which will take up Sparse Matrix as the parameter and thereby reducing the need to convert it into a full matrix
- PassiveAggressiveClassifiers were chosen because when working with memory-constrained devices, you may not able to keep all the training data in memory: passive-aggressive classifiers may help solve your memory problems.
- MiniBatchKMeans has the advantage of reducing the computational cost by not using all the dataset each iteration but a subsample of a fixed size.

Takeaway from the project:

- Learnt about and worked with Distributed File Systems such as Hadoop and their management.
- Worked with Data Streaming in order to work with large datasets which would not fit into memory otherwise.
- Learnt Incremental Learning and how it can be used to continuously improve models and also work with Data Streaming and Huge Datasets.
- Improved understanding of Machine Learning.