

Big Data Project Report

Machine Learning with Spark MLlib

Team Members

Abdul Mannan PES1UG19CS007

Ishita Chaudhary PES1UG19CS190

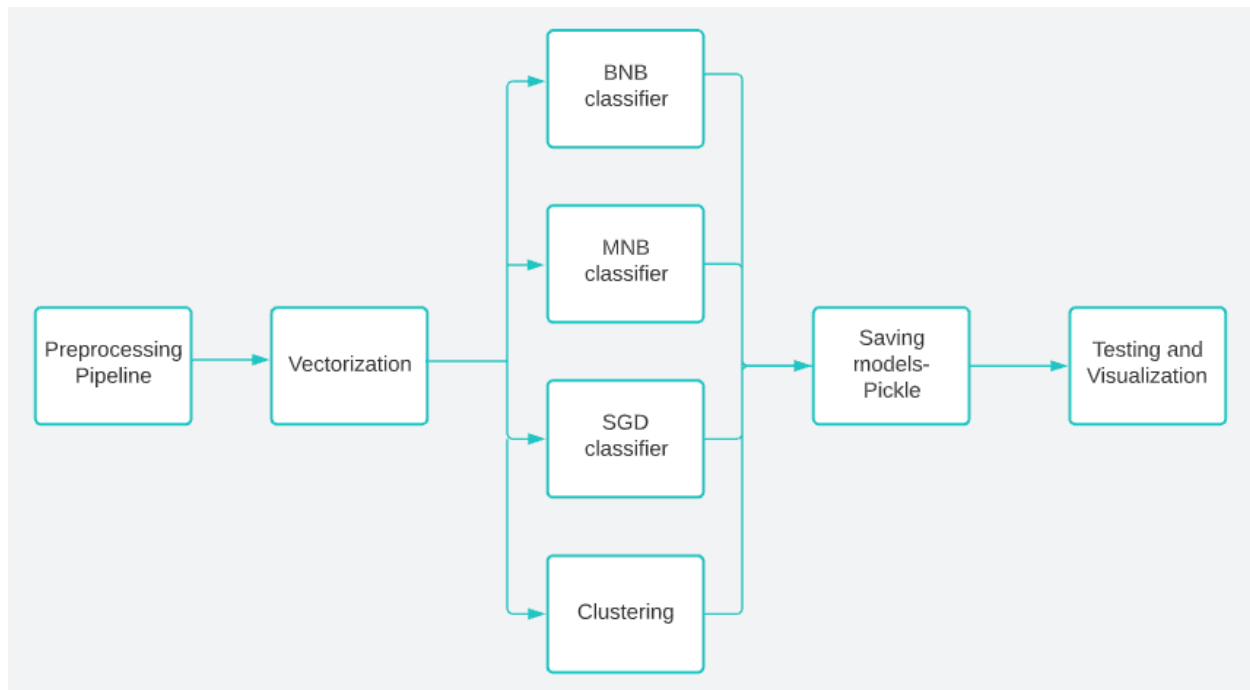
Rithika Shankar PES1UG19CS387

Dhruv Menon PES1UG19CS143

Project Title Chosen:

Machine Learning with Spark MLlib- Sentiment Dataset

Design Details:



1. **Stream** the input, convert it to spark **dataframes**
2. Build the **pipeline** for preprocessing
3. Using the **Hash Vectorizer** to convert the words to vectors, and for **Normalization**
4. Building and Training 3 **classifiers**- Stochastic Gradient Descent, Multinomial Naive Bayes, Bernoulli Naive Bayes
5. We save the classifiers and vectorizers using **Pickle**.
6. For **clustering**, we use MiniBatchKMeans.

Surface Level Implementation Details:

1. Taking the **input**:

- Streaming the data using the stream.py file with the appropriate parameters like file path, batch size
- We convert this dstream (which is a collection of RDDs) into spark dataframes for ease of use.

2. **Preprocessing** the text: Below are the methods used


- **Cleaning** the text: This involves removal of special characters etc
- **Tokenizing**: Tokenization is essentially splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms. Each of these smaller units are called tokens.
- **Stop word removal**: Stop words are commonly occurring words in a language like 'the', 'a' and so on. They can be removed from the text most of the time, as they don't provide valuable information.

3. Converting training dataset into **numpy arrays** and separating out the features and the target label.

4. We then use sklearn's **Hash Vectorizer** to map words to a corresponding vector of real numbers. This is also used to **normalize** the values. This text vectorizer implementation uses the hashing trick to find the token string name to feature integer index mapping.

5. We then **incrementally train** our three classifiers - Stochastic Gradient Descent, Multinomial Naive Bayes, Bernoulli Naive Bayes- for each batch of data. All these are implemented using scikit-learn. The best hyperparameters for Bayesian models were observed to be the default parameters. For SGD, the 'loss of elastic net' was the most suitable hyperparameter.

6. We **save and load** the classifiers and vectorizers using **Pickle**. These are saved in '.sav' files.



7. Testing: We **test** the models on the test dataset streamed in **batches** and collect the data in csv files. For the classifiers, we calculated the Precision, Recall, F1 score and Accuracy of the models.


8. For clustering, we use sklearn's incremental clustering technique **MiniBatchKMeans**. Mini-batches are subsets of the input data, randomly sampled in each training iteration. These mini-batches drastically reduce the amount of computation required to converge to a local solution. The hyperparameters found to be most suitable were- cluster-size= no of target variables, init= 'k-means++', and batch-size= 10000.

Reason Behind Design Decisions:

1. Pre-processing: We pre-process each tweet to make it into a **digestible form** for the machine learning model so it can perform better. We remove the unnecessary special characters which hold no significance in sentiment of the tweet.
Tokenization is done to split the tweet into words so we can further refine the text by removing stopwords, which are nothing but commonly occurring words like the, a, be, etc which have no value in sentiment analysis.
2. We convert the text and labels into numpy arrays so we can do the vectorization on it since sklearn's HashVectorizer requires input to be a np array. We use HashVectorizer over Tf-Idf vectorizer since Tf-Idf vectorizer uses a constant vocabulary of words which cannot be guaranteed across different batches, hence HashVectorizer is used since it has **constant feature size**. It is also very quick to pickle and unpickle.
3. We use Stochastic Gradient Descent, Multinomial Naive Bayes, Bernoulli Naive Bayes models from the sklearn library since they **support incremental learning** and are known to perform reasonably well in text classification problems.
4. Pickle allows us to persist our trained model across **batches of training**.
5. MiniBatchKMeans is the only supported known **incremental clustering** technique.

Take Away from the project:

We understood the working of Spark streaming and agreed that it provides a **scalable and efficient system** within very low latencies. It is very suitable for **real time data**



processing and its approach allows batch and streaming workloads to operate simultaneously.

Our results from the incremental learning models show its **high accuracy and prediction capabilities**. With increasing data, the model will only keep getting better. The incremental learning approach allows us to train models on **large amounts of data** upwards of even a few Terabytes which cannot be stored in memory at once, hence such a method has a lot of real world applications.