SAN-FRANCISCO CRIME CLASSIFICATION USING PYSPARK

-BD2_041_057_105_181

Design Details:

Our implementation mainly consists of two files, train.py and test.py. Data is read from port 6100 of the localhost machine. Streaming is done with the help of stream.py file. All the pre-processing and training techniques are written in different modules, to make the code more readable and easily understandable.

Train.py is used to perform all the required operations on the training data. Similarly, test.py is written to perform the required operations on test data. The batch size used is 50k. The models are saved in '.sav' files, for loading up during testing. Since the data comes in streams, incremental learning is performed.

Surface Level Implementation:

Training the dataset:

The datastream is iterated through and the following functions are run on each batch of rdds. The batch duration is set as 3 and the each of the RDD is obtained using the 'foreachRDD' function.

- readstream(rdd): This module reads the data from the stream and performs preprocessing. The schema of the incoming data is already specified. The data is converted to a dataframe from json format. The columns Category(here feature1) , District (feature4), Day (feature3). The date is converted to timestamp type and then the hour, month and the year are extracted.
- x_y(rdd): The required features are extracted and they are divided into x and y respectively. The dataframe is converted to a numpy array to train the models.
- model_train(rdd): This function trains all the models on the incoming batch and saves the model for the final predictions on the test set.
- save_model(): Dumping the model weights for testing and predictions.
- test_train(X,y): Divide the data into train and test sets.
- naïve_bayes(), stgd(),passive_agg(),minibatch() : Fitting the clustering and the classification models on the incoming batch data.
- metrics(): The classification metrics like the classification report and the accuracy are printed.

Testing on the test set:

foreachRDD is used to iterate through the datastream.

- readStream(rdd): Works the same way as the readStream function of train.py. The category column is not present in the input stream. The id and the required dataframe are returned.
- test(): Loads all the models and runs the predictions on each batch of the test data. The predictions are gathered in a list and converted to integer to decode the labels. These are merged along with the id of the crime to give the final output.
- predictions(filename,df): Predicting the output on the test dataframe df.
- datatype(lis,dtype='int'): To convert the label from str to int

Reason behind design decisions:

Modularity makes the code more readable. Hence, the implementation is divided into functions. The training stream and the test stream cannot be read in the same file. Hence the training is separated from the testing. The timeout was set as 110. This is the optimal time to run the file with a batch size of 50k. The batch size chosen takes lesser time and also give out a higher accuracy amongst all the other batch sizes that have been tried.

Takeaway from the project:

Of all the models that we trained,Naive Bayes gave the highest performance.

We learnt that spark is really helpful in situations where a huge amount has to be used. We also learnt about incremental learning for streaming, which makes real time predictions much easier. There is a lot of class imbalance in the given dataset. So, pre-processing techniques like scaling and label encoding and hyperparameter tuning did not really help us in increasing the performance. We found out a few overlapping labels in a few batches for the category column, because of a high number of categories(39). So, we had to manually encode the columns.

On the whole, it was really helpful learning experience.