**Assignment-3**

**Report: Predicting Click-Through-Rate using Logistic Regression**

The dataset for this Assignment has been taken from the <https://www.kaggle.com/c/avazu-ctr-prediction/data> i.e. Kaggle CTR Prediction dataset. It has 2 sets of files. One is train.csv as the Training dataset and the other one is test.csv for the Test Dataset. The test.csv does not contain labels so I have divided my train.csv files into 80% training dataset and 20% test dataset and using this test dataset as my main Testing file.

The data description for the train.csv file is:

* id: ad identifier
* click: 0/1 for non-click/click
* hour: format is YYMMDDHH, so 14091123 means 23:00 on Sept. 11, 2014 UTC.
* C1 -- anonymized categorical variable
* banner\_pos
* site\_id
* site\_domain
* site\_category
* app\_id
* app\_domain
* app\_category
* device\_id
* device\_ip
* device\_model
* device\_type
* device\_conn\_type
* C14-C21 -- anonymized categorical variables

**TASK: 1**

**Data Pre-Processing:**

Columns “id” and “hour” were excluded in the Data Pre-processing step. The Binary attribute CLICK is the target variable(i.e. label to be used for predictions). Almost all attributes were categorical attributes and had to be converted to numerical attributes using String Indexer and One-hot Encoder (from pyspark.ml.feature).

**Data Cleaning:**

Columns with null values (like C1, C20, etc) were replace by string “NA”. Numerical attributes with null values were dropped. Target attribute “CLICK” was converted from String to Double type.

**Feature Extraction:**

Attributes having more than 100 distinct values were dropped and Data Frame is created with the rest of the columns. Using Vector Assembler, all the categorical columns were converted into Vectors into one single column. Inducing all columns’ String Indexers and One-hot Encoders and Vector Assembler’s output into the Pipeline, so as to fit this model into the main DataFrame.

The two columns “click” and “Vectored Features” (Containing vectors of all the categorical columns into one column) were extracted into a model.

**Splitting into Training and Test Sets:**

The model thus formed after Feature Extraction stage was then splitted into two parts: ClickTrain and ClickTest with 80% Training Set and 20% Test Set.

**TASK: 2**

Using Logistic Regression to train the model with the training dataset:

1. Optimizing the parameters using Gradient Descent:

We train the model using LogisticRegressionWithSGD (from pyspark.mllib.classification) with train method by passing parameters as iterations=15, step=1, miniBatchFraction=1, regType=None, validateData="False".

We record the time to run the Gradient Descent on training dataset which is 110.34 minutes.

1. Optimizing the parameters using Stochastic Gradient Descent:

We train the model using LogisticRegressionWithSGD ( from pyspark.mllib.classification) with train method by passing parameters as iterations=100, step=0.01, miniBatchFraction=0.01, regType=None, validateData="False".

We record the time to run the Gradient Descent on training dataset which is 148.36 minutes.

1. We obtain the graph of time to train vs. size of the training set to compare both methods, for the best optimized parameters thus obtained.

From the graph, it can be inferred that the time to train the training model using Stochastic Gradient Descent was more than the Gradient Descent.

**TASK: 3**

Using Logistic Regression with Stochastic Gradient Descent to perform testing on the test dataset.

We test this model with the Test Dataset and predict the features for the labels in the Test Dataset and obtain the following Evaluation metrics:

Accuracy: 0.8341

AreaUnderROC= 0.5

False Positive Rate= 0.0

**TASK: 4**

Retraining the model using L2 Regularization along with the Stochastic Gradient Descent.

We perform the similar Data Preprocessing, Data Cleaning, Feature Extraction and Splitting the data into training and test datasets.

Now, again, we split our training dataset into Training Set(60%) and Validation Set(40%). Training this training dataset by passing different values for regParam (cost parameter) and setting:

regType=’l2’, convergenceTol=0.001, iterations=100, step=0.01, miniBatchFraction=0.01 and validateData=’true’

and testing on the Validation dataset. The cost parameter regParam needs to be tuned to get the highest accuracy.

I ran three models with three different parameters passed to the train method of LogisticRegressionWithSGD and obtained the best set of parameters for which the accuracy was high i.e. Accuracy = 0.8332 for regParam=0.5

Now, I tested it over the main Test Dataset with the best choice for values of parameters and obtained the following Evaluation metrics:

Accuracy: 0.8341

AreaUnderROC= 0.5

False Positive Rate= 0.0

With the Validation Dataset, the accuracy dropped by 0.001% but when tested on the main Test Dataset, it raised up to 0.8341