Stroke Prediction by Spark

1. Introduction:

According to the World Health Organization, ischaemic heart disease and stroke are the world's biggest killers. What we try to do is to predict the stroke probability using the given information of patients in a real-time situation. It is a classification problem, where we will try to predict the probability of an observation belonging to a category (in our case probability of having a stroke). There are several techniques built on the top of Spark involved in this project.

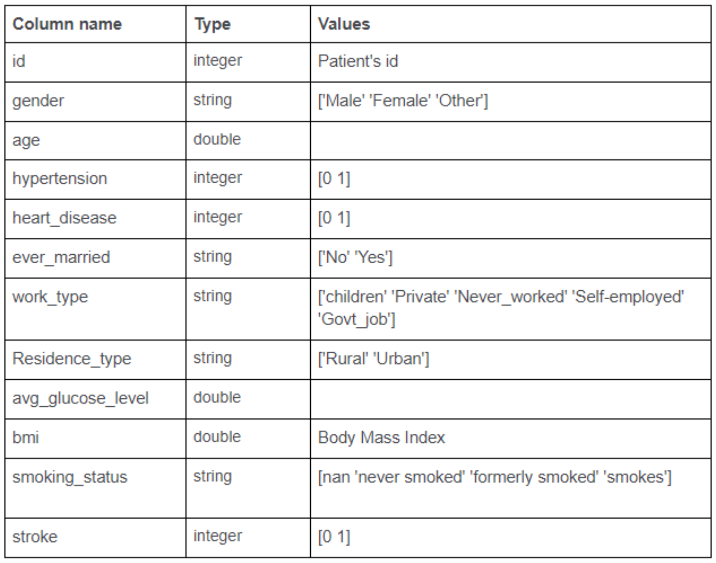
2. Objective:

There are three main objectives for this project. First, using various machine learning algorithms to predict the stroke probability with given information of patients. This could be a useful application combined machine learning and big data technology for healthcare in industry. Second, the team conducts data analysis on dynamic data which was imported with Spark Streaming. Third, this project tends to be an important step for the team to practice big data technologies. This project is combining streaming data pipelines with data science on top of Spark using Databricks. This could provide a great opportunity for the team to practice techniques learned in class.

3. Project Design:

3.1. Data

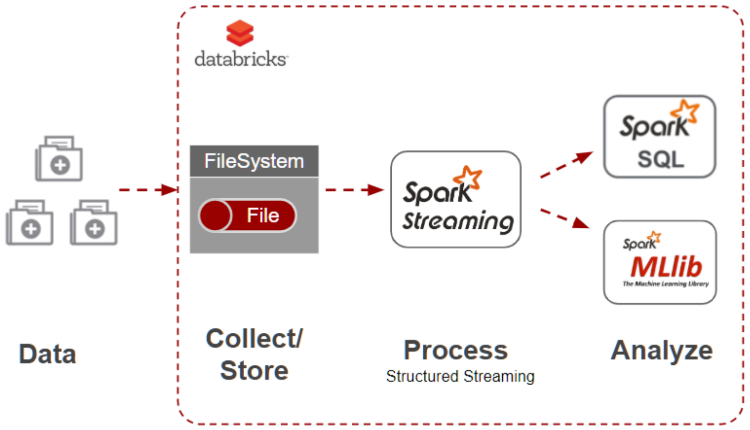
Dataset is acquired from a hackathon on Analytics Vidhya for Mckinsey dataset of healthcare. The McKinsey Analytics Hackathon provides participants the opportunity to experience and overcome some of the challenges that leading global organizations face. There are 43,400 records in the training dataset and 18,600 records in the test dataset. Since a structured streaming technique is used to import data into the system, the training dataset was divided into 100 files where each file contains 434 records. There are 12 features in the training datasets that include health information of patients. There is one feature named as stroke which represent whether a patient has stroke. This feature could be thought as the label of each entity. As for the test dataset, there is no such a feature named as stroke. Test dataset has 18,600 records in total that will be used to compare different models.



Column name	Type	Values
id	integer	Patient's id
gender	string	['Male' 'Female' 'Other']
age	double	
hypertension	integer	[0 1]
heart_disease	integer	[0 1]
ever_married	string	['No' 'Yes']
work_type	string	['children' 'Private' 'Never_worked' 'Self-employed' 'Govt_job']
Residence_type	string	['Rural' 'Urban']
avg_glucose_level	double	
bmi	double	Body Mass Index
smoking_status	string	[nan 'never smoked' 'formerly smoked' 'smokes']
stroke	integer	[0 1]

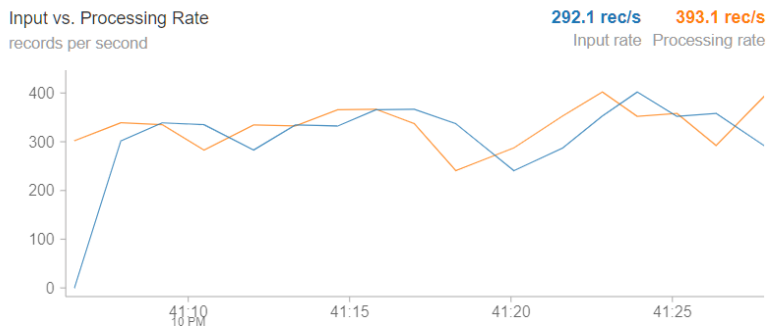
3.2. Architecture Design:

This project is mostly based on Databricks, which is a cloud-based big data service integrated applications on the top of Spark. Data files are imported into the file system of Databricks. Databricks has a file system which is similar to Hadoop. Then, Spark Streaming is used to process data in real-time. The streaming process is a structured process. Structured Streaming is a scalable and fault-tolerant stream processing engine built on the Spark SQL engine. After the data has been uploaded and streamed, Spark SQL is applied to explore data with various queries. Meanwhile, Spark MLlib is utilized to build models to predict the probability of patients having stroke based on their information. The overview of the architecture of this project is shown below. The purpose of building this architecture is to simulate the real application of predicting stroke probability in industry.



4. Streaming:

Structured streaming is a scalable and fault-tolerant stream processing engine built on the Spark SQL engine. Internally, by default, Structured Streaming queries are processed using a *micro-batch processing* engine, which processes data streams as a series of small batch jobs thereby achieving end-to-end latencies as low as 100 milliseconds and exactly-once fault-tolerance guarantees (Spark Documents, 2017). In this project, there is one file to be imported for each trigger. Then, an "append" mode has been chosen for data sink. The streaming data is stored as one table. As it can be seen from the figure below, the blue line shows input ratio of the streaming. The average rate is around 292 records per second, and it takes roughly one second to process one file. The data is loaded by this way and stored as one table in the file system. Then, it can be used for analysis either in dynamic or in static load.



5. Exploratory Data Analysis by Spark SQL

Exploratory data analysis (**EDA**) is an approach analyzing data sets to summarize their main characteristics, often with visual methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task. We wanted to get a sense of how the data look like and potentially dig out some clues to the next step. Therefore, we draw several visualization towards the data.

Twelve features are in out dataset and most of them are categorical variables. We are curious to know - if some of those variables are driving people to get stoked? In order to pulling out the key factors out of the variables, we compare those variables one by one and have the following discoveries.

Spark SQL is a Spark module for structured data processing. Unlike the basic Spark RDD API, the interfaces provided by Spark SQL provide Spark with more information about the structure of both the data and the computation being performed. Internally, Spark SQL uses this extra information to perform extra optimizations. Additionally, we can actually using regular SQL to query the data and visualize it directly. Spark SQL creates an efficient way for user to manipulate big data in a easier way.

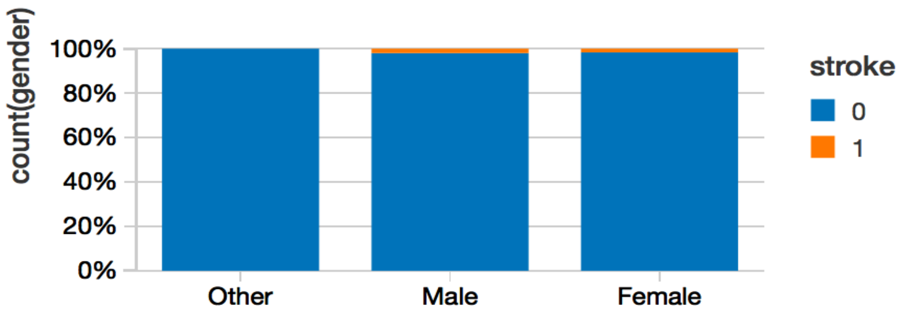
5.1. Overall distribution

In our common sense, stroked people are definitely less than non-stroked people, so we expected that the dataset will be highly imbalanced. Out of 43,000 training observations, we have only no more than 800 records are stroked. On other words, stroked records take up only 1.8%.

As we all know that, biased dataset will lead to very inaccurate model if we don't preprocess it. Hence we realized that, tactics in handling imbalanced data is very necessary in our case.

5.2. Gender distribution

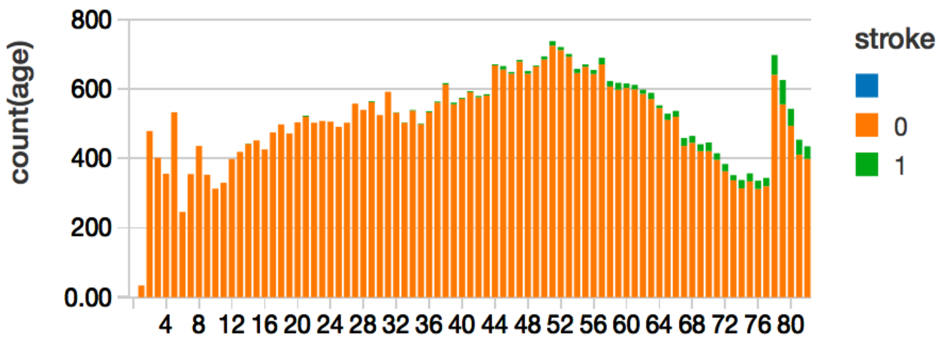
We compared the difference of gender in stroke people. There are 40% male and 60% female who took part in the research survey. Out of all positive records, 2% male and 1.7% female are stroked.



We don't see too much difference in term of gender. So theoretically gender is not a key factor deciding stroke.

5.3. Age distribution

And then we took a look at age distribution of all stroke records. As age increases, stroked records are more. At the first glance, at some point around 40 or 50, stroked bar starts to catch our eye.

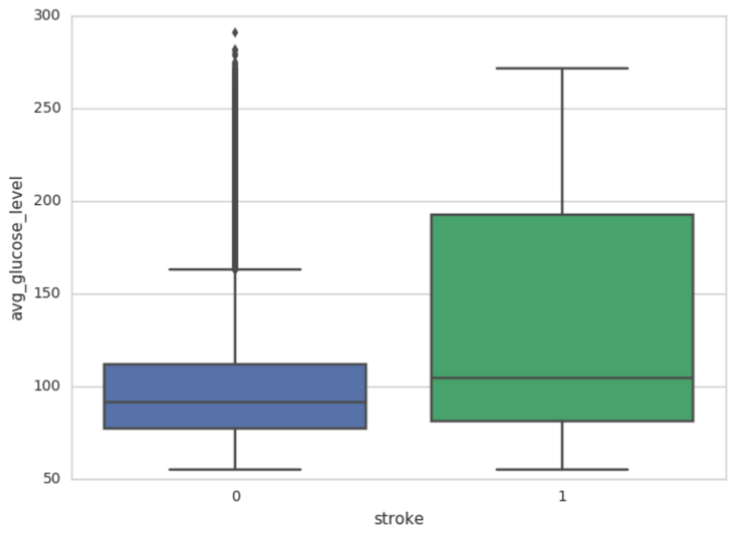


And as we dive into the age layer, we found out: 90% stroke people are aged at 50 and above. That brighten us that, if age is getting larger, is stroke more likely to happen?

5.4. Glucose level distribution

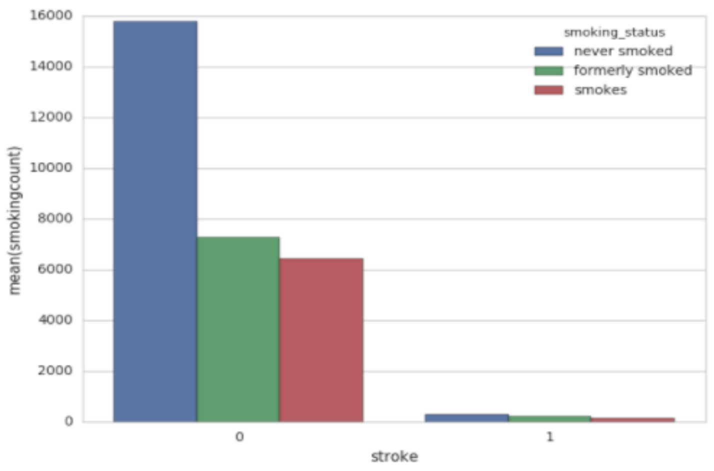
When we were looking at the boxplot of glucose level distribution between non-stroke and stroke records, the stroke box spread out but non-stroke records are more condense.

Additionally, the minimum, 1st quartile, medium, 3rd quartile and maximum of stroke records are all higher than non-stroke. This leads us to believe that, if someone is at a higher glucose level than normal, the person is risky in getting stroke.



5.5. Smoking Status

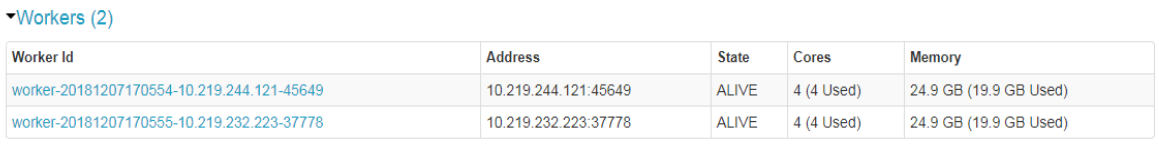
The last one we would like to call out is smoking status. The reason was that, we assumed smoking will make difference in stroke. That conform to our common sense that smoking leads to a higher possibility of getting stroke. However, the data tells a different story.



First off, we didn't see too much difference in smoking status within stoke population. Secondly, people who never smoked but get stroked are even more than people who smoked! We realized that, even some disease we commonly know actually doesn't relate too much to smoking status. Especially in this case, smoking status is the contributor of stroke.

6. Machine Learning

After capturing and processing data from real-time stream, it is the time to build a machine learning model properly to predict whether a person will possibly be diagnosed as stroke. Our project is constructed on the platform of Databricks 5.0 including Apache Spark 2.4.0 and Scala 2.11 (however, the code is complied in Python so all the libraries we applied origin from Pyspark). The cloud compuation and storage are provided by AWS standard free tier services. According to the scale of the input dataset, our project is assigned with one Master node (spark://10.219.245.32:7077) and two Worker nodes, each of them having 4 cores for parallel computation.



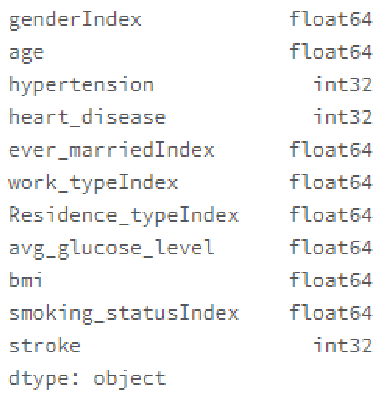
▾Workers (2)

Worker Id	Address	State	Cores	Memory
worker-20181207170554-10.219.244.121-45649	10.219.244.121:45649	ALIVE	4 (4 Used)	24.9 GB (19.9 GB Used)
worker-20181207170555-10.219.232.223-37778	10.219.232.223:37778	ALIVE	4 (4 Used)	24.9 GB (19.9 GB Used)

6.1 Prepossessing for machine learning

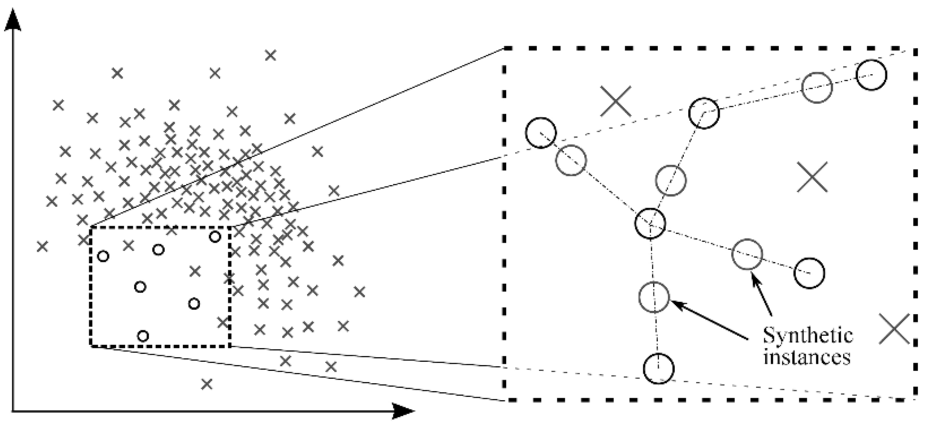
The first task is missing value imputation. We discover that "bmi" has about 3.3% values as null, while "smoking_status" has about 30.6% values as null. For the former variable, we just impute those missing values with the mean of "bmi". However, it is not appropriate to impute such a large amount of missing value in "smoking_status" by mean or median or nearest neighbors, so they are regarded as a new category named "Unknown".

Next, we need to encoding all the categorical features into numerical features for better performance of training the classification model. The Spark MLlib provides package *ml.feature* with many functions for extracting, transforming and selecting features. Here we utilize StringIndexer to transform the a string column of labels to a column of label indices, then all the data in training set becomes numerical type.



```
genderIndex           float64
age                   float64
hypertension            int32
heart_disease           int32
ever_marriedIndex     float64
work_typeIndex        float64
Residence_typeIndex   float64
avg_glucose_level     float64
bmi                   float64
smoking_statusIndex   float64
stroke                  int32
dtype: object
```

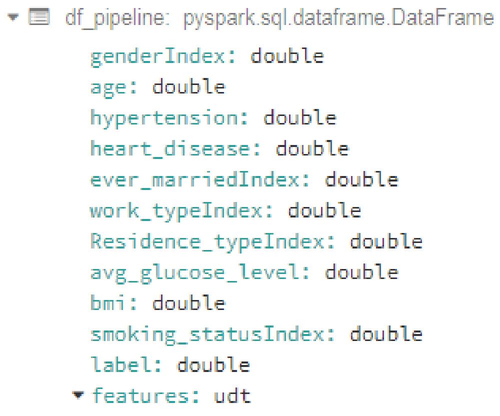
Another vital operation in the data prepossessing is resampling. As mentioned before, only 1.8% samples in the training set are labeled as “stroke” and the rest are all “not stroke”. If applying such an imbalanced dataset for training, the predictive model developed using conventional machine learning algorithms could be biased and inaccurate (for instance, classifying all samples as “not stroke” can achieve accuracy higher than 0.982). We choose to use a method called SMOTE (Synthetic Minority Over-sampling Technique) to over-sampling the minority class. Different with simply replicating samples in minority class, SMOTE generates subset of synthetic similar instances (shown in the figure below) and adds them to the original dataset, which can mitigate the problem of overfitting caused by random oversampling, and prevent a siginificant loss of useful information.



Applying SMOTE by importing *SMOTE* from imblearn.over_sampling, we transfer the original training set to a new one with 85234 rows (both "stroke" and "not stroke" have 42617 samples).

6.2 Build ML classification model

Fisrtly, we use VectorAssembler to merge features in the training set into a single vector column, and create a pipeline model to specify the workflow for machine learning. This pipeline is provided by Spark MLlib, consisting of a sequence of PipelineStages as an ordered array. After that, we split the dataset into train and test set (0.8/0.2).



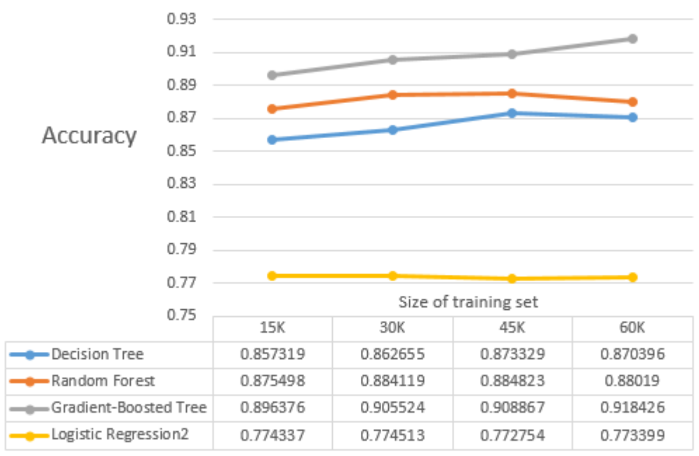
```
▾ df_pipeline:  pyspark.sql.dataframe.DataFrame
        genderIndex: double
        age: double
        hypertension: double
        heart_disease: double
        ever_marriedIndex: double
        work_typeIndex: double
        Residence_typeIndex: double
        avg_glucose_level: double
        bmi: double
        smoking_statusIndex: double
        label: double
      ▾ features: udt
```

We choose four ML algorithms for prediction: Logistic Regression and Decision Tree are simple classifiers, and Random Forest and Gradient-Boosted Tree are ensemble classifiers. The following shows important parameters of these classification models.

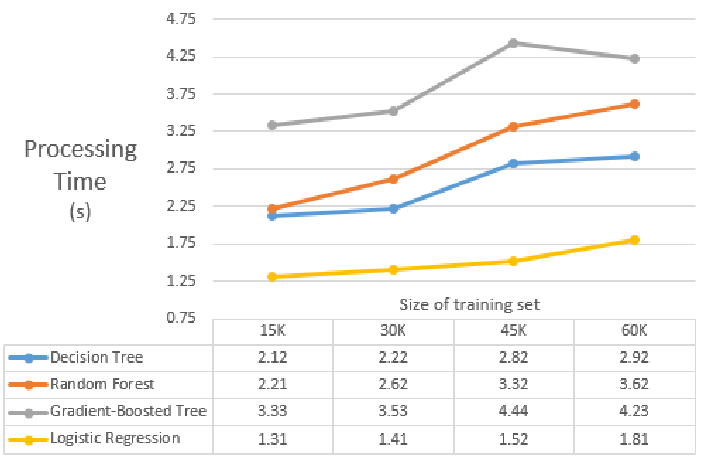
- Logistic Regression (threshold=0.5, maxIter=10)
- Decision Tree (maxDepth=5, impurity="gini")
- Random Forest (maxDepth=5, numTrees=20, subsamplingRate=1.0)
- Gradient-Boosted Tree (maxDepth=5, maxIter=5, subsamplingRate=1.0)

7. Performance Measuring

The first part is measuring the impact of dataset scale on the accuracy and processing time. We put 15k, 30k, 45k, and 60k samples from the training set to train each ML model in turn, and use *MulticlassClassificationEvaluator* from *ml.evaluation* to measure the accuracy of prediction. Meanwhile, we record the processing time to execute the classification command. The result is shown below.



	15K	30K	45K	60K
Decision Tree	0.857319	0.862655	0.873329	0.870396
Random Forest	0.875498	0.884119	0.884823	0.88019
Gradient-Boosted Tree	0.896376	0.905524	0.908867	0.918426
Logistic Regression2	0.774337	0.774513	0.772754	0.773399



	15K	30K	45K	60K
Decision Tree	2.12	2.22	2.82	2.92
Random Forest	2.21	2.62	3.32	3.62
Gradient-Boosted Tree	3.33	3.53	4.44	4.23
Logistic Regression	1.31	1.41	1.52	1.81

In these two figure, we find that logistic regression requires least processing time but perform relative poorly compared with other models. As the scale of dataset increases, the accuracy of logistic regression model does not change significantly. Decision tree classifier cannot achieve the accuracy as higher as that of random forest and gradient-boosted tree, but its accuracy increases with the expand of dataset scale, so does the processing time.

Random forest and gradient-boosted tree, as ensemble classifiers, require more processing time but perform higher prediction accuracy. As the scale of training set increases, both the accuracy and processing time increasing. The only exception is the accuracy of random forest model decreases a little bit when the train set increase from 45k to 60k, which could be casued by the overfitting.

8. Conclusion & Improvement

We summarize three conclusions from conducting this project.

1) As scale of data increasing, the accuracy of classification model also increase, so does the processing time.
2) Ensemble classifiers have better prediction than simple classifiers, although requiring longer time. If accuracy is the only vital criteria, it

would be better to choose the gradient-boosted tree algorithm. If processing time is more emphasized, random forest is a better choice.

3) If the dataset is imbalanced, where anomaly detection is crucial, a proper way of resampling is highly recommended for avoid biased and inaccurate performance.

As for further improvement, we hope to apply streaming processing and analysis via Spark Streaming, so that we can give better real-time eveluation and prediction, and modify the model in dynamic. Besides, the application in reality must have much larger volume data with much more features, we should enhance the scalability of the model and related parallel compution. In order to get better performance, we should also conduct grid search to tune the parameter in prediction models, and Introduce the ROC curve and AUC to measure the performance.

Appendiex

Please click the links to access the code. These codes were written in the form of

Databricks notebooks.

Streaming Code:

https://databricks-prod-

cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/8164427

149082084/1512320488523824/2135811732621404/latest.html

Spark SQL Code :

https://databricks-prod-

cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/8164427

149082084/346555611904370/2135811732621404/latest.html