PROJECT REPORT

TITLE : CREDIT CARD FRAUD PREDICTION USING JAVA AND MACHINE LEARNING

INTRODUCTION:

Main challenges involved in credit card fraud detection are:

1. Enormous Data is processed every day and the model build must be fast enough to respond to the scam in time.
2. Imbalanced Data i.e most of the transactions *(99.8%)* are not fraudulent which makes it really hard for detecting the fraudulent ones
3. Data availability as the data is mostly private.
4. Misclassified Data can be another major issue, as not every fraudulent transaction is caught and reported.
5. Adaptive techniques used against the model by the scammers.

How to tackle these challenges?

1. The model used must be simple and fast enough to detect the anomaly and classify it as a fraudulent transaction as quickly as possible.
2. Imbalance can be dealt with by properly using some methods which we will talk about in the next paragraph
3. For protecting the privacy of the user the dimensionality of the data can be reduced.
4. A more trustworthy source must be taken which double-check the data, at least for training the model.
5. We can make the model simple and interpretable so that when the scammer adapts to it with just some tweaks we can have a new model up and running to deploy.

Hardware/Software REQUIREMENTS:

HARDWARE:

Processor : Intel i5 CPU at 1.60 Ghz

RAM : 4GB

System type : 64 bit OS

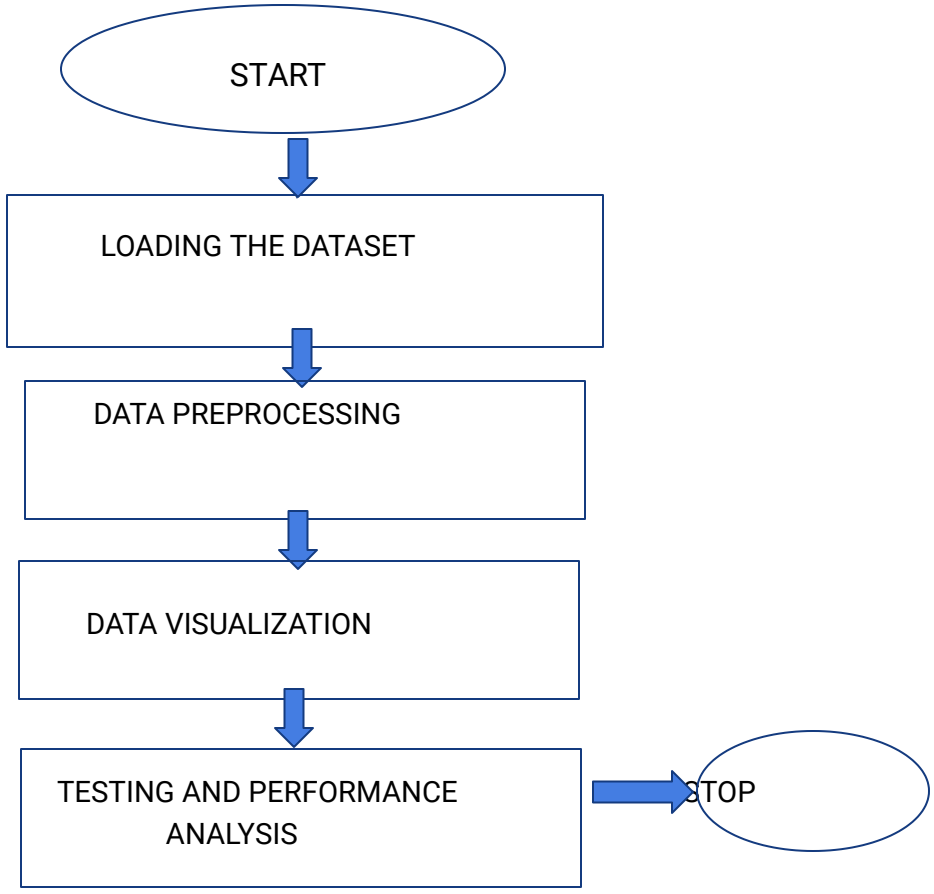
SOFTWARE:

OS : WINDOWS 7 OR ABOVE /ARCH LINUX

Programming language : Java

IDE : Eclipse

FLOW DIAGRAM:



SOURCE CODE:

```java
package org.ml;

import tech.tablesaw.api.Table;
import weka.classifiers.Evaluation;
import weka.core.Instances;
import weka.core.converters.CSVLoader;
import java.io.File;
import java.io.IOException;
import weka.classifiers.trees.RandomForest;

public class RandomForestDemo
{

    public static Instances getDataSet(String fileName) throws IOException
    {

        /**
         * we can set the file i.e., loader.setFile("filename") to load the data
         */
        int classIdx = 1;
        /** the CSVLoader to load the CSV file */
        CSVLoader loader = new CSVLoader();
        /** load the training data */
        //loader.setSource(RandomForestDemo.class.getResourceAsStream("/" + fileName));
        /**
         * we can also set the file like loader3.setFile(new
         * File("test-confused.arff"));
         */
        loader.setFile(new File(fileName));
        Instances dataSet = loader.getDataSet();
        /** set the index based on the data given in the CSV files */
        dataSet.setClassIndex(classIdx);
        return dataSet;

    }
```

```java
    /**
     * This method is used to process the input and return the statistics.
     *
     * @throws Exception
     */
    public static void main(String args[]) throws Exception
    {

        try
        {
            Table fraud_data=Table.read().csv("C:\\Users\\A. SRINIDHI\\eclipse-workspace\\org.ml\\src\\main\\java\\org\\ml\\fraud_datas
            System.out.println(fraud_data.shape());
        }
        catch(IOException e)
        {
            e.printStackTrace();
        }

        Instances trainingDataSet = getDataSet("C:\\Users\\A. SRINIDHI\\eclipse-workspace\\org.ml\\src\\main\\java\\org\\ml\\credit_fra
        Instances testingDataSet = getDataSet("C:\\Users\\A. SRINIDHI\\eclipse-workspace\\org.ml\\src\\main\\java\\org\\ml\\credit_frau

        RandomForest forest=new RandomForest();
        forest.setNumFeatures(10);
        forest.buildClassifier(trainingDataSet);
        /**
         * train the algorithm with the training data and evaluate the
         * algorithm with testing data
        */
        Evaluation eval = new Evaluation(trainingDataSet);
        eval.evaluateModel(forest, testingDataSet);

        /** Print the algorithm summary */
        System.out.println("** Random Forest Evaluation with Datasets **");
        System.out.println(eval.toSummaryString());
        System.out.print("The expression for the input data as per alogorithm is : ");
        System.out.println(forest);
    }
}
```

OUTPUT OF THE CODE:

```
827 rows X 13 cols
** Random Forest Evaluation with Datasets **

Correlation coefficient                 0.9916
Mean absolute error                     0.0056
Root mean squared error                 0.0386
Relative absolute error                 1.334  %
Root relative squared error             9.1558 %
Total Number of Instances             248

The expression for the input data as per alogorithm is : RandomForest

Bagging with 100 iterations and base learner
```

REASONS FOR USING RANDOM FOREST CLASSIFIER:

1) INCREASE THE PREDICTIVE POWER:

Firstly, there is the n_estimators hyperparameter, which is just the number of trees the algorithm builds before taking the maximum voting or taking the averages of predictions. In general, a higher number of trees increases the performance and makes the predictions more stable, but it also slows down the computation.

Another important hyperparameter is max_features, which is the maximum number of features random forest considers to split a node. Sklearn provides several options, all described in the documentation.

The last important hyperparameter is min_sample_leaf. This determines the minimum number of leafs required to split an internal node.

2) INCREASE IN MODEL SPEED

The **n_jobs** hyperparameter tells the engine how many processors it is allowed to use. If it has a value of one, it can only use one processor. A value of "-1" means that there is no limit.The **random_state** hyperparameter makes the model's output replicable. The model will always produce the same results when it has a definite value of random_state and if it has been given the same hyperparameters and the same training data.Lastly, there is the **oob_score** (also called oob sampling), which is a random forest cross-validation method. In this

sampling, about one-third of the data is not used to train the model and can be used to evaluate its performance. These samples are called the out-of-bag samples. It's very similar to the leave-one-out-cross-validation method, but almost no additional computational burden goes along with it.

CONCLUSION:

The result obtained by using random forest clasifier is around 0.991 i.e 99.1%.Hence the proposed method works well when compared to existing models based on the decision tree algorithms.

SCOPE:

A practical mobile application based on fraud alert can be a suitable prototype as for the future scope.

References:

https://www.romexsoft.com/blog/implement-credit-card-fraud-detection/

Github link:

https://github.com/smartinternz02/SPS-10809-Creditcard-Fraud-Prediction-