Udacity - Machine Learning Engineer Nanodegree - Project Capstone

Vinícius de Oliveira Silva

July 7, 2020.

Capstone Project

1. Definition

1.1 Projetc Overview:

In the financial context of credit card transactions, we are often faced with customers who are charged with paying for items that they have not actually purchased. We call this fraudulent credit card transactions. In order to avoid such a problem, it is important that companies are able to identify / recognize when a credit card transaction is a fraud or not.

The purpose of this project is to create a classifier model that is able to recognize fraudulent transactions using machine learning. The dataset used in this experiment has transactions made by credit cards in September 2013 by european cardholders.

It offers us 28 pre-generated features from the use of PCA (Principal Components Analysis), the time and amount involved in the transaction.

1.2 Problem Statement:

The dataset used in this study can be found at the link below.

Link Dataset link: https://www.kaggle.com/mlg-ulb/creditcardfraud

The question we want to answer is whether a credit card transaction is considered fraudulent or not. In other words, we have a dataset available with information on credit card transactions already labeled as fraudulent or not and, based on them, we want to train a machine learning model to identify whether new transactions are fraudulent.

In order to obtain a good predictor model, we will first train our dataset with a hold-out cross validation of 80% for training and 20% for testing in the techniques of Logistic Regression, Decision Tree, Random Forest, KneighborsClassifier, SVM with linear and rbf kernel types. After this step, we will perform a k-fold cross validation on the best model and adjust its hyperparameters.

1.3 Metrics:

The metrics we will be to evaluate our models are the **accuracy of classification** and also **Precision-Recall curves**. Once we choose the best model, we will fine-tune it by using a **grid search technique** and also a **k-fold cross validation** method to decide which are its best hyperparameters to finally have our classifier.

a) **Accuracy:** Accuracy represents the fraction of samples correctly classified among all evaluated samples. That is, the number of true positive and true negative samples among all samples.

$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

b) **Precision and Recall:** Precision represents the ratio between the number of positive samples correctly classified (true positive) and the sum of the amount of true positive rating and false positive rating. Recall recall represents the ratio between the number of positive samples correctly classified (true positive) and the sum of the amount of true positive rating and false negative rating.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

c) **Precision-Recall Curve:** It is a curve with the Precision and Recall metrics that are evaluated, as we modify the threshold of classification of the output of our model. In the case of this work, as we want to avoid false negatives as much as possible (it is worse to classify as a non-fraudulent transaction when it is a fraud than to classify it as fraudulent when it is not a fraud), we will pay attention to both the area under the curve and the metric of Recall. See below Precision-Recall curve example taken from the site https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/.

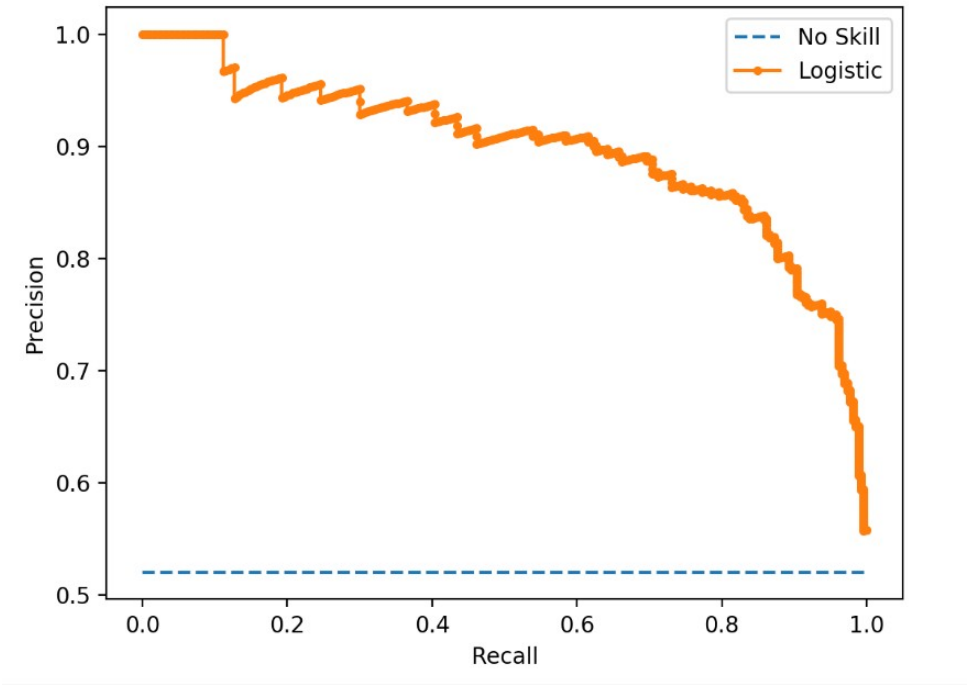

Figure 1 – Precision-Recall Curve example.

2. Analysis

2.1 Data Exploration:

As mentioned earlier, the dataset used in this project can be easily downloaded from the Kaggle machine learning competition website below:
link: https://www.kaggle.com/mlg-ulb/creditcardfraud

There we will find a ***creditcard.csv*** file containing credit card transactions from European customers collected in the September 2013 period.

The dataset has transactions referring to 2 days of which we have 492 fraud examples out of 284807 transactions, being, therefore, an unbalanced dataset. Fraudulent transactions represent only 0.172% of total transactions. See Figure 2 below with its corresponding dataframe.

The aforementioned .csv file presents only numeric variables that are the results of applying PCA (Principal Components Analysis) on the original features for reasons of confidentiality.

We have 28 features generated from the PCA that are called V1, V2, ..., V28. Only two of the original features have not been transformed by the PCA: Time and Amount.

“Time”: Time elapsed in seconds between each transaction and the first transaction in the dataset.

“Amount”: Corresponds to the Amount of the transaction.

“Class”: Represents the class to which each transaction belongs. It presents a value of 1 when it is a fraud and a value of 0 when it is not a fraud.

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267

5 rows × 31 columns

Figure 2 – Dataset Dataframe.

As shown in Figure 3 below, we present no missing value in our dataset, basically with numerical data.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
Time      284807 non-null float64
V1        284807 non-null float64
V2        284807 non-null float64
V3        284807 non-null float64
V4        284807 non-null float64
V5        284807 non-null float64
V6        284807 non-null float64
V7        284807 non-null float64
V8        284807 non-null float64
V9        284807 non-null float64
V10       284807 non-null float64
V11       284807 non-null float64
V12       284807 non-null float64
V13       284807 non-null float64
V14       284807 non-null float64
V15       284807 non-null float64
V16       284807 non-null float64
V17       284807 non-null float64
V18       284807 non-null float64
V19       284807 non-null float64
V20       284807 non-null float64
V21       284807 non-null float64
V22       284807 non-null float64
V23       284807 non-null float64
V24       284807 non-null float64
V25       284807 non-null float64
V26       284807 non-null float64
V27       284807 non-null float64
V28       284807 non-null float64
Amount    284807 non-null float64
Class     284807 non-null int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

Figure 3 – Dataframe information.

Below we find the dataset statistics that we calculated using the pandas *describe()* method:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	1.165980e-15	3.416908e-16	-1.373150e-15	2.086869e-15	9.604066e-16	1.490107e-15	-5.556467e-16	1.177556e-16
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01

8 rows × 31 columns

Figure 4 – Dataframe description.

2.2 Exploratory Visualization:

As already mentioned in the Data Exploration session, the ***creditcard.csv*** dataset is unbalanced. To check the number of samples for each class, we will plot a pie chart. See Figure 5 below:

	label	count
0	0	284315
1	1	492

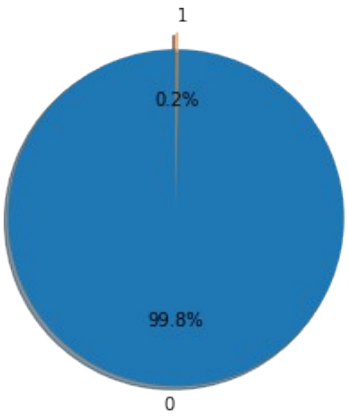

Figure 5 – Pie chart of classes and dataframe with counted labels.

As we can see, there is a big imbalance between the two classes. With approximately 0.2% of those sampled belonging to the fraudulent transaction class.

In order to check which features we will use in the classification process, we calculate the correlation matrix between all features. With that, we are able to decide to remove some feature that eventually is strongly correlated with another and, in this way, we will reduce the entry space of our model. Looking at the correlation matrix in Figure 6

below, we have identified that all features are quite uncorrelated, so we will not discard any features from our model.

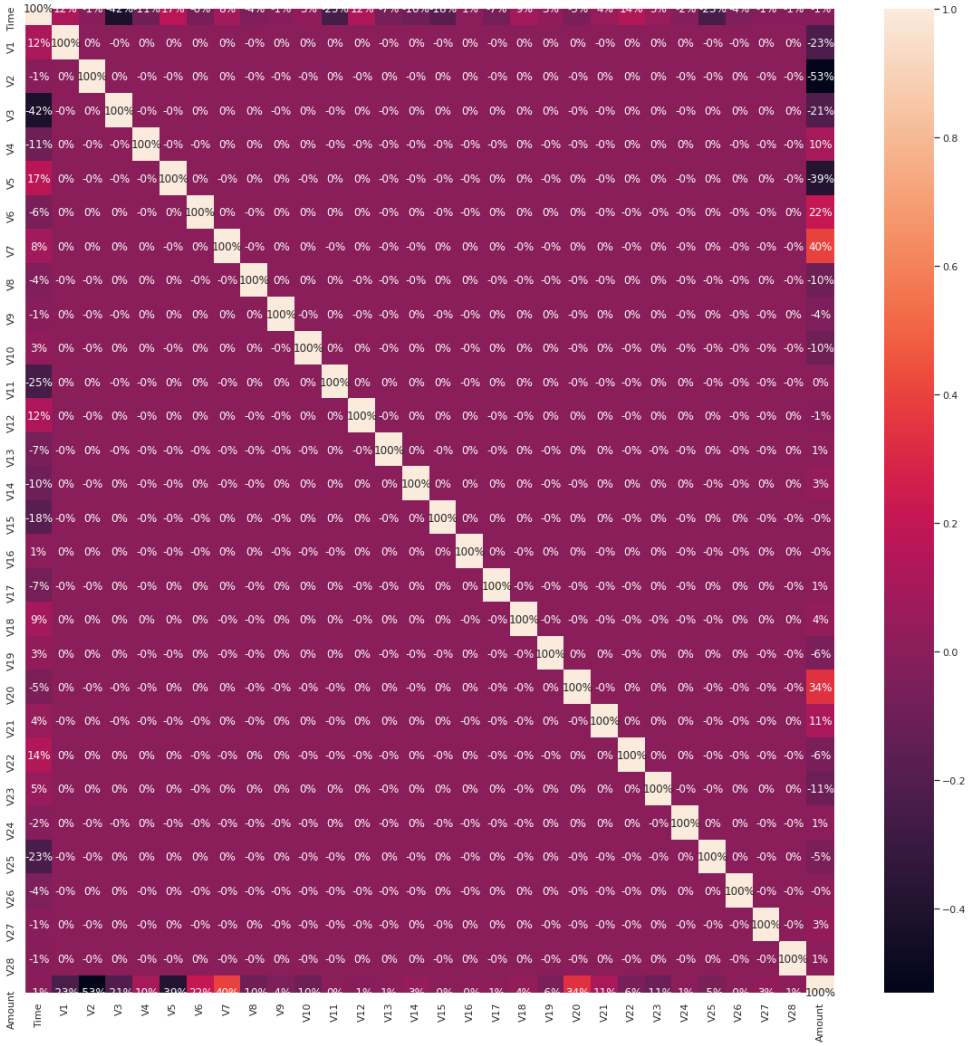

Figure 6 – Correlation Matrix of features.

2.3 Algorithms and Techniques:

The framework used for modeling this project is Scikit-learn (its documentation can be accessed through the link https://scikit-learn.org/stable/. With it, we can do both statistical analysis and machine learning modeling and data pre-processing.

To train our credit card transaction fraud detection model, we will train a 80% hold-out for training and 20% for testing in the Logistic Regression, Decision Tree, Random Forest, KneighborsClassifier, SVM with linear and rbf kernel algorithms types. After selecting the best accuracy and recall result among the aforementioned models, we will perform a k-fold cross validation training with k=5, using grid-search to tune the hyperparameters of the chosen model and evaluate its accuracy and its Precision-Recall Curve.

KNN algorithm (https://scikit-learn.org/stable/modules/neighbors.html) is a supervised learning technique based on on neighbors. It can be used for solving

classification or regression problems. In this porjetect we used that for classification. The target is to find a predefined number of training samples closest distance (usuallu Euclidian distance is used) to the point we wanto to classify.

Decision Tree is a supervised non-parametric method of machine learning that can solve both classification and regression problems. Its objective is to build a model capable of making a prediction by learning decision rules that are inferred based on the features of the dataset. We can find its documentation on scikit-learn at the link https://scikit-learn.org/stable/modules/tree.html.

Random Forest is an estimator in which several decision tree classifiers are trained in various subsamples of the dataset, using the average of these trainings to improve predictive accuracy and control overfitting.

Random Forest documentation on scikit-leran can be found at link https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

Logistic Regression is a technique used for binary classification with input values that are combined linearly using weights or coefficient values to predict an output value. The main difference of this method for linear regression is that the output value being modeled is a discrete value (0 or 1) instead of a numerical value. See reference in link https://machinelearningmastery.com/logistic-regression-for-machine-learning/. Its documentation in the scikit-learn framework can be found at the link https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

SVM (Support Vector Machines) represent a set of supervised methods for classification, regression and even for detecting outliers. In this project we use this technique for classification and as our benckmark method for comparison, since it is a very effective algorithm when dealing with high dimensional data, it is also an efficient method in terms of memory consumption and can use different kernel functions. We can find the reference to SVM method in the scikit-learn library at the link https://scikit-learn.org/stable/modules/svm.html.

2.4 Benchmark

we first trained 6 different models with their default parameters and from the **Precision-Recall curve** for each training, we decided to refine the training **of the Random Forest algorithm**, in order to find the best hyperparameter configuration of it. As a result, our Benchmark model will be **SVM (Support Vector Machines)**, widely used to solve multidimensional data problems (as is the case) and which has already shown good results for this sort of classification task.

3. Methodology

3.1 Data Preprocessing:

Since some Machine Learning algorithms are variants to scale, it would be interesting to normalize the numerical variables of the dataset so that it has a mean of zero and unit variance and, thus, all the input data will be on the same scale.

The documentation of this method of normalization can be found at the link **https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.**

The equation of this sort of normalization follows the equation below:

$$x_{scale} = \frac{X - mean_X}{std_X}$$

In this equation X represents our dataset, $mean_X$ is the feature wise mean trough the dataset and std_X is the feature wise standard deviation through all data points.

3.2 Implementation:

To train the credit card transaction classification model, we initially applied a hold-out cross validation of 80% training / 20% testing samples on Logistic Regression, Decision Tree, Random Forest, KneighborsClassifier, SVM with linear and rbf kernel algorithms types and their default hyperparameters.

- Here we apply a cross-validation hold-out using 20% of the data for testing the model and 80% for effective training:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

```
print(X_train.shape, '-->', y_train.shape)
print(X_test.shape, '-->', y_test.shape)
```

```
(227845, 30) --> (227845,)
(56962, 30) --> (56962,)
```

- Now we scale features:

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

We analyzed both accuracy and recall, as having false negatives in this problem would be the worst case.

The results of accuracy and recall are shown in Figure 7 below.

```
0 - Logistic Regression:    0.9992012113498212 of accuracy and 0.6163682864450127 of recall
1 - Decision Tree:          1.0 of accuracy and 1.0 of recall
2 - Random Forest:          0.9999166099760802 of accuracy and 0.9514066496163683 of recall
3 - KNN:                    0.9995918277776559 of accuracy and 0.8005115089514067 of recall
4 - SVM with linear kernel: 0.9993811582435428 of accuracy and 0.7877237851662404 of recall
5 - SVM with rbf kernel:    0.9996532730584389 of accuracy and 0.8132992327365729 of recall
```

Figure 7 – Hold-out results.

Analyzing the Precision-Recall curves of Figures 8, 9, 10, 11, 12 and 13, we observe that the curves with a tendency for a higher Recall and also a high precision are in the Decision Tree, Random Forest and SVM models with rbf kernel.

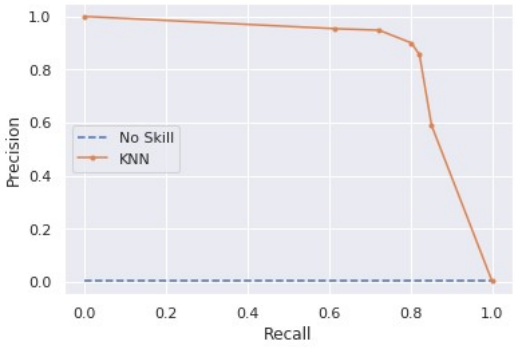

Figure 8 – Precision-Recall for KNN.

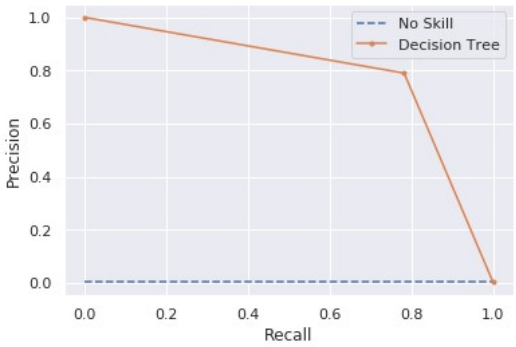

Figure 9 – Precision-Recall for Decision Tree.

Figure 10 – Precision-Recall for Logistic Regression.

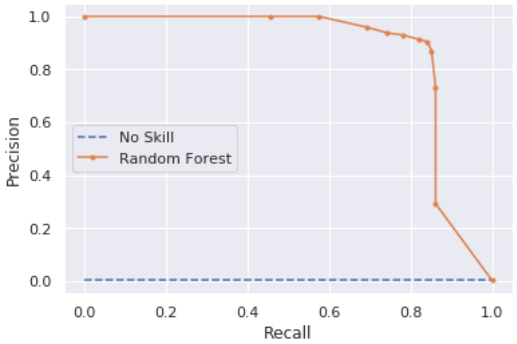

Figure 11 – Precision-Recall for Random Forest.

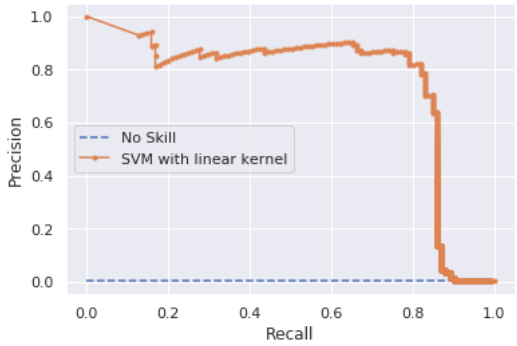

Figure 12 – Precision-Recall for SVM kernel linear.

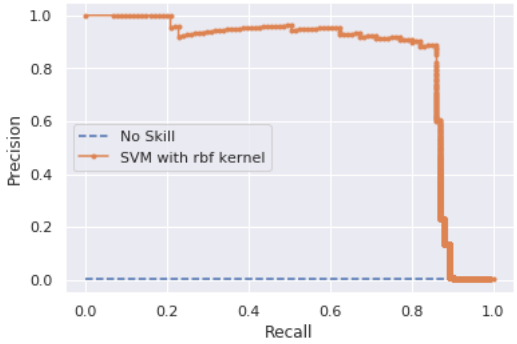

Figure 13 – Precision-Recall for SVM kernel RBF.

Since the two best methods in terms of accuracy and recall were the Decision Tree and Random Forest models and, knowing that Random Forest is effective in combating overfitting, we decided to use it for refinement.

3.3 Refinement:

Once we decided to adjust the Random Forest model, we used **GridSearchCV** model selection tool from scikit-learn to make our model more robust. We used a 5-fold cross-validation on training dataset and a grid search on **'classifier__n_estimators'**, **'classifier__max_depth'** and **'classifier__criterion'** hyperparameters of Random Forest were tuned. See code below to understand the pipeline used for training.

```
import pickle

from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.metrics import accuracy_score, recall_score, confusion_matrix, classification_report
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score, auc
```

```
grid_steps = [('scaller', StandardScaler()),
        ('classifier', RandomForestClassifier())]
```

```
pipeline = Pipeline(grid_steps)
```

```
param_grid = {
    'classifier__n_estimators': [200, 300, 500, 600],
    'classifier__max_depth': [2, 3, 5],
    'classifier__criterion': ['gini', 'entropy']
}
```

Grid search was implemented as seen in code below and the best hyperparameters found are also presented (training criterion based on entropy, with depth 5 and 300 estimators).

```
search = GridSearchCV(pipeline, param_grid, n_jobs=-1, scoring='recall') # by defaul it creates a 5-fold cross-vali
search.fit(X_train, y_train)
```

```
GridSearchCV(estimator=Pipeline(steps=[('scaller', StandardScaler()),
                                       ('classifier',
                                        RandomForestClassifier())]),
             n_jobs=-1,
             param_grid={'classifier__criterion': ['gini', 'entropy'],
                         'classifier__max_depth': [2, 3, 5],
                         'classifier__n_estimators': [200, 300, 500, 600]},
             scoring='recall')
```

```
search.best_params_
```

```
{'classifier__criterion': 'entropy',
 'classifier__max_depth': 5,
 'classifier__n_estimators': 300}
```

The Precision-Recall Curve of this training process is shown below in Figure 14:

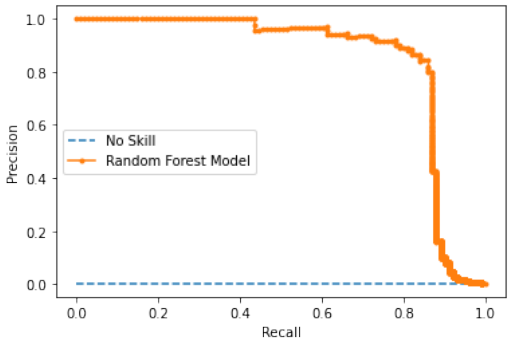

Figure 14 – Precision-Recall for 5-fold CV on Random Forest.

The accuracy and AUC (area under the curve) of the model obtained with the 5-fold cross-validation and the grid search were **99.95%** and **0.85**, respectively.

4. Results

4.1 Model Evaluation and Validation:

We initially trained 6 different classifiers with their default settings in scikit-learn. We decided from the Precision-Recall curves and accuracy the best among them for refinement. The selected model was the Random Forest which, when trained using a 5-fold cross-validation and a grid search to choose the best hyperparameters, obtained an accuracy of 99.95% and an AUC of 0.85 on the Precision-Recall curve.

The best hyperparameters found for Random Forest were:

```
{'classifier__criterion': 'entropy',
 'classifier__max_depth': 5,
 'classifier__n_estimators': 300}
```

In order to compare this model with our benchmark method (SVM) we also trained an SVM with rbf kernel (once it was the best bewteen linear and rbf kernel in the first part of the experiment) using a 5-fold cross-validation and applying a grid search on **‘C’** and **‘gamma’** hyperparameters as well.

4.2 Justification:

The Precision-Recall Curve of SVM with kernel rbf is shown below in Figure 15:

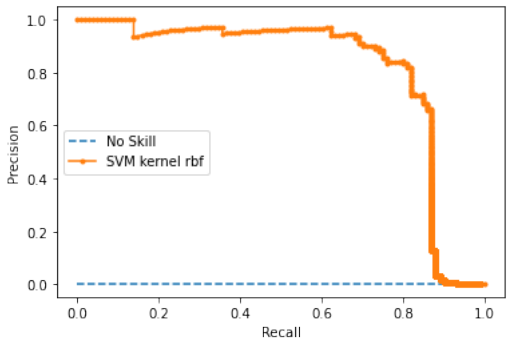

Figure 15 – Precision-Recall for 5-fold CV on SVM-rbf.

The accuracy and AUC metrics for this SVM training with rbf kernel were **99.94%** and **0.818**, respectively, indicating that it was a good model too, but Random Forest proved to be a little better. And we can also check **f1-score** as a measure of the balance between precision and recall. See metrics in dataframe below.

	Random Forest	SVM-rbf
acc	99.95%	99.94%
AUC	0.85	0.818
f1-score	0.832	0.816

Figure 16 – Metric comparison dataframe between Random Forest and SVM-rbf.

By observing that dataframe we can be sure that Random Forest obtained more robust results on credit card transactions dataset. Then we can use it to classify transactions between fraud or not with low False Negative Rating.