In [1]: import numpy as np import pandas as pd import sklearn import scipy import matplotlib.pyplot as plt import seaborn as sns from sklearn.metrics import classification_report,accuracy_score from sklearn.neighbors import KNeighborsClassifier from sklearn.svm import OneClassSVM import matplotlib.pyplot as plt from sklearn.metrics import roc_curve, auc %matplotlib inline from sklearn.metrics import confusion_matrix from sklearn.metrics import accuracy_score from sklearn.svm import SVC SVC(probability=**True**) from sklearn.svm import SVR from sklearn.naive_bayes import GaussianNB from sklearn.tree import DecisionTreeClassifier import math from pylab import rcParams rcParams['figure.figsize'] = 14, 8 $RANDOM_SEED = 42$ LABELS = ["Normal", "Fraud"] from sklearn.preprocessing import StandardScaler from sklearn.model_selection import train_test_split dataset = pd.read_csv('creditcard.csv') In [2]: In [3]: dataset.head() Out[3]: Time V1 V2 **V3** V4 **V5** V6 **V7 V8** 0.462388 0.239599 0.0 -1.359807 -0.072781 2.536347 1.378155 -0.338321 0.098698 0.3 0.0 1.191857 0.266151 0.166480 0.448154 0.060018 -0.082361 -0.078803 0.085102 -0.2 1 0.791461 1.0 -1.358354 -1.340163 1.773209 0.379780 -0.503198 1.800499 0.247676 -1.5 1.247203 1.0 -0.966272 -0.185226 1.792993 -0.863291 -0.010309 0.237609 0.377436 -1.3 0.592941 -0.270533 0.8 5 rows × 31 columns In [4]: dataset.info() <class 'pandas.core.frame.DataFrame'> RangeIndex: 284807 entries, 0 to 284806 Data columns (total 31 columns): Column Non-Null Count Dtype # -----0 Time 284807 non-null float64 V1 1 284807 non-null float64 2 V2 284807 non-null float64 3 ٧3 284807 non-null float64 4 V4 284807 non-null float64 284807 non-null float64 5 V5 6 ٧6 284807 non-null float64 7 ٧7 284807 non-null float64 V8 284807 non-null float64 8 9 V9 284807 non-null float64 10 V10 284807 non-null float64 284807 non-null float64 11 V11 284807 non-null float64 12 V12 13 V13 284807 non-null float64 284807 non-null float64 14 V14 15 V15 284807 non-null float64 16 V16 284807 non-null float64 284807 non-null float64 17 V17 V18 284807 non-null float64 18 284807 non-null float64 19 V19 20 V20 284807 non-null float64 V21 284807 non-null float64 21 284807 non-null float64 22 V22 284807 non-null float64 23 V23 V24 284807 non-null float64 24 284807 non-null float64 25 V25 26 V26 284807 non-null float64 284807 non-null float64 27 V27 28 284807 non-null float64 V28 Amount 284807 non-null float64 29 Class 284807 non-null int64 30 dtypes: float64(30), int64(1)memory usage: 67.4 MB In [5]: dataset.isnull().values.any() Out[5]: False In [6]: set_class = pd.value_counts(dataset['Class'], sort = True) set_class.plot(kind = 'bar', rot=0) plt.title("Class Distribution of Transaction") plt.xticks(range(2), LABELS) plt.xlabel("Classes") plt.ylabel("No of occurences") Out[6]: Text(0, 0.5, 'No of occurences') Class Distribution of Transaction 250000 150000 100000 50000 Classes In [7]: fraud_class = dataset[dataset['Class']==1] normal_class = dataset[dataset['Class']==0] outlier_fraction = len(fraud_class)/float(len(normal_class)) In [8]: | print(fraud_class.shape, normal_class.shape) (492, 31) (284315, 31) #undersampled_data = pd.concat([normal_class.sample(frac = (len(fraud_cl ass)/len(normal_class))), fraud_class.sample(frac=1)],axis=0) #x = pd.DataFrame(undersampled_data.iloc[:,undersampled_data.columns!='C lass']) #y = undersampled_data.iloc[:,undersampled_data.columns == 'Class'] x = dataset.iloc[: , 1:30].valuesy = dataset.iloc[:, 30].values In [10]: #from sklearn.preprocessing import StandardScaler #sc = StandardScaler() #x["scaled_Amount"]= sc.fit_transform(x.iloc[:,29].values.reshape(-1, 1)) #'''Dropping Time and Old amount''' #x = x.drop(["Time", "Amount"], axis= 1)print("Input Range : ", x.shape) print("Output Range : ", y.shape) Input Range: (284807, 29) Output Range: (284807,) In [11]: fraud_class.Amount.describe() Out[11]: count 492.000000 mean 122.211321 256.683288 std min 0.000000 25% 1.000000 50% 9.250000 75% 105.890000 2125.870000 max Name: Amount, dtype: float64 In [12]: normal_class.Amount.describe() Out[12]: count 284315.000000 mean 88.291022 250.105092 std min 0.000000 25% 5.650000 50% 22.000000 75% 77.050000 25691.160000 max Name: Amount, dtype: float64 In [13]: ## Correlation import seaborn as sns #get correlations of each features in dataset corrmat = dataset.corr() top_corr_features = corrmat.index plt.figure(figsize=(20,20)) #plot heat map g=sns.heatmap(dataset[top_corr_features].corr(),annot=**True**,cmap="RdYlGn" Time - 1 012-0.012-0.012-0.012-0.012-0.012-0.012-0.012-0.012-0.013-0.022-0.013-0.022-0.030.0085-0.030.0085-0.030.0085-0.030.0085-0.030.0085-0.030.0085-0.008 V1 - 0.12 1 47e-137.4e-139e-137.4e-137.4e-139.5e-327.e-137.4e-137 - 0.8 V5 - 0.176 4e-172e-18.4e-259e-1 1 9e-145 2e-1756e-1753e-1753e-1756e-1756e-1756e-1756e-1751 V6 -0.062.4e-16e-16.4e-15.7e-169e-1 1 4e-16.7e-161e-159e-169e-151e-15.3e-169e-164e-155e-164e-155e-169e-159e-166e-159e-166e-159e-166e-159e-166e-159e-160-22 0.044 V7 - 0.085 2e-1.54e-162.2e-1.56e-146 2e-1.64e-1 1 3, 7e-1/79e-1.62e-171.1e-1.5e-1.99e-12/7e-1.69e-12/79e-1.69e-12/79e-1.69e-12/79e-1.69e-12/79e-1.69e-12/79e-1.69e-12/79e-1.69e-12/79e-1.69e-12/79e-1.69e-12/79e-1.69e-12/79e-1.69e-12/79e-1.69e-12/79e-1.69e-12/79e-1.69e-12/79e-1.69e-12/79e-1.69e-1 0.6 V8 - 0.039.5e-1474e-1574e-1552e-156e-157e-156e-157e-156e-157e-145e-16.3e-172e-16.3e-172e-16.3e-172e-16.1e-162e-165e-166-168.5e-1663e-1551e-1553e-1551e-1558e-1558e-1558e-1554e-1552e-1564e-1552e-1569e-1568e-1564e-1552e-1569e-1568e-1564e-1552e-1569e-1568e-1568e-1564e-1562e-1569e-1568e-1564e-1562e-1569e-1568e-1564e-1562e-1569e-1568e-1564e-1562e-1569e-1568e-1564e-1562e-1569e-1568e-1564e-1562e-1569e-1568e-1564e-1562e-1569e-1568e-1564e-1562e-1569e-1568e V9 -0.008272c-15.7e-1472c-189e-1452c-1151c-1159c-1259c-1254c vio -0.0317.4e-147.8e-1463e-1461e-146.6e-1469e-146e-1-139 le-127.8e-1 1 2.6e-1464e-148.9e-146e-145.7e-157e-158e-142.7e-137-14e-151e-146.7e-158e-146-1-12-9e-156e-151e-155e-160.1 0 2 V12 - 0.122 4e-16 6e-162e-167e-168e-161e-165e-16.3e-124e-154e-152e-15 1 2.3e-148e-155e-155e-159e-165e-123e-152e-159e-164e-169e-151e-158e-153e-163e-163e-16009).183.3e-1288e-1261e-1265e-2661e-2664e-1189e-172e-161.6e-1256e-1468e-1269e-1467e-242e-172 1 1.3e-1352e-1255e-266e-1262e-1269e-136,9e-161e-1466e-1269e-1368e-1251e-129.0030,0042 V16 -0.0125.3e-1469e-1272e-145.9e-1455e-1255e-1256e-146e-146.3e-1257e-1452e-1855e-1259e-1273e-127. V17 -0.0735e-19.9e-186e-14.4e-186e-181e-18.5e-186e-18.7e-18.7e-18.7e-18.9e-182e-186e-132e-189e-15.6e-189e-186e-186e-186e-186e-186e-186e-18.2e-186073-0.33 V18 - 0.092 9e-1256e-1254e-1255e-151e-1258e-151e-1451e-146e-146e-146e-146-9e-162e-19e-177.5e-19e-15.6e-1_0 1 2.4e-1459e-151e-1257e-126e-151e-1253e-152e-1254e-1258e-1254e-1258e-1254e-1258e-1254e-1258e-1254e-1258e-1254e-1258e-1254e-1258e-1254e-1258e-1254e-1258e-1254e-1258e-1254e-1258e-1254e-1258e-1254e-1258e-1254 V19 -0.029.8e-18.5e-1276e-1267e-1261e-1267e-126.9e-1263e-126e-1262e-1263e-127.6e-1261e-1266e-16e-153.9e-1264e-151 2.9e-164e-169.7e-167e-161e-1274e-1266e-161e-1264e-190.0560.035 -0.2 V24 -0.016.4e-18.1e-18.4e-17.2e-18.3e-1256e-17.8e-1253e-146e-17.2e-18.9e-18.5e-186e-18.5e-18.5e-17.1e-18.1e-17.6e-18.3e-18.2e-14.4e-17.1 1.6e-18.1e-18.7e-1253e-18600510.0072 V25 - 0.23 - 8.8-163e-1878e-1164e-186e-1161e-1152e-15.4e-1161e-1359e-1466e-1861e-1356e-1361e-136e-1468e-1353e-1364e-1356e-1361e-1359e-1466e-1661e-1361 V27 -0.00512e-145.5e-1652e-1654e-165e-1556e-1559e-167e-153e-1516e-1556e-1559e-167e-1558e-1568e-1558e-1568e-1558e-1568e-1568e-1568e-1568e-1568e-1568e-1568e-1568e-1568e-1568e-1568e-1568e-1 Class -0.012 -0.1 0.091 -0.19 0.13 -0.0950.044 -0.19 0.02 -0.098 -0.22 0.15 -0.260.0046 -0.3 0.0042 -0.2 -0.33 -0.11 0.035 0.02 -0.040.00081.0020.0078.0038.00450.0180.0099.0056 1 In [14]: | print(x,y) [[-1.35980713e+00 -7.27811733e-02 2.53634674e+00 ... 1.33558377e-01 -2.10530535e-02 1.49620000e+02] [1.19185711e+00 2.66150712e-01 1.66480113e-01 ... -8.98309914e-03 1.47241692e-02 2.69000000e+00] [-1.35835406e+00 -1.34016307e+00 1.77320934e+00 ... -5.53527940e-02-5.97518406e-02 3.78660000e+02] [1.91956501e+00 -3.01253846e-01 -3.24963981e+00 ... 4.45477214e-03 -2.65608286e-02 6.78800000e+01] $[-2.40440050e-01 \quad 5.30482513e-01 \quad 7.02510230e-01 \quad \dots \quad 1.08820735e-01$ 1.04532821e-01 1.00000000e+01] [-5.33412522e-01 -1.89733337e-01 7.03337367e-01 ... -2.41530880e-031.36489143e-02 2.17000000e+02]] [0 0 0 ... 0 0 0] In [15]: xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.20, ra $ndom_state = 42)$ In [16]: print("xtrain.shape : ", xtrain.shape)
 print("xtest.shape : ", xtest.shape) print("ytrain.shape : ", ytrain.shape) print("ytest.shape : ", ytest.shape) xtrain.shape : (227845, 29) xtest.shape : (56962, 29) ytrain.shape : (227845,) ytest.shape : (56962,) In [17]: stdsc = StandardScaler() xtrain = stdsc.fit_transform(xtrain) xtest = stdsc.transform(xtest) In [18]: | print("Training Set after Standardised : \n", xtrain[0]) Training Set after Standardised : [0.99785119 - 0.22962626 - 0.20738468 0.23421529 - 0.36779128 - 0.0640219]-0.50588871 0.03060393 0.95995457 -0.02686352 0.61149957 1.689990331.26149805 -0.26397318 -0.36292946 0.34942719 -1.09376318 0.77802759 $0.20615616 \ -0.1625072 \qquad 0.32483903 \quad 1.3366986 \qquad 0.08456891 \ -0.45980186$ -0.08436785 -0.44894226 0.11248883 -0.14374055 -0.30788875] In [19]: knn = KNeighborsClassifier(n_neighbors = 5) knn.fit(xtrain, ytrain) Out[19]: KNeighborsClassifier() In [20]: |y_pred = knn.predict(xtest) In [21]: | cm1 = confusion_matrix(ytest,y_pred) print("Confusion Matrix : \n\n", cm1) Confusion Matrix: [[56859 5] 20 78]] In [22]: **from sklearn.metrics import** f1_score print("The accuracy is "+str((cm1[1,1]+cm1[0,0])/(cm1[0,0] + cm1[0,1]+cm 1[1,0] + cm1[1,1])*100) + "%")print("The recall is "+ str(cm1[1,1]/(cm1[1,0] + cm1[1,1])*100) +" %") print("The precision is "+ str(cm1[1,1]/(cm1[0,1] + cm1[1,1])*100) +" %" print('F1 score:', f1_score(ytest,y_pred)) The accuracy is 99.95611109160492 % The recall is 79.59183673469387 % The precision is 93.97590361445783 % F1 score: 0.8618784530386742 In [23]: | from sklearn.metrics import roc_auc_score k_probs = knn.predict_proba(xtest) k_probs = k_probs[:, 1] auc = roc_auc_score(ytest, k_probs) print(': %.3f' % auc) AUC: 0.939 In [24]: $Accuracy_Model = ((cm1[0][0] + cm1[1][1]) / cm1.sum()) *100$ print("Accuracy_knn : ", Accuracy_Model) $Error_rate_Model = ((cm1[0][1] + cm1[1][0]) / cm1.sum()) *100$ print("Error_rate_knn : ", Error_rate_Model) # True Fake Rate Specificity_Model= (cm1[1][1] / (cm1[1][1] + cm1[0][1])) *100print("Specificity_knn : ", Specificity_Model) # True Genuine Rate Sensitivity_Model= (cm1[0][0] / (cm1[0][0] + cm1[1][0])) *100print("Sensitivity_knn : ", Sensitivity_Model) : 99.95611109160492 Accuracy_knn Error_rate_knn : 0.0438889083950704 Specificity_knn : 93.97590361445783 Sensitivity_knn : 99.9648376377925 In [25]: **from sklearn.calibration import** CalibratedClassifierCV svm = SVC(C= 10, kernel= 'rbf', random_state= 0) svc_classifier = CalibratedClassifierCV(svm) svc_classifier.fit(xtrain, ytrain) Out[25]: CalibratedClassifierCV(base_estimator=SVC(C=10, random_state=0)) In [26]: y_pred2 = svc_classifier.predict(xtest) In [27]: cm = confusion_matrix(ytest, y_pred2) print("Confusion Matrix : \n\n", cm) Confusion Matrix : 30 68]] In [28]: from sklearn.metrics import f1_score print("The accuracy is "+str((cm[1,1]+cm[0,0])/(cm[0,0] + cm[0,1]+cm[1,0])] + cm[1,1])*100) + " %") print("The recall is "+ str(cm[1,1]/(cm[1,0] + cm[1,1])*100) +" %") print("The precision is "+ str(cm[1,1]/(cm[0,1] + cm[1,1])*100) +" %") print('F1 score:', f1_score(ytest,y_pred2)) The accuracy is 99.9420666409185 % The recall is 69.38775510204081 % The precision is 95.77464788732394 % F1 score: 0.8047337278106508 In [29]: | from sklearn.metrics import roc_auc_score svc_probs = svc_classifier.predict_proba(xtest) svc_probs = svc_probs[:, 1] auc = roc_auc_score(ytest,svc_probs) print('AUC: %.3f' % auc) AUC: 0.972 In [31]: $|Error_rate_Model| = ((cm[0][1] + cm[1][0]) / cm.sum()) *100$ print("Error_rate_svc : ", Error_rate_Model) # True Fake Rate Specificity_Model= (cm[1][1] / (cm[1][1] + cm[0][1])) *100print("Specificity_svc : ", Specificity_Model) # True Genuine Rate Sensitivity_Model= (cm[0][0] / (cm[0][0] + cm[1][0])) *100print("Sensitivity_svc : ", Sensitivity_Model) Error_rate_svc : 0.05793335908149292 Specificity_svc : 95.77464788732394 Sensitivity_svc : 99.94726758186708 In [32]: dt_classifier = DecisionTreeClassifier(criterion = 'entropy', random_sta te = 0) dt_classifier.fit(xtrain, ytrain) Out[32]: DecisionTreeClassifier(criterion='entropy', random_state=0) In [33]: y_pred3 = dt_classifier.predict(xtest) In [34]: cm3 = confusion_matrix(ytest, y_pred3) print("Confusion Matrix : \n\n", cm3) Confusion Matrix : [[56842 22] [20 78]] In [35]: **from sklearn.metrics import** f1_score print("The accuracy is "+str((cm3[1,1]+cm3[0,0])/(cm3[0,0] + cm3[0,1]+cm $^{-1}$) 3[1,0] + cm3[1,1])*100) + "%")print("The recall is "+ str(cm3[1,1]/(cm3[1,0] + cm3[1,1])*100) + "%")print("The precision is "+ str(cm3[1,1]/(cm3[0,1] + cm3[1,1])*100) +" %" print('F1 score:', f1_score(ytest,y_pred3)) The accuracy is 99.92626663389628 %

The recall is 79.59183673469387 %

In [37]: $Error_rate_Model = ((cm3[0][1] + cm3[1][0]) / cm3.sum()) *100$

Specificity_Model= (cm3[1][1] / (cm3[1][1] + cm3[0][1])) *100

Sensitivity_Model= (cm3[0][0] / (cm3[0][0] + cm3[1][0])) *100

print("Error_rate_rt : ", Error_rate_model)

print("Specificity_rf : ", Specificity_Model)

print("Sensitivity_rf : ", Sensitivity_Model)

dt_probs = dt_classifier.predict_proba(xtest)

svm_fpr, svm_tpr, _ = roc_curve(ytest, svc_probs)
dt_fpr, dt_tpr, _ = roc_curve(ytest, dt_probs)

plt.plot(knn_fpr, knn_tpr, marker='.',label='KNN')
plt.plot(svm_fpr, svm_tpr, marker='.', label='SVM')

plt.plot(dt_fpr, dt_tpr, marker='.', label='Decision Tree')

False Positive Rate

→ KNN → SVM

Decision Tree

Error_rate_rf : 0.07373336610371826

Sensitivity_rf: 99.96482712532095

auc = roc_auc_score(ytest, dt_probs)

In [40]: knn_fpr, knn_tpr, _ = roc_curve(ytest, k_probs)

plt.plot([0, 1], [0, 1], linestyle='--')

plot the roc curve for the model

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

In [38]: **from sklearn.metrics import** roc_auc_score

The precision is 78.0 %

True Fake Rate

True Genuine Rate

Specificity_rf: 78.0

dt_probs = dt_probs[:, 1]

print('AUC: %.3f' % auc)

AUC: 0.898

axis labels

plt.legend()
show the plot

plt.show()

1.0

0.8

9.0 8afe

₫ 0.4

0.2

0.0

0.0

show the legend

F1 score: 0.78787878787878