5. Classification

November 30, 2020

0.1 5. Classification

Load the dataset *creditcard_fraud_subsample* from Stine.

[50]:
```python
import pandas as pd
df = pd.read_csv("creditcard_fraud_subsample.csv", sep = ';')
df.head()
```

[50]:
```
       Time        V1        V2        V3        V4        V5        V6  \
0  129095.0 -1.836940 -1.646764 -3.381168  0.473354  0.074243 -0.446751
1   69394.0  1.140431  1.134243 -1.429455  2.012226  0.622800 -1.152923
2  148476.0 -1.125092  3.682876 -6.556168  4.016731 -0.425571 -2.031210
3   48533.0  1.243848  0.524526 -0.538884  1.209196  0.479538 -0.197429
4  154493.0 -7.381547 -7.449015 -4.696287  3.728439  6.198304 -6.406267

         V7        V8        V9  …       V21       V22       V23       V24  \
0  3.791907 -1.351045  0.095186  …  0.010663  1.786681 -0.151178 -0.582098
1  0.221159  0.037372  0.034486  … -0.367136 -0.891627 -0.160578 -0.108326
2 -2.650137  1.131249 -2.946890  …  1.185580  1.348156 -0.053686  0.284122
3  0.049166  0.037792  0.128119  … -0.051660 -0.084089 -0.192846 -0.917392
4 -5.831452  1.457175 -0.646203  …  1.176575 -0.978692 -0.278330 -0.635874

        V25       V26       V27       V28  Amount  Class
0 -0.956062 -0.334369  0.715600  0.370450  720.80      1
1  0.668374 -0.352393  0.071993  0.113684    1.00      1
2 -1.174469 -0.087832  0.718790  0.676216    0.76      1
3  0.681953 -0.194419  0.045917  0.040136    1.00      1
4  0.123539  0.404729  0.704915 -1.229992   35.00      1

[5 rows x 31 columns]
```

The dataset contains transactions made by credit cards in two days in September 2013 by european cardholders. It contains only numerical input variables which are the result of a principal component analysis transformation (due to confidentiality issues there is not more background on the original features available). The only features which have not been transformed with PCA are 'Time' and 'Amount', which display the seconds elapsed between each transaction and the first transaction in the dataset and the transaction amount. The Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

1. The first task is to preprocess the dataset. Remove the feature 'Time' from the dataset and standardize the feature 'Amount' by substracting the mean and scaling to unit variance.
2. Split the data into a training and test set to be able to test the out of sample performance (use 40 percent of the data for the test set).
3. Use a decision tree to classify the data and plot the tree using sklearn.tree.plot_tree. Evaluate the performance of your classifier based on the accuracy on the test set.
4. Compare the accuracy of the decision tree with a naive classifier that simply 'predicts' no fraud in every possible case. Why is the accuracy of the second classifier still very close to one?
5. To be able to evaluate the performance we define the confusion matrix (binary classification, $1 \hat{=}$ positive)
$$C := \begin{pmatrix} C_{1,1} & C_{1,0} \\ C_{0,1} & C_{0,0} \end{pmatrix} = \begin{pmatrix} \textit{true positives} & \textit{false negatives} \\ \textit{false positives} & \textit{true negatives} \end{pmatrix},$$
where $C_{i,j}$ contains the number of observations of the test sample, which are in class i and classified as class j. Write a function which takes two vectors (where each entry is either zero or one) as input and calculates the confusion matrix. Use your function to calculate the confusion matrix on the test set for both classifiers.
6. We define
$$\textit{precision} := \frac{C_{1,1}}{C_{1,1} + C_{0,1}}$$
and
$$\textit{recall} := \frac{C_{1,1}}{C_{1,1} + C_{1,0}}.$$
What do precision and recall measure? Write functions which take two vectors as input and calculate precision and recall. Use your functions to evaluate your classifier (here nan is considered a valid result if you divide by zero).
7. Finally, we combine precision and recall to the F_1-score
$$F_1 := 2\frac{\textit{precision} \cdot \textit{recall}}{\textit{precision} + \textit{recall}}.$$
Write a function which takes two vectors as input and calculates the F_1-score. Try to train new classifiers (e.g. Random Forest, SVM, Logistic Regression) with different tuning parameters and maximize the F_1-score on the test set.

0.2 5. Classification

[51]:
```
from IPython.core.interactiveshell import InteractiveShell     #allows printing␣
 ↪multiple lines
InteractiveShell.ast_node_interactivity = "all"
```

[52]:
```
import pandas as pd
df = pd.read_csv("creditcard_fraud_subsample.csv", sep = ';')
df.head()
df.shape
```

[52]:
```
     Time        V1        V2        V3        V4        V5        V6  \
0  129095.0 -1.836940 -1.646764 -3.381168  0.473354  0.074243 -0.446751
1   69394.0  1.140431  1.134243 -1.429455  2.012226  0.622800 -1.152923
2  148476.0 -1.125092  3.682876 -6.556168  4.016731 -0.425571 -2.031210
3   48533.0  1.243848  0.524526 -0.538884  1.209196  0.479538 -0.197429
4  154493.0 -7.381547 -7.449015 -4.696287  3.728439  6.198304 -6.406267

        V7        V8        V9  …       V21       V22       V23       V24  \
0  3.791907 -1.351045  0.095186  …  0.010663  1.786681 -0.151178 -0.582098
1  0.221159  0.037372  0.034486  … -0.367136 -0.891627 -0.160578 -0.108326
2 -2.650137  1.131249 -2.946890  …  1.185580  1.348156 -0.053686  0.284122
3  0.049166  0.037792  0.128119  … -0.051660 -0.084089 -0.192846 -0.917392
4 -5.831452  1.457175 -0.646203  …  1.176575 -0.978692 -0.278330 -0.635874

       V25       V26       V27       V28  Amount  Class
0 -0.956062 -0.334369  0.715600  0.370450  720.80      1
1  0.668374 -0.352393  0.071993  0.113684    1.00      1
2 -1.174469 -0.087832  0.718790  0.676216    0.76      1
3  0.681953 -0.194419  0.045917  0.040136    1.00      1
4  0.123539  0.404729  0.704915 -1.229992   35.00      1

[5 rows x 31 columns]
```

[52]:
```
(50300, 31)
```

[53]:
```python
#df1 = df.loc[:,"Time":"V4"]
#df1
```

[54]:
```python
df = df.drop(columns=["Time"])
```

[55]:
```python
df["Amount"]
```

[55]:
```
0        720.80
1          1.00
2          0.76
3          1.00
4         35.00
          …
50295     12.00
50296     15.41
50297      3.78
50298     29.18
50299     12.99
Name: Amount, Length: 50300, dtype: float64
```

[56]:
```python
from sklearn import preprocessing
import numpy as np
```

```
amount = preprocessing.scale(df["Amount"])
df["Amount"] =amount
```

[57]:
```
df["Amount"]
```

[57]:
```
0        2.625242
1       -0.360633
2       -0.361629
3       -0.360633
4       -0.219594
           …
50295   -0.315003
50296   -0.300858
50297   -0.349101
50298   -0.243737
50299   -0.310896
Name: Amount, Length: 50300, dtype: float64
```

[58]:
```
df["Amount"].mean()
df["Amount"].std()
```

[58]:
```
1.7220375776984112e-16
```

[58]:
```
1.0000099405060854
```

[59]:
```
df.head()
df.shape
```

[59]:
```
         V1        V2        V3        V4        V5        V6        V7  \
0 -1.836940 -1.646764 -3.381168  0.473354  0.074243 -0.446751  3.791907
1  1.140431  1.134243 -1.429455  2.012226  0.622800 -1.152923  0.221159
2 -1.125092  3.682876 -6.556168  4.016731 -0.425571 -2.031210 -2.650137
3  1.243848  0.524526 -0.538884  1.209196  0.479538 -0.197429  0.049166
4 -7.381547 -7.449015 -4.696287  3.728439  6.198304 -6.406267 -5.831452

         V8        V9       V10  …       V21       V22       V23       V24  \
0 -1.351045  0.095186 -0.084500  …  0.010663  1.786681 -0.151178 -0.582098
1  0.037372  0.034486 -1.879644  … -0.367136 -0.891627 -0.160578 -0.108326
2  1.131249 -2.946890 -4.816401  …  1.185580  1.348156 -0.053686  0.284122
3  0.037792  0.128119 -0.552903  … -0.051660 -0.084089 -0.192846 -0.917392
4  1.457175 -0.646203 -4.029129  …  1.176575 -0.978692 -0.278330 -0.635874

        V25       V26       V27       V28    Amount  Class
0 -0.956062 -0.334369  0.715600  0.370450  2.625242      1
1  0.668374 -0.352393  0.071993  0.113684 -0.360633      1
2 -1.174469 -0.087832  0.718790  0.676216 -0.361629      1
```

```
3  0.681953 -0.194419  0.045917  0.040136 -0.360633      1
4  0.123539  0.404729  0.704915 -1.229992 -0.219594      1

[5 rows x 30 columns]
```

[59]:
```
(50300, 30)
```

Split Data

[60]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(df.loc[:,"V1":"Amount"],␣
 ↪df["Class"], test_size = 0.4, random_state = 1)
print("Model:")
print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
```

```
Model:
(30180, 29) (20120, 29) (30180,) (20120,)
```

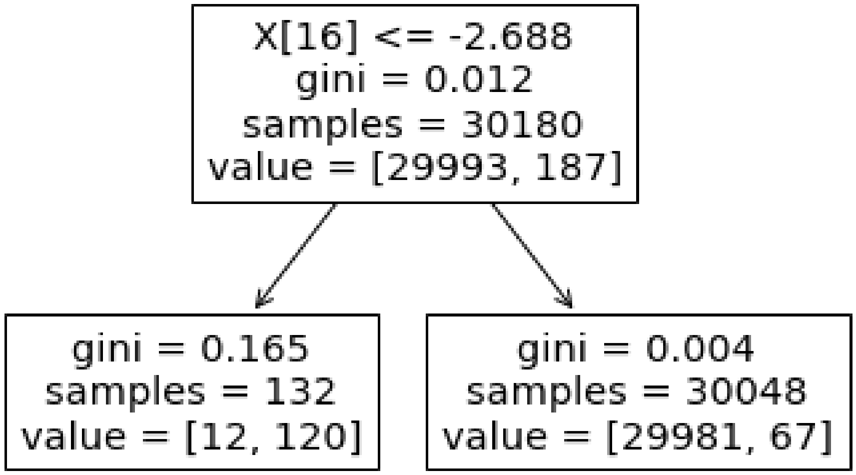
Fit Decison Tree

[61]:
```
from sklearn.tree import DecisionTreeClassifier
decisiontree = DecisionTreeClassifier(max_depth=1, random_state=1)

model = decisiontree.fit(X_train, Y_train)
print("Accuracy:", np.mean(np.equal(model.predict(X_test), Y_test)) )
tree_predict = model.predict(X_test)
```

```
Accuracy: 0.9977634194831014
```

[62]:
```
import sklearn

sklearn.tree.plot_tree(model)
```

[62]:
```
[Text(167.4, 163.07999999999998, 'X[16] <= -2.688\ngini = 0.012\nsamples =
30180\nvalue = [29993, 187]'),
 Text(83.7, 54.360000000000014, 'gini = 0.165\nsamples = 132\nvalue = [12,
120]'),
 Text(251.10000000000002, 54.360000000000014, 'gini = 0.004\nsamples =
30048\nvalue = [29981, 67]')]
```



Naive Classifier

[63]:
```
df["Class"].sum()     #there are 300 Fraud Transactions (labeled as "1"),␣
 ↪not-Fraud is labeled as "0"
```

[63]: 300

[64]:
```
naive_predict = np.array(np.linspace(0,0,X_test.shape[0]))       # 0 is for no␣
 ↪fraud
naive_predict[:10]             #naive classifier predicts only non-fraud (0)␣
 ↪for all observations
naive_predict.shape
Y_test.shape
print("Accuracy:", np.mean(np.equal(naive_predict, Y_test)) )
```

[64]: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

[64]: (20120,)

[64]: (20120,)

```
Accuracy: 0.9943836978131213
```

The accuracy of the naive classifier is close the the accuracy of the tree classifier, because there are predominantly non-fraud transaction in the set. Set is highly inbalanced.

0.2.1 Confusion Matrix

Create test numbers for coding the confusion matrix function

[65]:
```
a = np.array([1,0,1,1,0,1,1])
b = np.array([1,1,1,0,0,0,0])
a
b
```

[65]:
```
array([1, 0, 1, 1, 0, 1, 1])
```

[65]:
```
array([1, 1, 1, 0, 0, 0, 0])
```

Create Confusion Matrix Function

[66]:
```
def conf_matrix(a,b):
    tru_pos=0
    tru_neg=0
    fals_pos=0
    fals_neg=0

    for i,s in zip(a,b):
        if i == 1 and s == 1:
            tru_pos +=1
        elif i == 0 and s == 0:
            tru_neg +=1
        elif i == 0 and s == 1:
            fals_pos +=1
        elif i == 1 and s == 0:
            fals_neg +=1
        #print(i, s)

    conf_mx=np.array([[tru_pos, fals_neg], [fals_pos, tru_neg]])
    #print(conf_mx)
    #acc = (tru_pos + tru_neg)/(tru_pos + tru_neg + fals_neg + fals_pos)    #if␣
↪want to see accuracy enable this(precison, recall wont work)
    #acc
    return conf_mx
```

Test it

[67]:
```
conf_mx = conf_matrix(a,b)
```

[68]:
```
conf_mx
```

[68]:
```
array([[2, 3],
       [1, 1]])
```

Apply function on two tree and naive classifier predictions

[69]:
```
conf_matrix(Y_test,naive_predict)
```

[69]: array([[0,   113],
       [0, 20007]])

[70]:
```
conf_matrix(Y_test,tree_predict)
```

[70]: array([[77,   36],
       [9, 19998]])

0.2.2 Precision and recall functions

[71]:
```
def precision(a,b):
    conf_mx = conf_matrix(a,b)
    precis = conf_mx[0][0]/ (conf_mx[0][0] + conf_mx[1][0])      #true␣
 ↪positives divided by false positives
    return precis
```

[72]:
```
def recall(a,b):
    conf_mx = conf_matrix(a,b)
    rec = conf_mx[0][0] / (conf_mx[0][0] + conf_mx[0][1])      #true positives␣
 ↪divided by false negatives
    return rec
```

Precision and Recall: Naive and Tree Model Native

[73]:
```
precision(Y_test,naive_predict)
recall(Y_test,naive_predict)
```

```
&lt;ipython-input-71-99676f7ca788&gt;:3: RuntimeWarning: invalid value encountered in
long_scalars
  precis = conf_mx[0][0]/ (conf_mx[0][0] + conf_mx[1][0])      #true positives
divided by false positives
```

[73]: nan

[73]: 0.0

Tree

[74]:
```
tree_precis = round(precision(Y_test,tree_predict), 3)
tree_recall = round(recall(Y_test,tree_predict), 3)
tree_precis
tree_recall
```

[74]: 0.895

[74]: 0.681

Manually calculate to check results

[75]:
```python
77/(77+9)   #precision tree
```

[75]: 0.8953488372093024

[76]:
```python
77/(77+36)  #recall tree
```

[76]: 0.6814159292035398

F-Score Function

[77]:
```python
def f_score(a, b):
    precision_ = precision(a,b)
    recall_ = recall(a,b)
    return 2*( (precision_ * recall_)/(precision_ + recall_) )
```

[78]:
```python
f_score(Y_test,tree_predict)        #
```

[78]: 0.7738693467336683

0.2.3 Try other Models

Random Forrest

[79]:
```python
from sklearn.ensemble import RandomForestClassifier

randforest = RandomForestClassifier(max_depth=4, random_state=0, n_estimators =␣
↪100)

model = randforest.fit(X_train, Y_train)
print('Accuracy:', np.mean(np.equal(model.predict(X_test),Y_test)) )
forrest_predict = model.predict(X_test)
```

```
Accuracy: 0.9988071570576541
```

Support Vector Machine

[80]:
```python
from sklearn import svm

X_train.shape
Y_train.shape

#supvecma = svm.SVC(kernel = "linear", C = 2)                          #hat␣
↪nicht functioniert
#supvecma = svm.SVC(kernel = "poly", degree = 2, gamma = "scale")
#supvecma = svm.SVC(kernel = "poly", degree = 3, gamma = "scale")
supvecma = svm.SVC(kernel = "rbf", C = 3, gamma = "scale")

svm_model = supvecma.fit(X_train, Y_train)
```

```
print('Accuracy:', np.mean(np.equal(svm_model.predict(X_test),Y_test)) )
svm_predict = svm_model.predict(X_test)
```

[80]: (30180, 29)

[80]: (30180,)

```
Accuracy: 0.998558648111332
```

Logistic Regression

[81]:
```
from sklearn.linear_model import LogisticRegression

logisticRegr = LogisticRegression(penalty = "l2", solver = "liblinear",␣
 ↪multi_class = "auto")
model = logisticRegr.fit(X_train, Y_train)

print("Accuracy:", np.mean(np.equal(model.predict(X_test), Y_test)) )
logistic_predict = model.predict(X_test)
```

```
Accuracy: 0.9981610337972167
```

[82]:
```
f_score(Y_test,tree_predict)        #
f_score(Y_test,forrest_predict)        #
f_score(Y_test,svm_predict)        #
f_score(Y_test,logistic_predict)        #
```

[82]: 0.7738693467336683

[82]: 0.883495145631068

[82]: 0.8542713567839195

[82]: 0.814070351758794

[ ]:

[ ]:

[ ]: