Credit_Prediction_ML

June 6, 2023

[ ]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn import tree
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn import svm
from itertools import combinations
import warnings
from sklearn.feature_selection import VarianceThreshold
from sklearn.feature_selection import SelectKBest, f_classif
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_classification
from matplotlib.colors import ListedColormap
from sklearn.naive_bayes import GaussianNB
import xgboost as xgb
from sklearn.neighbors import KNeighborsClassifier
from sklearn.utils.multiclass import unique_labels
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.preprocessing import MinMaxScaler
```

[ ]:
```
app = pd.read_csv('application_record.csv')
cred = pd.read_csv('credit_record.csv')
```

[ ]:
```
app.head()
```

[ ]:
```
        ID CODE_GENDER FLAG_OWN_CAR FLAG_OWN_REALTY  CNT_CHILDREN  \
0  5008804           M            Y               Y             0
1  5008805           M            Y               Y             0
2  5008806           M            Y               Y             0
3  5008808           F            N               Y             0
4  5008809           F            N               Y             0

   AMT_INCOME_TOTAL      NAME_INCOME_TYPE            NAME_EDUCATION_TYPE  \
0          427500.0               Working               Higher education
1          427500.0               Working               Higher education
2          112500.0               Working  Secondary / secondary special
3          270000.0  Commercial associate  Secondary / secondary special
4          270000.0  Commercial associate  Secondary / secondary special

     NAME_FAMILY_STATUS  NAME_HOUSING_TYPE  DAYS_BIRTH  DAYS_EMPLOYED  \
0        Civil marriage   Rented apartment      -12005          -4542
1        Civil marriage   Rented apartment      -12005          -4542
2               Married  House / apartment      -21474          -1134
3  Single / not married  House / apartment      -19110          -3051
4  Single / not married  House / apartment      -19110          -3051

   FLAG_MOBIL  FLAG_WORK_PHONE  FLAG_PHONE  FLAG_EMAIL OCCUPATION_TYPE  \
0           1                1           0           0             NaN
1           1                1           0           0             NaN
2           1                0           0           0  Security staff
3           1                0           1           1     Sales staff
4           1                0           1           1     Sales staff

   CNT_FAM_MEMBERS
0              2.0
1              2.0
2              2.0
3              1.0
4              1.0
```

[ ]:
```python
cred.head()
```

[ ]:
```
        ID  MONTHS_BALANCE STATUS
0  5001711               0      X
1  5001711              -1      0
2  5001711              -2      0
3  5001711              -3      0
4  5001712               0      C
```

[ ]:
```python
merge = pd.merge(cred,app,on = "ID" , how = "inner")
merge.head()
```

[ ]:
```
        ID  MONTHS_BALANCE STATUS CODE_GENDER FLAG_OWN_CAR FLAG_OWN_REALTY  \
0  5008804               0      C           M            Y               Y
1  5008804              -1      C           M            Y               Y
2  5008804              -2      C           M            Y               Y
3  5008804              -3      C           M            Y               Y
4  5008804              -4      C           M            Y               Y

   CNT_CHILDREN  AMT_INCOME_TOTAL NAME_INCOME_TYPE NAME_EDUCATION_TYPE  \
0             0          427500.0          Working    Higher education
1             0          427500.0          Working    Higher education
2             0          427500.0          Working    Higher education
3             0          427500.0          Working    Higher education
4             0          427500.0          Working    Higher education

  NAME_FAMILY_STATUS NAME_HOUSING_TYPE  DAYS_BIRTH  DAYS_EMPLOYED  FLAG_MOBIL  \
0     Civil marriage  Rented apartment      -12005          -4542           1
1     Civil marriage  Rented apartment      -12005          -4542           1
2     Civil marriage  Rented apartment      -12005          -4542           1
3     Civil marriage  Rented apartment      -12005          -4542           1
4     Civil marriage  Rented apartment      -12005          -4542           1

   FLAG_WORK_PHONE  FLAG_PHONE  FLAG_EMAIL OCCUPATION_TYPE  CNT_FAM_MEMBERS
0                1           0           0             NaN              2.0
1                1           0           0             NaN              2.0
2                1           0           0             NaN              2.0
3                1           0           0             NaN              2.0
4                1           0           0             NaN              2.0
```

[ ]:
```
np.random.seed(1234)

merge.rename(columns={'NAME_FAMILY_STATUS': 'IsMarried'}, inplace=True)
train, test = train_test_split(merge, test_size=0.2)

def prep_data(data):
    df = data.copy()
    df = df.dropna()

    sentiment_mapping = {'X': 1, '0': 0, '1': 0, '2': 0, '3': 0, '4': 0, '5':␣
 ↪0, 'C': 1}
    df['STATUS'] = df['STATUS'].map(sentiment_mapping)

    le = LabelEncoder()
    df["CODE_GENDER"] = le.fit_transform(df["CODE_GENDER"])
    df["FLAG_OWN_CAR"] = le.fit_transform(df["FLAG_OWN_CAR"])
    df["FLAG_OWN_REALTY"] = le.fit_transform(df["FLAG_OWN_REALTY"])
```

```python
    maps = {'Academic degree': 4, 'Higher education': 4, 'Incomplete higher':␣
↪3, 'Secondary / secondary special': 2, 'Lower secondary': 1}
    df['NAME_EDUCATION_TYPE'] = df['NAME_EDUCATION_TYPE'].map(maps)

    maps2 = {'Married': 1, 'Single / not married': 0, 'Civil marriage': 1,␣
↪'Separated': 0, 'Widow': 0}
    df['IsMarried'] = df['IsMarried'].map(maps2)

    maps3 = {'Working': 1, 'Commercial associate': 1, 'State servant': 1,␣
↪'Pensioner': 1, 'Student': 0}
    df['NAME_INCOME_TYPE'] = df['NAME_INCOME_TYPE'].map(maps3)

    x = df.drop(["NAME_HOUSING_TYPE", "OCCUPATION_TYPE", "ID","STATUS"], axis=1)
    y = df["STATUS"]

    return x, y

x_train, y_train = prep_data(train)
x_test,y_test = prep_data(test)
```

[ ]:
```python
x_test.head()
```

[ ]:
```
        MONTHS_BALANCE  CODE_GENDER  FLAG_OWN_CAR  FLAG_OWN_REALTY  \
99062              -51            0             0                1
730461             -23            0             0                1
527997             -25            0             0                1
18953               -7            1             1                1
423587              -2            0             1                0

        CNT_CHILDREN  AMT_INCOME_TOTAL  NAME_INCOME_TYPE  NAME_EDUCATION_TYPE  \
99062              0          247500.0                 1                    2
730461             1          121500.0                 1                    4
527997             1           90000.0                 1                    2
18953              0          360000.0                 1                    4
423587             0          180000.0                 1                    2

        IsMarried  DAYS_BIRTH  DAYS_EMPLOYED  FLAG_MOBIL  FLAG_WORK_PHONE  \
99062           1      -16340          -8647           1                0
730461          1      -11981          -2965           1                1
527997          0      -12037          -2706           1                0
18953           1      -19958          -7465           1                0
423587          1      -18341          -5574           1                1

        FLAG_PHONE  FLAG_EMAIL  CNT_FAM_MEMBERS
99062            0           0              2.0
730461           0           0              3.0
527997           0           0              2.0
```

```
18953            0            0              2.0
423587           1            0              2.0
```

```
[ ]: variance_filter = VarianceThreshold()
     variance_filter.fit(x_train)
     non_constant_features = variance_filter.get_support(indices=True)
     x_train = x_train.iloc[:, non_constant_features]
```

```
[ ]: selector = SelectKBest(score_func=f_classif, k=10)

     X_new = selector.fit_transform(x_train, y_train)

     mask = selector.get_support()
     new_features = []

     for bool, feature in zip(mask, x_train.columns):
         if bool:
             new_features.append(feature)

     print('The best features are: ', new_features)
```

```
The best features are:  ['MONTHS_BALANCE', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'IsMarried', 'DAYS_BIRTH',
'DAYS_EMPLOYED', 'FLAG_WORK_PHONE', 'FLAG_EMAIL']
```

```
[ ]: cols= new_features
     x_train = x_train[cols]
     x_test = x_test[cols]
```

0.0.1 Logistic Regression

```
[ ]: LR = LogisticRegression()
     LR.fit(x_train,y_train)
     y_pred = LR.predict(x_test)

     score_pred = accuracy_score(y_test, y_pred)
     print('Accuracy: %f' % score_pred)
```

```
Accuracy: 0.613943
```

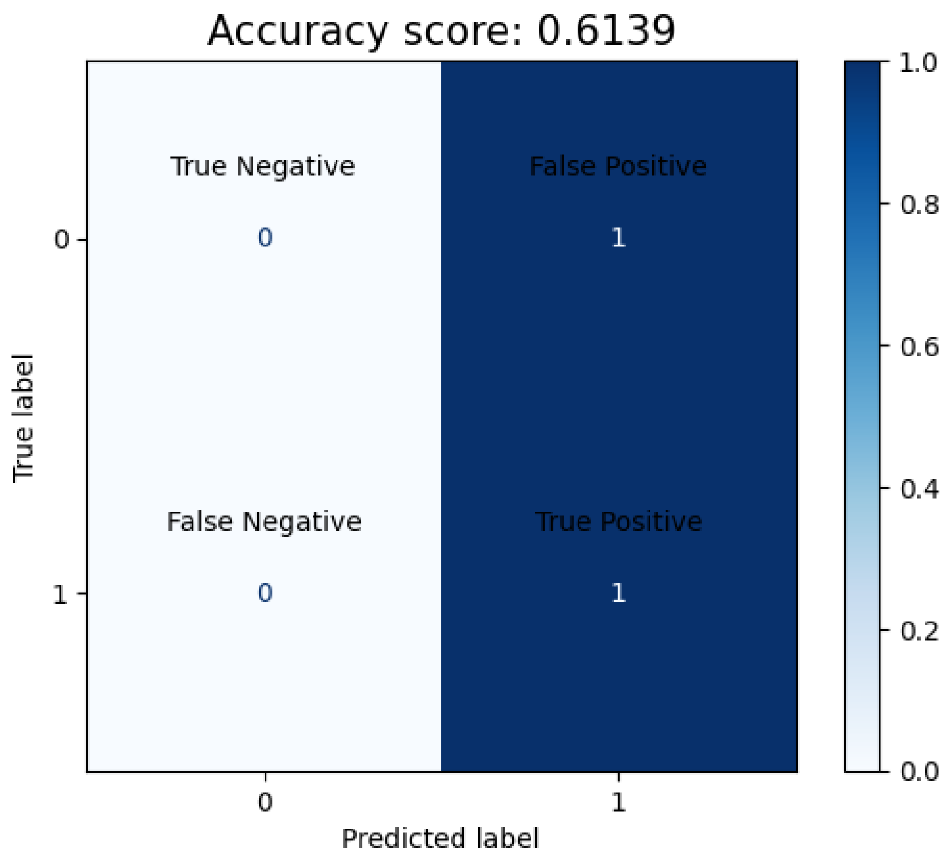
```
[ ]: cm = confusion_matrix(y_test, y_pred, normalize='true')
     labels = unique_labels(y_test)
     cm_display = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
     fig, ax = plt.subplots()
     cm_display.plot(ax=ax, cmap='Blues')

     y_positions = [-.2, -.2, .8, .8]
     x_positions = [0, 1, 0, 1]
```

```
texts = ['True Negative', 'False Positive', 'False Negative', 'True Positive']

for x, y, text in zip(x_positions, y_positions, texts):
    ax.text(x, y, text, va='center', ha='center', color='black', fontsize=10)

plt.title(f"Accuracy score: {score_pred.round(4)}", size = 15)
plt.show()
```



0.0.2 Random Forest

[ ]:
```
forest = RandomForestClassifier()

forest.fit(x_train, y_train)
y_pred = forest.predict(x_test)

score_pred = accuracy_score(y_test, y_pred)
print('Accuracy: %f' % score_pred)
```

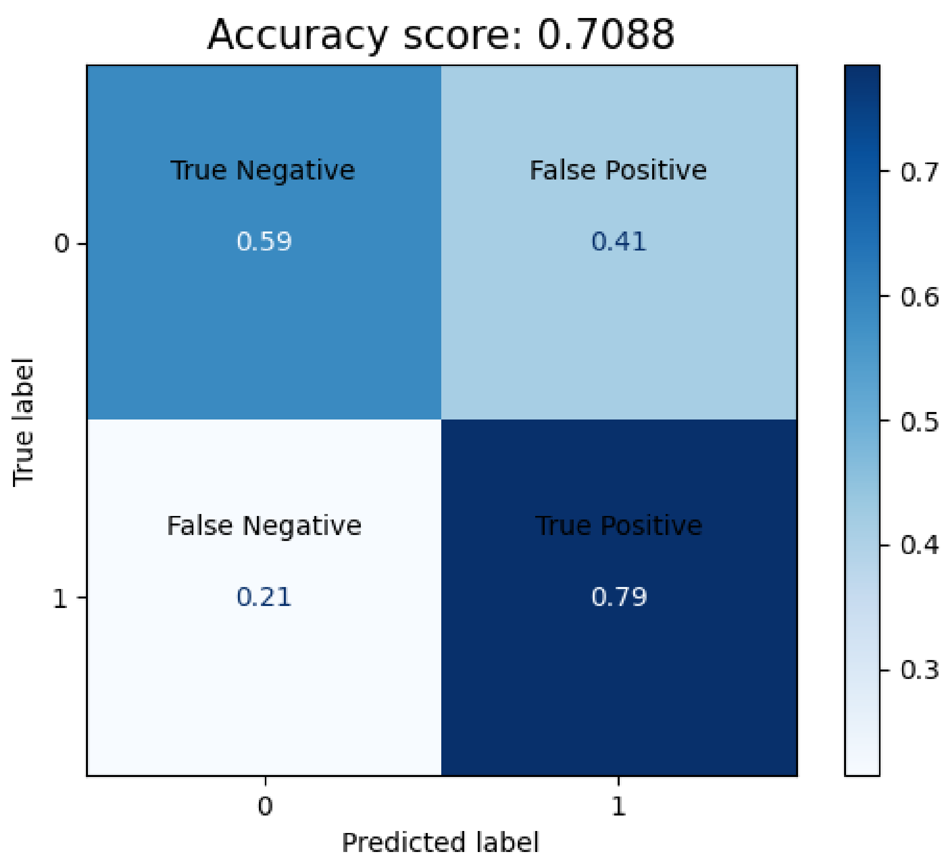
```
Accuracy: 0.708812
```

[ ]:
```python
cm = confusion_matrix(y_test, y_pred, normalize='true')
labels = unique_labels(y_test)
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
fig, ax = plt.subplots()
cm_display.plot(ax=ax, cmap='Blues')

y_positions = [-.2, -.2, .8, .8]
x_positions = [0, 1, 0, 1]
texts = ['True Negative', 'False Positive', 'False Negative', 'True Positive']

for x, y, text in zip(x_positions, y_positions, texts):
    ax.text(x, y, text, va='center', ha='center', color='black', fontsize=10)

plt.title(f"Accuracy score: {score_pred.round(4)}", size = 15)
plt.show()
```



0.0.3 Gaussian Naive Bayes

[ ]:
```
nb_classifier = GaussianNB()
nb_classifier.fit(x_train, y_train)
y_pred = nb_classifier.predict(x_test)

score_pred = accuracy_score(y_test, y_pred)
print('Accuracy: %f' % score_pred)
```

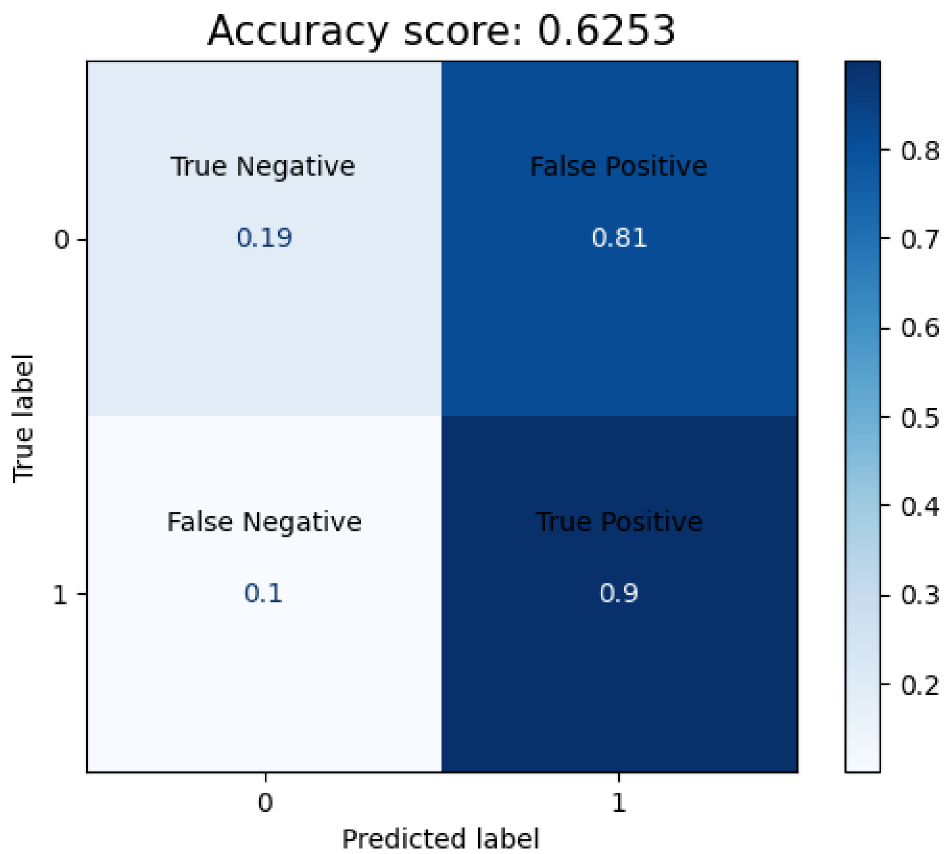
```
Accuracy: 0.625334
```

[ ]:
```
cm = confusion_matrix(y_test, y_pred, normalize='true')
labels = unique_labels(y_test)
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
fig, ax = plt.subplots()
cm_display.plot(ax=ax, cmap='Blues')

y_positions = [-.2, -.2, .8, .8]
x_positions = [0, 1, 0, 1]
texts = ['True Negative', 'False Positive', 'False Negative', 'True Positive']

for x, y, text in zip(x_positions, y_positions, texts):
    ax.text(x, y, text, va='center', ha='center', color='black', fontsize=10)

plt.title(f"Accuracy score: {score_pred.round(4)}", size = 15)
plt.show()
```



0.0.4 XG Boost

[ ]:
```
xgb_model = xgb.XGBClassifier()

xgb_model.fit(x_train, y_train)
y_pred = xgb_model.predict(x_test)

score_pred = accuracy_score(y_test, y_pred)
print('Accuracy: %f' % score_pred)
```

```
Accuracy: 0.701423
```

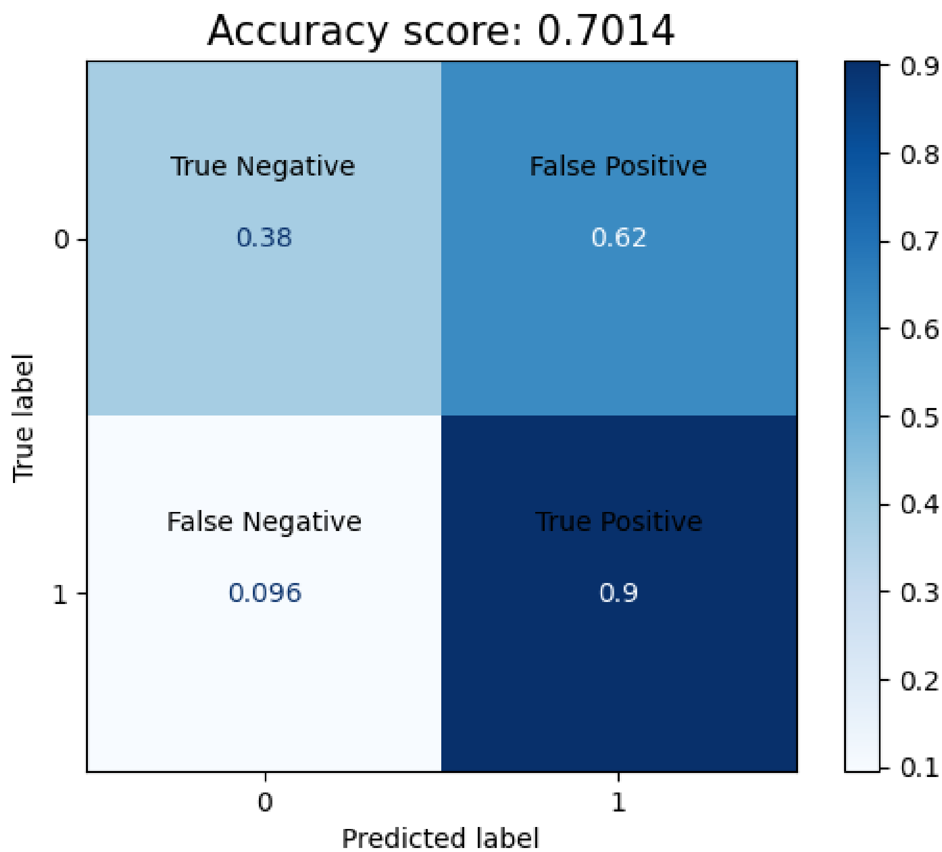
[ ]:
```
cm = confusion_matrix(y_test, y_pred, normalize='true')
labels = unique_labels(y_test)
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
fig, ax = plt.subplots()
cm_display.plot(ax=ax, cmap='Blues')

y_positions = [-.2, -.2, .8, .8]
x_positions = [0, 1, 0, 1]
```

```
texts = ['True Negative', 'False Positive', 'False Negative', 'True Positive']

for x, y, text in zip(x_positions, y_positions, texts):
    ax.text(x, y, text, va='center', ha='center', color='black', fontsize=10)

plt.title(f"Accuracy score: {score_pred.round(4)}", size = 15)
plt.show()
```



0.0.5 K-Nearest Neighbors

[ ]:
```
# Normalize Data for KNN

scaler = MinMaxScaler()
x_train_normalized = scaler.fit_transform(x_train)
x_test_normalized = scaler.transform(x_test)
```

[ ]:
```
knn = KNeighborsClassifier()
knn.fit(x_train_normalized, y_train)
y_pred = knn.predict(x_test_normalized)
```

```
score_pred = accuracy_score(y_test, y_pred)
print('Accuracy: %f' % score_pred)
```

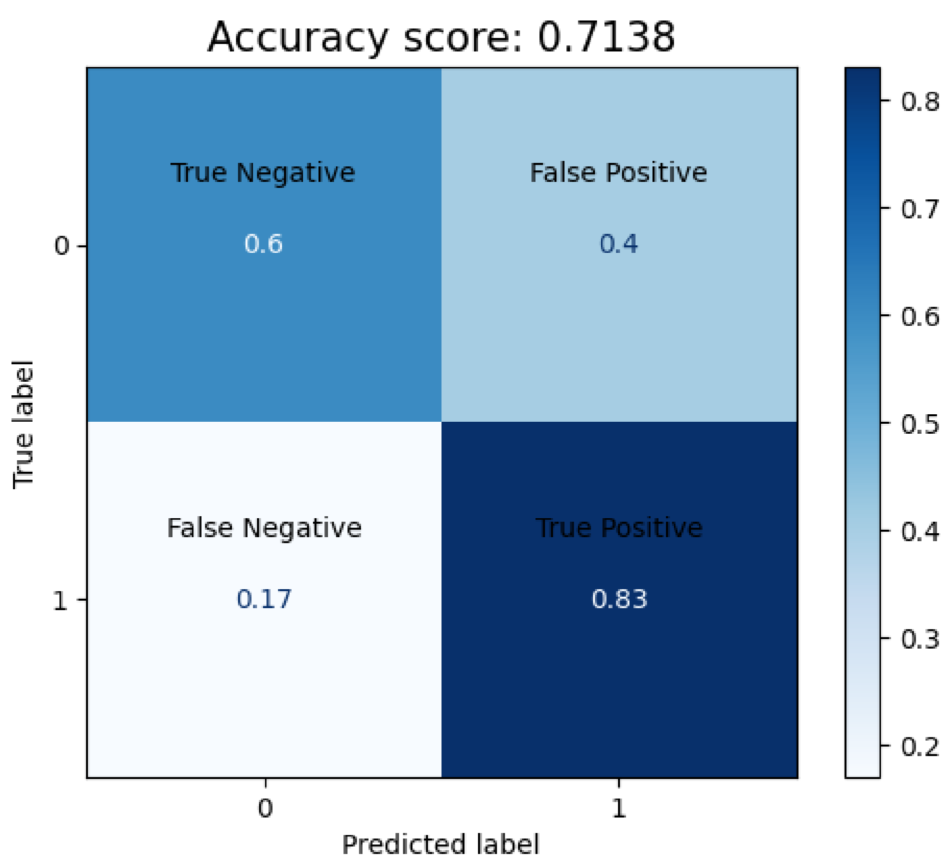
Accuracy: 0.713828

[ ]:
```
cm = confusion_matrix(y_test, y_pred, normalize='true')
labels = unique_labels(y_test)
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
fig, ax = plt.subplots()
cm_display.plot(ax=ax, cmap='Blues')

y_positions = [-.2, -.2, .8, .8]
x_positions = [0, 1, 0, 1]
texts = ['True Negative', 'False Positive', 'False Negative', 'True Positive']

for x, y, text in zip(x_positions, y_positions, texts):
    ax.text(x, y, text, va='center', ha='center', color='black', fontsize=10)

plt.title(f"Accuracy score: {score_pred.round(4)}", size = 15)
plt.show()
```



KNN Hyperparameter Optimization

[ ]:
```
hyperparameters = {
    'n_neighbors': [1, 3, 5, 7, 9],
    'metric': ['euclidean', 'manhattan']
}

scoring = 'accuracy'

param_grid = hyperparameters

grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, scoring=scoring)
grid_search.fit(x_train_normalized, y_train)

best_params = grid_search.best_params_

knn = KNeighborsClassifier(**best_params)
knn.fit(x_train_normalized, y_train)
y_pred = knn.predict(x_test_normalized)
accuracy = accuracy_score(y_test, y_pred)

print("Best hyperparameters:", best_params)
print("Test set accuracy:", accuracy)
```

```
Best hyperparameters: {'metric': 'manhattan', 'n_neighbors': 9}
Test set accuracy: 0.7409099792467404
```

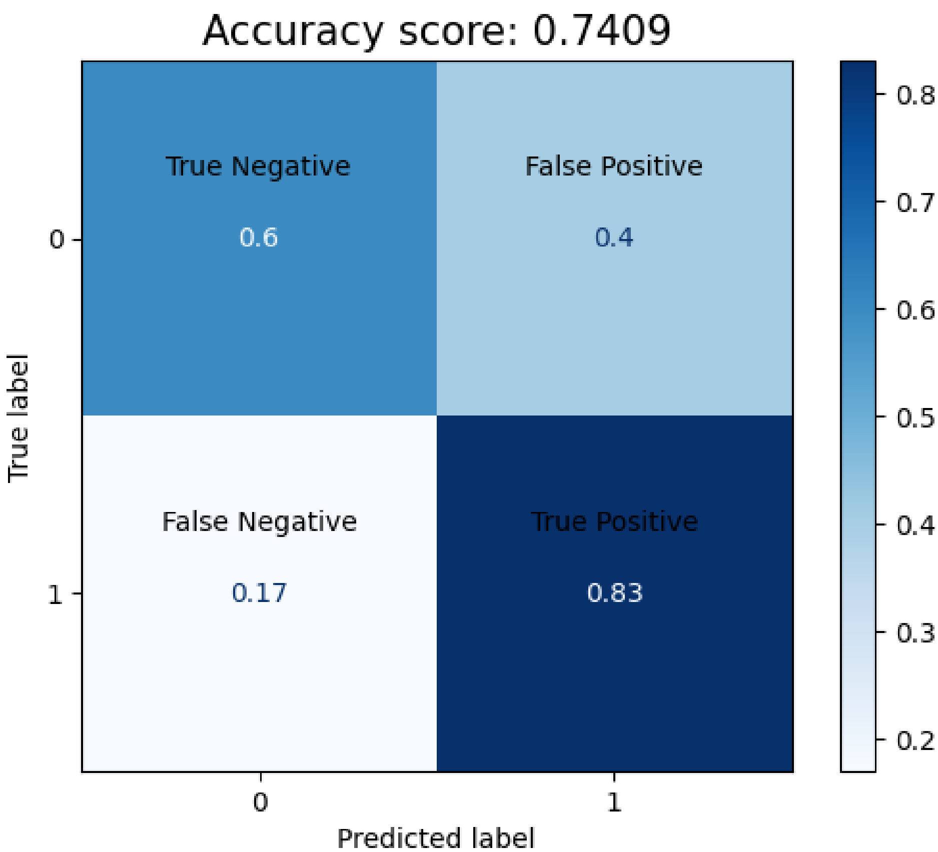
Best hyperparameters for KNN: {'metric': 'manhattan', 'n_neighbors': 9}

[ ]:
```
cm = confusion_matrix(y_test, y_pred, normalize='true')
labels = unique_labels(y_test)
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
fig, ax = plt.subplots()
cm_display.plot(ax=ax, cmap='Blues')

y_positions = [-.2, -.2, .8, .8]
x_positions = [0, 1, 0, 1]
texts = ['True Negative', 'False Positive', 'False Negative', 'True Positive']

for x, y, text in zip(x_positions, y_positions, texts):
    ax.text(x, y, text, va='center', ha='center', color='black', fontsize=10)

plt.title(f"Accuracy score: {accuracy.round(4)}", size = 15)
plt.show()
```



The best model we arrived on for classification was a KNN model using Manhattan distance and 9 neighbors. It resulted in an accuracy of approximately 0.741