06b_eval_xgboost

May 26, 2024

1 Evaluate Classification with XGBoost

1.1 Content

1. Import Data
2. Create / Train Model
3. Metrics / Confusion Matrix
4. Grid Search

[13]:
```
# imports
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from xgboost import plot_importance
from matplotlib import pyplot
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, roc_auc_score
```

Import Data

[14]:
```
# import files
df = pd.read_csv('files/fo_smote.csv')
not_automatable = [item[0] for item in pd.read_csv("files/not_automatable.csv").
 ↪values.tolist()]

# calculate fo_computerisation based on 'df_probability'
df['fo_computerisation'] = df['fo_probability'].apply(lambda x: 1 if x >= 0.5␣
 ↪else 0)

df.head(2)
```

[14]:
```
   isco08  Berufshauptgruppe    s1    s2    s3    s4    s5    s6    s7    s8  \
0    2655                  2  0.72  0.72  0.69  0.69  0.50  0.50  0.50  0.47
1    2612                  2  0.81  0.75  0.81  0.72  0.81  0.66  0.56  0.72

   …  a45  a46  a47  a48  a49  a50  a51  a52  fo_probability  \
0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0            0.37
1  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0            0.40

   fo_computerisation
0                   0
1                   0

[2 rows x 91 columns]
```

Split in Train / Test Set and train Model

[15]:
```
# Declare x, y & split Data in training and test (80/20)
X = df[not_automatable]
y = df['fo_computerisation']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[16]:
```
X.head(2)
```

[16]:
```
    s15   s8  s31  s26   s24   a12   a13   s4  s27   a5
0  0.38  0.47  0.0  0.0  0.06  0.47  0.47  0.69  0.0  0.72
1  0.56  0.72  0.0  0.0  0.19  0.78  0.81  0.72  0.0  0.81
```

[17]:
```
# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create and train the model
model = XGBClassifier(random_state=42, learning_rate=0.1, max_depth=2,␣
 ↪n_estimators=100)
model.fit(X_train_scaled, y_train)

# You can now use model.predict to make predictions on unseen data
predictions = model.predict(X_test_scaled)
```

[18]:
```
# Make prediction with train data
y_pred_train = model.predict(X_train_scaled)

# Make prediction with test data
y_proba = model.predict_proba(X_test_scaled)
```

```
y_pred = model.predict(X_test_scaled)

# Create a DataFrame with the probabilities and predictions
prediction_df = pd.DataFrame(np.c_[y_proba, y_pred], columns = ['Wkt Nicht␣
 ↪Substituierbar', 'Wkt Substituierbar', 'Vorhersage der Klasse'])

prediction_df.head(5)
```

[18]:

	Wkt Nicht Substituierbar	Wkt Substituierbar	Vorhersage der Klasse
0	0.321151	0.678849	1.0
1	0.301669	0.698331	1.0
2	0.219224	0.780776	1.0
3	0.869238	0.130762	0.0
4	0.098395	0.901605	1.0

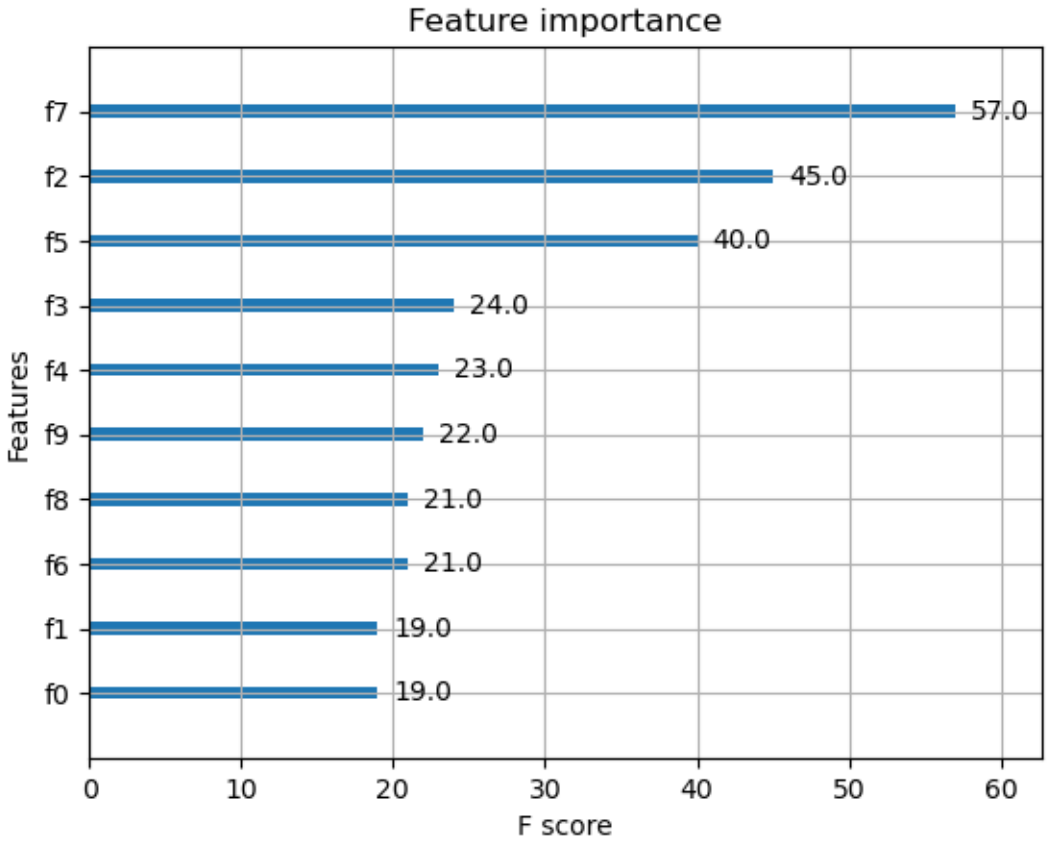
Metrics

[19]:
```
# Calculate metrics for the training set
train_accuracy = accuracy_score(y_train, y_pred_train)
train_precision = precision_score(y_train, y_pred_train)
train_recall = recall_score(y_train, y_pred_train)
train_f1 = f1_score(y_train, y_pred_train)
train_auc = roc_auc_score(y_train, model.predict_proba(X_train_scaled)[:, 1])

# Calculate metrics for the test set
test_accuracy = accuracy_score(y_test, y_pred)
test_precision = precision_score(y_test, y_pred)
test_recall = recall_score(y_test, y_pred)
test_f1 = f1_score(y_test, y_pred)
test_auc = roc_auc_score(y_test, y_proba[:, 1])
```

[20]:
```
# Read the CSV file into a DataFrame
try:
    metrics_df = pd.read_csv('files/metrics.csv')
except pd.errors.EmptyDataError:
    metrics_df = pd.DataFrame(columns=['Model', 'Test Accuracy', 'Train␣
 ↪Accuracy', 'Precision', 'Recall', 'F1 Score', 'AUC'])

# Check if the model exists in the DataFrame
if 'XGBoost' in metrics_df['Model'].values:
    # Update the row for the XGBoost model
    metrics_df.loc[metrics_df['Model'] == 'XGBoost', ['Test Accuracy', 'Train␣
 ↪Accuracy', 'Precision', 'Recall', 'F1 Score', 'AUC']] = [test_accuracy,␣
 ↪train_accuracy, test_precision, test_recall, test_f1, test_auc]
else:
    # Create a new DataFrame for the XGBoost model
```

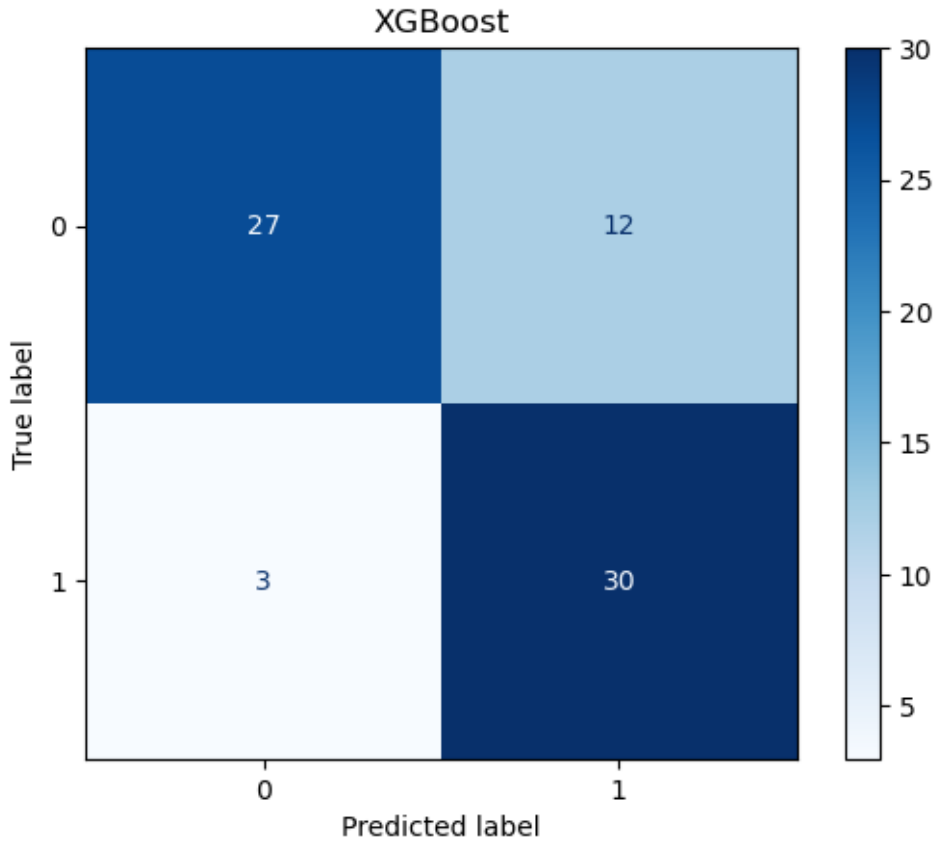
```
    new_row = pd.DataFrame({'Model': ['XGBoost'], 'Test Accuracy':␣
 ↪[test_accuracy], 'Train Accuracy': [train_accuracy], 'Precision':␣
 ↪[test_precision], 'Recall': [test_recall], 'F1 Score': [test_f1], 'AUC':␣
 ↪[test_auc]})

    # Concatenate the new row with the existing DataFrame
    metrics_df = pd.concat([metrics_df, new_row], ignore_index=True)

# Save the DataFrame to the CSV file
metrics_df.to_csv('files/metrics.csv', index=False)
```

[21]:
```
# Getting the most important feature
plot_importance(model)
pyplot.show()
print(model.feature_importances_)

# Making the Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Applying k-Fold Cross Validation
accuracies = cross_val_score(estimator=model, X=X_train, y=y_train, cv=5)
```



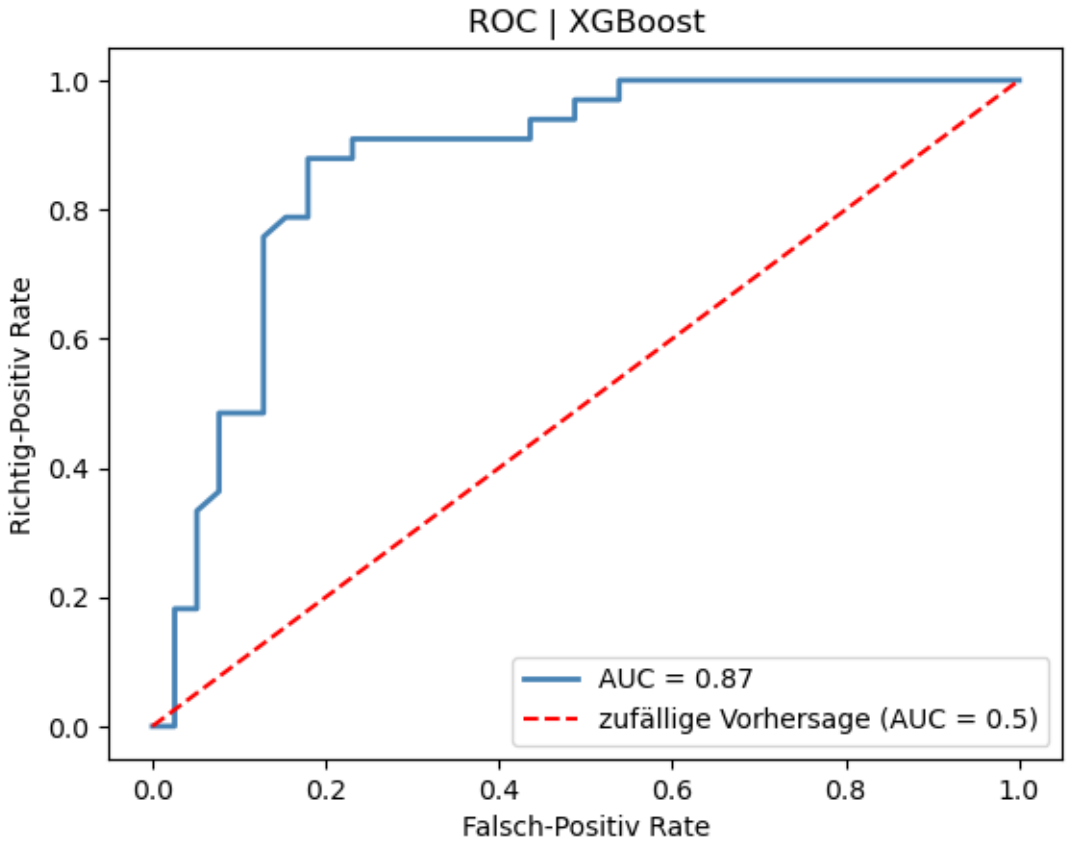
```
[0.0562953  0.03695671 0.04034574 0.04558448 0.03720998 0.26645005
 0.30468643 0.0885523  0.06841855 0.05550039]
```

[22]:
```
# Define the labels for your classes
class_names = ['0', '1']

# Create the ConfusionMatrixDisplay instance
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix,␣
 ↪display_labels=class_names)

# Plot the confusion matrix
disp.plot(cmap='Blues')

plt.title('XGBoost')
plt.show()
```



[23]:
```
# plot ROC curve
fpr, tpr, _ = roc_curve(y_test, y_proba[:, 1])
```

```
roc_auc = auc(fpr, tpr)

plt.figure()
lw = 2
plt.plot(fpr, tpr, color='#4682B4', lw=lw, label='AUC = %0.2f' % roc_auc)
plt.ylabel('Richtig-Positiv Rate')
plt.plot([0 ,1] ,[0 ,1], 'r', label='zufällige Vorhersage (AUC = 0.5)',␣
 ↪linestyle='--')
plt.xlabel('Falsch-Positiv Rate')
plt.title('ROC | XGBoost')
plt.legend(loc="lower right")

plt.show()
```



Grid Search

[24]:
```
# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 200, 300],
    'max_depth': [2, 3, 4, 5],
```

```
    'learning_rate': [0.01, 0.1, 0.2]
}

# Create a GridSearchCV object
grid_search = GridSearchCV(XGBClassifier(random_state=42), param_grid, cv=5,␣
 ↪scoring='roc_auc')

# Fit the GridSearchCV object to the data
grid_search.fit(X_train_scaled, y_train)

# Print the best parameters and the best score
print("best params: ",grid_search.best_params_)
print(grid_search.best_score_)
```

```
best params:  {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 200}
0.8581417624521073
```