PD3-Jan-Smoleń

April 13, 2021

1 PD3 - Jan Smoleń

[1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import sklearn
import category_encoders as ce
import sklearn.metrics as metrics
import statistics
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
import xgboost as xgb
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

[2]:
```
df=pd.read_csv("australia.csv")
```

W celu przyśpieszeniea późniejszego tuningowania hiperparametrów ograniczę liczbę rekordów.

[3]:
```
df=df.head(5000)
```

[4]:
```
from sklearn.model_selection import train_test_split
X=df.drop(["RainTomorrow"], axis=1)
y=df["RainTomorrow"]
X_train, X_test, y_train, y_test = train_test_split(X, y,␣
↪stratify=y,random_state = 42)
```

1.1 SVM

Pierwszym testowanym przez nas modelem będzie SVM.

[5]:
```
svm_base=SVC(random_state=42)
svm_base.fit(X_train, y_train)
preds=svm_base.predict(X_test)
```

[6]:
```
accuracy_score(preds,y_test)
```

[6]: 0.7784

Jak widzimy, SVM z domyślnymi hiperparametrami osiąga accuracy score powyżej 77%. Spróbujemy teraz znaleźć dobrą kombinacje hiperparametrów gamma i C korzystając z narzędzia GridSearchCV.

[7]:
```
svm_tuned=SVC(random_state=42)
c=[]  # wartości parametru C
gamma=[]  #wartości parametru gamma
for i in range(-4, 5):      # orientacyjne wartości na podstawie informacji␣
↪znalezionych w internecie
    c.append(10**i)
for i in range(-4, 5):
    gamma.append(10**i)
gamma.append("auto")
gamma.append("scale")
params = [{'C': c,
        'gamma': gamma}]
gs_svm=GridSearchCV(svm_tuned, param_grid=params, scoring='accuracy', cv=4,␣
↪n_jobs=2)
gs_svm.fit(X_train, y_train)
gs_svm.best_params_
```

[7]: {'C': 100, 'gamma': 0.0001}

[8]:
```
svm_acc=accuracy_score(gs_svm.predict(X_test),y_test)
svm_acc
```

[8]: 0.864

Tuning dwóch hiperparametrów pozwala zatem na poprawienie accuracy score modelu o prawie 10% przy wartościach C=100, gamma=0.0001. Gdyby celem zadania było znalezienie optymalnych parametrów to moglibyśmy poszukać także w okolicach tych wartości oraz zmodyfikować atrybut kernel.

1.2 XGBoost

[9]:
```
import xgboost as xgb
xgb_model = xgb.XGBClassifier(objective = "binary:logistic", seed = 1613,␣
↪use_label_encoder=False)
xgb_model.fit(X_train, y_train)
```

```
[11:32:51] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default
evaluation metric used with the objective 'binary:logistic' was changed from
'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
```

[9]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=4, num_parallel_tree=1,
              random_state=1613, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              seed=1613, subsample=1, tree_method='exact',
              use_label_encoder=False, validate_parameters=1, verbosity=None)
```

[10]:
```
accuracy_score(xgb_model.predict(X_test), y_test)
```

[10]:
```
0.8488
```

Surowy XGBoost daje znacznie lepszy wynik accuracy score niż SVM z domyślnymi parametrami.

[11]:
```
xgb_tuned=xgb.XGBClassifier(objective = "binary:logistic", seed = 1613,␣
 ↪use_label_encoder=False)
eta=[]  #wartości parametru eta
max_depth=[]
for i in range(10):
    eta.append(0.01+0.03)
    max_depth.append(i)
params = [{'eta': eta,
        'max_depth': max_depth}]
gs_xgb=GridSearchCV(xgb_tuned, param_grid=params, scoring='accuracy', cv=4,␣
 ↪n_jobs=2)
gs_xgb.fit(X_train, y_train)
gs_xgb.best_params_
```

```
[11:33:53] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default
evaluation metric used with the objective 'binary:logistic' was changed from
'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
```

[11]:
```
{'eta': 0.04, 'max_depth': 5}
```

[12]:
```
xgb_acc=accuracy_score(gs_xgb.predict(X_test), y_test)
xgb_acc
```

[12]:
```
0.8448
```

W tym przypadku nie udało się polepszyć wyników modelu poprzez tuning hiperparametrów max_depth i eta.

1.3 Random Forest

[13]:
```
rfc = RandomForestClassifier(random_state=16)
rfc.fit(X_train,y_train)
```

[13]:
```
RandomForestClassifier(random_state=16)
```

[14]:
```
accuracy_score(rfc.predict(X_test),y_test)
```

[14]:
```
0.848
```

Czyli takie same accuracy jak używając XGBoosta.

[15]:
```
rfc_tuned=RandomForestClassifier(random_state=16)
n_estimators = [int(x) for x in np.linspace(start = 50, stop = 1000, num = 5)]␣
 ↪# przykładowe wartości znalezione w internecie
max_depth = [int(x) for x in np.linspace(5, 55, num = 5)]

params = [{'n_estimators': n_estimators,
        'max_depth': max_depth}]
gs_rfc=GridSearchCV(rfc_tuned, param_grid=params, scoring='accuracy', cv=4,␣
 ↪n_jobs=2)
gs_rfc.fit(X_train, y_train)
gs_rfc.best_params_
```

[15]:
```
{'max_depth': 30, 'n_estimators': 762}
```

[16]:
```
rfc_acc=accuracy_score(gs_rfc.predict(X_test),y_test)
rfc_acc
```

[16]:
```
0.8544
```

Czyli udało się trochę polepszyć wynik naszego modelu.

1.4 Ocena jakości modeli

1.4.1 Accuracy Score

[17]:
```
scores=[]
labels=[]
scores.append(svm_acc)
labels.append("SVM")
scores.append(xgb_acc)
labels.append("XGB")
scores.append(rfc_acc)
labels.append("RFC")
```

[18]:
```
pd.DataFrame({"Accuracy Score": scores}, index=labels)
```

[18]:

	Accuracy Score
SVM	0.8640
XGB	0.8448
RFC	0.8544

1.4.2 Confusion Matrix

SVM

[19]:
```
tn, fp, fn, tp = confusion_matrix(y_test, gs_svm.predict(X_test)).ravel()
pd.DataFrame({"Actual positives": [tp, fp], "Actual negatives": [fn, tn]},␣
↪index = ["Positive predictions", "Negative predictions"])
```

[19]:

	Actual positives	Actual negatives
Positive predictions	135	142
Negative predictions	28	945

XGB

[20]:
```
tn, fp, fn, tp = confusion_matrix(y_test, gs_xgb.predict(X_test)).ravel()
pd.DataFrame({"Actual positives": [tp, fp], "Actual negatives": [fn, tn]},␣
↪index = ["Positive predictions", "Negative predictions"])
```

[20]:

	Actual positives	Actual negatives
Positive predictions	128	149
Negative predictions	45	928

RFC

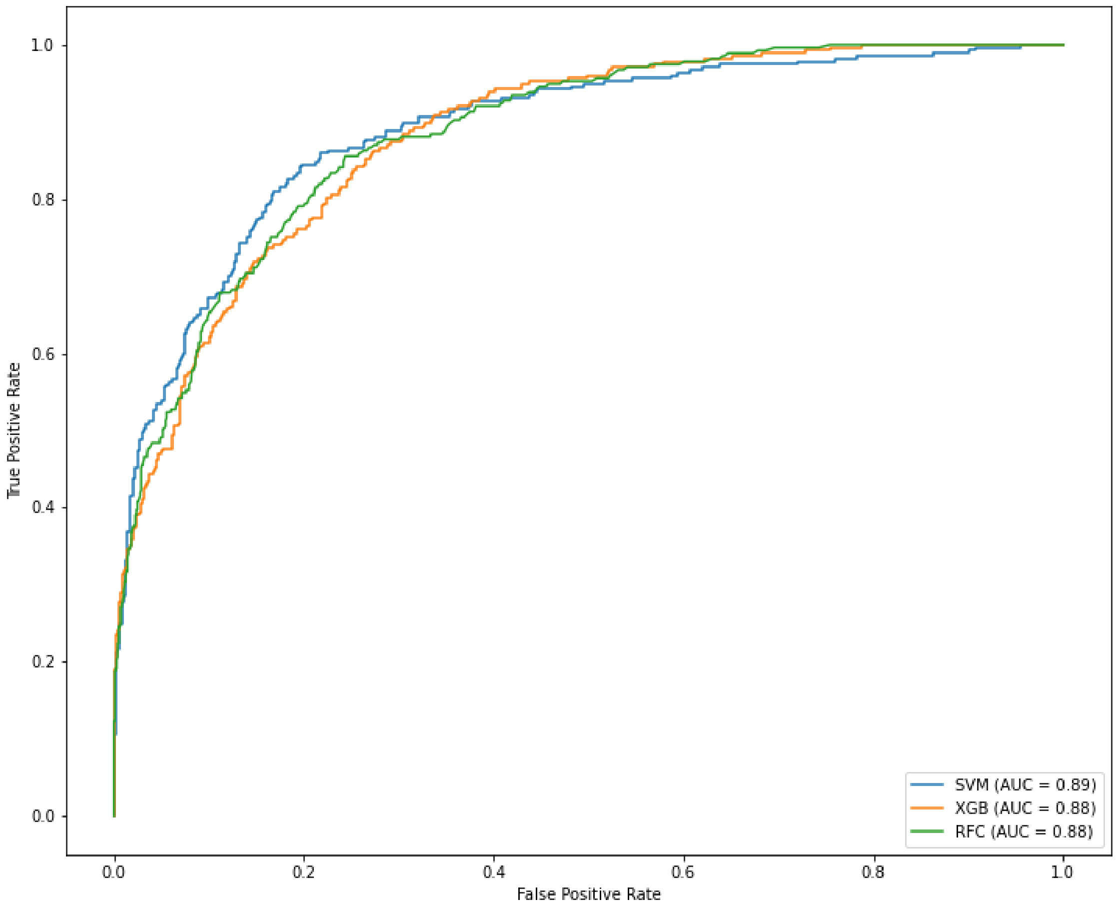
[21]:
```
tn, fp, fn, tp = confusion_matrix(y_test, gs_rfc.predict(X_test)).ravel()
pd.DataFrame({"Actual positives": [tp, fp], "Actual negatives": [fn, tn]},␣
↪index = ["Positive predictions", "Negative predictions"])
```

[21]:

	Actual positives	Actual negatives
Positive predictions	134	143
Negative predictions	39	934

1.4.3 ROC

[22]:
```
gs_svm
plt.figure(figsize=(12,10))
classifiers = [gs_svm, gs_xgb, gs_rfc]
labels=["SVM", "XGB", "RFC"]
ax = plt.gca()
for i in range(3):
    metrics.plot_roc_curve(classifiers[i], X_test, y_test, ax=ax,␣
↪name=labels[i])
```



Biorąc pod uwagę powyższe oceny jakości klasyfikatorów, w tym konkretnym przypadku najlepszym z nich wydaje się **SVM** z wytuningowanymi hiperparametrami C i gamma. Być może inne wyniki byśmy otrzymali przeprowadzając wcześniej feature engineering.