Praca Domowa 3

Bartosz Siński

In [2]:
```
import pandas as pd
import numpy as np
```

Załadowanie modelu i podział zbioru na treningowy i testowy.

In [3]:
```
from sklearn.model_selection import train_test_split
df_aus = pd.read_csv("./src/australia.csv")
X= df_aus.drop(["RainTomorrow"], axis=1)
y = df_aus["RainTomorrow"]
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,random_state = 42
```

Regresja logistyczna

In [4]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
lreg = LogisticRegression(random_state=1613,penalty='l1',solver='saga',max_iter=500).f
aus_pred1 = lreg.predict(X_test)
accuracy_score(aus_pred1,y_test)
```

```
C:\Users\komp\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.8_qbz5n2kfra8p0
\LocalCache\local-packages\Python38\site-packages\sklearn\linear_model\_sag.py:328: Co
nvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
```

Out[4]:
```
0.8519673874512584
```

Dobraliśmy parametry *penalty, max_iter* oraz *solver*. *Penalty* ustawia rodzaj kary, którą nakładamy na model za overfitting. Parametr *Solver* odpowiada za wybór algorytmu opdowiedzialnego za optymalizacje, a *max_iter* za ograniczenie liczby iteracji naszego solvera. Wybraliśmy *saga* ponieważ pozwala on na wybranie parametru regularyzacji L1 i działa szybko na dużych zbiorach danych. Zmiana kary na L1 i zwiększenie liczby iteracji nieznacznie poprawiło accuracy naszego modelu.

SVM

In [5]:
```
from sklearn.svm import SVC
svm = SVC(random_state=1613,kernel = 'linear',C=10)
svm.fit(X_train, y_train)
aus_pred2 = svm.predict(X_test)
accuracy_score(aus_pred2,y_test)
```

Out[5]:
```
0.8530308401276143
```

W powyższym modelu SVC ustawiliśmy parametr *kernel*, który ustala typ jądra używanego w algorytmie na liniowy. Zmieniliśmy także parametr regularyzacji *C* z domyślnego 1 na 10. W obu przypadkach zmiana parametru podniosła nasze accuracy.

Random Forrest

In [6]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=1613,n_estimators = 1000,max_features='log2
rfc.fit(X_train,y_train)
aus_pred3 = rfc.predict(X_test)
accuracy_score(aus_pred3,y_test)
```

Out[6]:
```
0.8644452321871676
```

W modelu Random Forrest zmieniliśmy domyślne wartości parametrów *n_estimators* i *max_features*. *n_estimators* ustala liczbę drzew w naszym lesie na podstawie których będziemy przewidywać wartość targetu. Parametr *max_features* odpowiada za liczbę zmiennych przy podziałach liści. Wartość *n_estimators* podnieśliśmy z 100 do 1000 co zwiększyło czas przygotowania modelu jednak podniosło też jego accuracy. *max_features* zmieniliśmy z domyślnej *sqrt* na *log2* co nie wpłynęło na predykcje naszego modelu.

Porównanie wyników

In [12]:
```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import f1_score
results = {
    "algorithm" : ['Logistic Regression','SVM','Random Forrest'],
    "accuracy" : [accuracy_score(y_test,aus_pred1),accuracy_score(y_test,aus_pred2),ac
    "precision" : [precision_score(y_test,aus_pred1),precision_score(y_test,aus_pred2)
    "recall" :[recall_score(y_test,aus_pred1),recall_score(y_test,aus_pred2),recall_sc
    'ROC AUC' : [roc_auc_score(y_test,aus_pred1),roc_auc_score(y_test,aus_pred2),roc_a
    'F1' : [f1_score(y_test,aus_pred1),f1_score(y_test,aus_pred2),f1_score(y_test,aus_

}
pd.DataFrame(results)
```

Out[12]:

	algorithm	accuracy	precision	recall	ROC AUC	F1
0	Logistic Regression	0.851967	0.726344	0.526231	0.735110	0.610302
1	SVM	0.853031	0.726357	0.533956	0.738564	0.615470
2	Random Forrest	0.864445	0.775473	0.541358	0.748539	0.637604

Najlepszym klasyfikatorem okazał się Random Forest. We wszystkich metrykach osiągnął lepszy wynik niż pozostałe modele. Ciekawe wydaje się być, że SVM i Regresja Logistyczna osiągnęły bardzo podobne wyniki we wszystkich metrykach.Dodatkowo zaskakujący jest tak wysoki wynik Regresji logistycznej w porównaniu do bardziej zaawansowanych modeli SVM i Random Forrest. Spojrzymy także na *table of confusion* dla tych modeli.

Regresja Logistyczna

In [8]:
```
from sklearn.metrics import confusion_matrix
tn, fp, fn, tp = confusion_matrix(y_test, aus_pred1).ravel()
pd.DataFrame({"Actual positives": [tp, fp], "Actual negatives": [fn, tn]}, index = ["P
```

Out[8]:

	Actual positives	Actual negatives
Positive predictions	1635	1472
Negative predictions	616	10382

SVM

In [9]:
```
tn, fp, fn, tp = confusion_matrix(y_test, aus_pred2).ravel()
pd.DataFrame({"Actual positives": [tp, fp], "Actual negatives": [fn, tn]}, index = ["P
```

Out[9]:

	Actual positives	Actual negatives
Positive predictions	1659	1448
Negative predictions	625	10373

Random Forest

In [10]:
```
tn, fp, fn, tp = confusion_matrix(y_test, aus_pred3).ravel()
pd.DataFrame({"Actual positives": [tp, fp], "Actual negatives": [fn, tn]}, index = ["P
```

Out[10]:

	Actual positives	Actual negatives
Positive predictions	1682	1425
Negative predictions	487	10511

Powyżej widzimy, że Random Forrest najbardziej różni się od pozostałych modeli pod względem zdecydowanie mniejszej liczby predykcji *False Negative*. Widzimy także, że modele o wiele lepiej radzą sobie z klasyfikacją obserwacji, gdzie model przewiduje brak deszczu. Może się to wiązać z dużo większą ilością obserwacji gdzie deszczu nie ma.