pd3_spytek_mikolaj

April 12, 2021

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_curve
from sklearn import svm
```

[2]:
```
#pobranie danych
df = pd.read_csv("https://raw.githubusercontent.com/mini-pw/2021L-WUM/main/
↪Prace_domowe/Praca_domowa3/australia.csv")
df.head()
```

[2]:

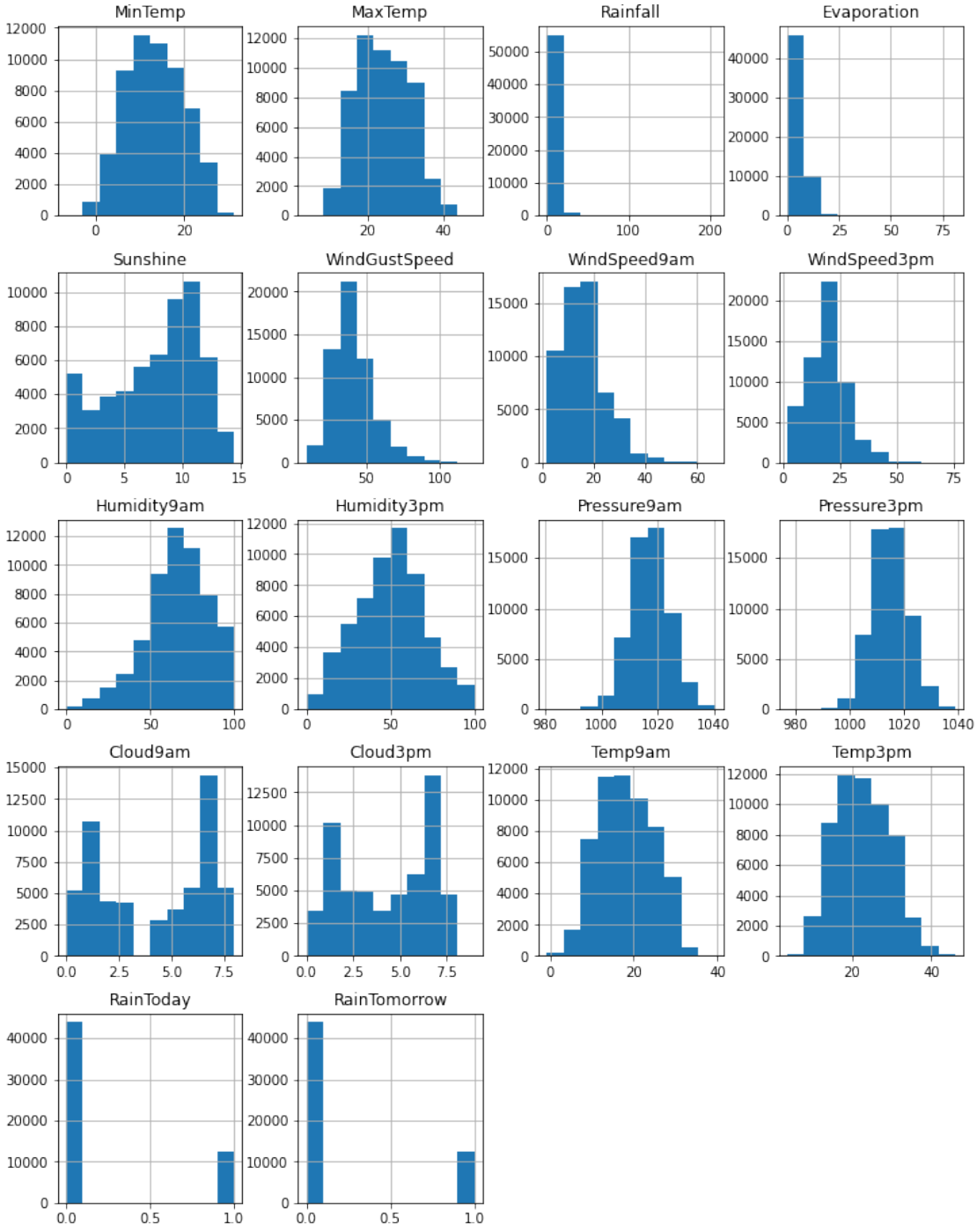
	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	\
0	17.9	35.2	0.0	12.0	12.3	48.0	
1	18.4	28.9	0.0	14.8	13.0	37.0	
2	19.4	37.6	0.0	10.8	10.6	46.0	
3	21.9	38.4	0.0	11.4	12.2	31.0	
4	24.2	41.0	0.0	11.2	8.4	35.0	

	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	\
0	6.0	20.0	20.0	13.0	1006.3	
1	19.0	19.0	30.0	8.0	1012.9	
2	30.0	15.0	42.0	22.0	1012.3	
3	6.0	6.0	37.0	22.0	1012.7	
4	17.0	13.0	19.0	15.0	1010.7	

	Pressure3pm	Cloud9am	Cloud3pm	Temp9am	Temp3pm	RainToday	RainTomorrow
0	1004.4	2.0	5.0	26.6	33.4	0	0
1	1012.1	1.0	1.0	20.3	27.0	0	0
2	1009.2	1.0	6.0	28.7	34.9	0	0
3	1009.1	1.0	5.0	29.1	35.6	0	0

```
4        1007.4        1.0        6.0        33.6        37.6              0                  0
```

[3]:
```
df.hist(figsize=(12,16))
plt.show()
```



Dla pewności sprawdziłem rozkłady wszystkich zmiennych. Wydają się wyglądać w porządku, bez żadnych wyraźnych anomalii. Dodatkowo widać tu, że zmienna celu nie jest zbalansowana - mamy więcej zer.

[4]:
```
X = df.drop("RainTomorrow", axis=1)
y = df[["RainTomorrow"]]
```

Żeby sprawdzić, czy `X` i `y` dobrze się przycięły można wyświetlić kilka pierwszych wierszy i sprawdzić, czy ramki mają odpowiedni kształt.

[5]:
```
X.head()
```

[5]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	Cloud9am	Cloud3pm	Temp9am	Temp3pm	RainToday
0	17.9	35.2	0.0	12.0	12.3	48.0	6.0	20.0	20.0	13.0	1006.3	1004.4	2.0	5.0	26.6	33.4	0
1	18.4	28.9	0.0	14.8	13.0	37.0	19.0	19.0	30.0	8.0	1012.9	1012.1	1.0	1.0	20.3	27.0	0
2	19.4	37.6	0.0	10.8	10.6	46.0	30.0	15.0	42.0	22.0	1012.3	1009.2	1.0	6.0	28.7	34.9	0
3	21.9	38.4	0.0	11.4	12.2	31.0	6.0	6.0	37.0	22.0	1012.7	1009.1	1.0	5.0	29.1	35.6	0
4	24.2	41.0	0.0	11.2	8.4	35.0	17.0	13.0	19.0	15.0	1010.7	1007.4	1.0	6.0	33.6	37.6	0

[6]:
```
y.head()
```

[6]:

	RainTomorrow
0	0
1	0
2	0
3	0
4	0

[7]:
```
y.describe()
```

[7]:

	RainTomorrow
count	56420.000000
mean	0.220259
std	0.414425

```
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000
```

Widać tutaj, że zbiór danych jest raczej przekrzywiony w stronę zera - mamy dysproporcję klas. Więcej wierszy pochodzi z dni, gdy nie padał deszcz. Wypada użyć argumentu `stratify` przy podziale, aby zachować odpowiednie proporcje klas.

[8]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,␣
↪test_size=0.2)
```

Pierwszym modelem, z którego skorzystam jest regresja logistyczna. W tym przypadku ustawiłem następujące hiperparametry: - `penalty = "l2"` - ustawienie regularyzacji na l2 - `C = 1.2` - ustawienie wagi regularyzacji na trochę łagodniejszą niż domyślna - `max_iter = 1000` - zwiększenie domyślnej maksymalnej liczby iteracji tego modelu

[9]:
```
logit = LogisticRegression(penalty="l2", C=1.2, max_iter=1000, random_state=42)
logit.fit(X_train, y_train)

logit_pred = logit.predict(X_test)

acc_logit = accuracy_score(y_test, logit_pred)
rec_logit = recall_score(y_test, logit_pred)
f1_logit = f1_score(y_test, logit_pred)

print("Accuracy: {:.3f}\nRecall: {:.3f}\nF1-score: {:.3f}".format(acc_logit,␣
↪rec_logit, f1_logit))
```

```
c:\users\mikołaj\appdata\local\programs\python\python38\lib\site-
packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

```
Accuracy: 0.853
Recall: 0.522
F1-score: 0.610
```

Kolejnym modelem jest las losowy. Parametry, które tu ustawiłem to: - `n_estimators=20` - liczba drzew, z których korzysta ten klasyfikator - `max_depth=4` - największa wysosość pojedynczego drzewa - `max_features=3` - ile najwięcej featerów może rozdzielać jedno drzewo

[10]:
```
rfc = RandomForestClassifier(n_estimators=20, max_depth=4,max_features=3,␣
↪random_state=42)

rfc.fit(X_train, y_train)

rfc_pred = rfc.predict(X_test)
```

```
acc_rfc = accuracy_score(y_test, rfc_pred)
rec_rfc = recall_score(y_test, rfc_pred)
f1_rfc = f1_score(y_test, rfc_pred)

print("Accuracy: {:.3f}\nRecall: {:.3f}\nF1-score: {:.3f}".format(acc_rfc,␣
 ↪rec_rfc, f1_rfc))
```

```
<ipython-input-10-9a33a092fb17>:3: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  rfc.fit(X_train, y_train)
```

```
Accuracy: 0.843
Recall: 0.369
F1-score: 0.508
```

Kolejnym klasyfikatorem jest model opierający się na Stochastic Gradient Descent. Przy tym modelu wyspecyfikowałem następujące hiperparametry: - `loss="hinge"` - określa funkcję straty - `penalty="l2"` - regularyzacja l2

[11]:
```
sgd = SGDClassifier(loss="hinge", penalty="l2", max_iter=1000)
sgd.fit(X_train, y_train)

sgd_pred = sgd.predict(X_test)

acc_sgd = accuracy_score(y_test, sgd_pred)
rec_sgd = recall_score(y_test, sgd_pred)
f1_sgd = f1_score(y_test, sgd_pred)

print("Accuracy: {:.3f}\nRecall: {:.3f}\nF1-score: {:.3f}".format(acc_sgd,␣
 ↪rec_sgd, f1_sgd))
```

```
c:\users\mikołaj\appdata\local\programs\python\python38\lib\site-
packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

```
Accuracy: 0.844
Recall: 0.362
F1-score: 0.506
```

Ostatnim modelem jest klasyfikator działający na SVM, zastosowałem w nim kernel `linear` - linowy.

[12]:
```
svc = svm.SVC(kernel="linear", random_state=42)
svc.fit(X_train, y_train)
svc_pred = svc.predict(X_test)
```

```
acc_svc = accuracy_score(y_test, svc_pred)
rec_svc = recall_score(y_test, svc_pred)
f1_svc = f1_score(y_test, svc_pred)

print("Accuracy: {:.3f}\nRecall: {:.3f}\nF1-score: {:.3f}".format(acc_svc,␣
 ↪rec_svc, f1_svc))
```

```
c:\users\mikołaj\appdata\local\programs\python\python38\lib\site-
packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

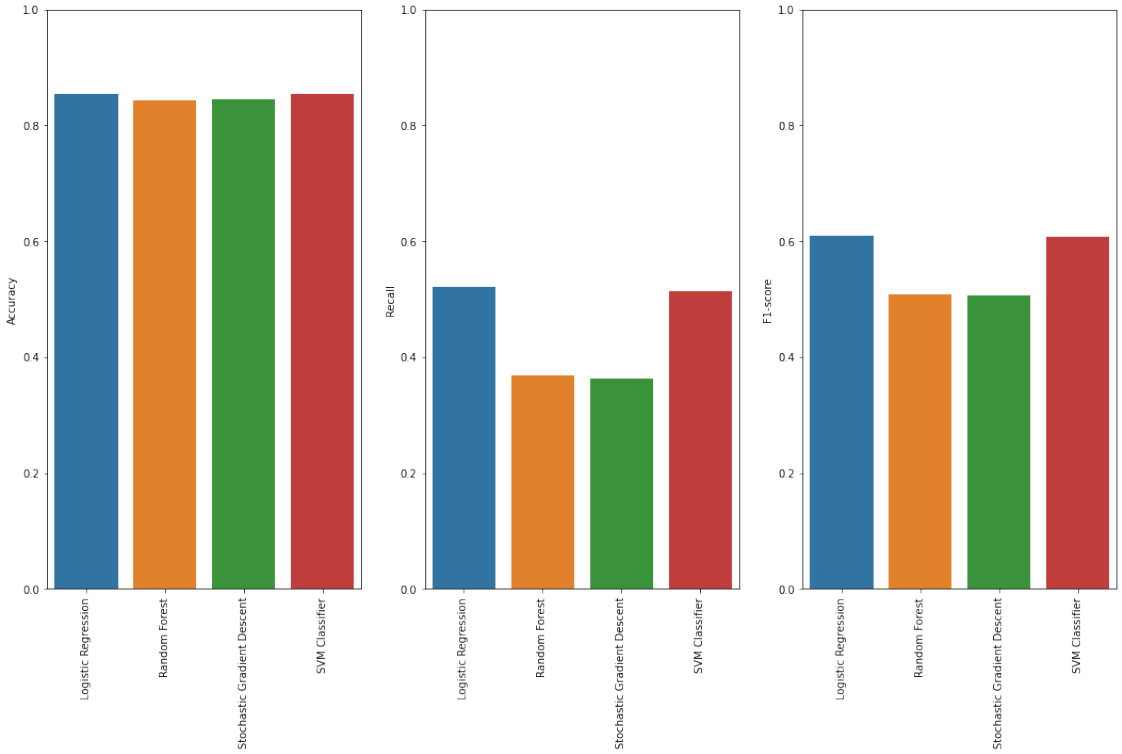
```
Accuracy: 0.854
Recall: 0.515
F1-score: 0.608
```

[13]:
```
res_arr = [
    [acc_logit, rec_logit, f1_logit],
    [acc_rfc, rec_rfc, f1_rfc],
    [acc_sgd, rec_sgd, f1_sgd],
    [acc_svc, rec_svc, f1_svc]
]

results = pd.DataFrame(res_arr,columns = ["Accuracy", "Recall", "F1-score"],␣
 ↪index=["Logistic Regression", "Random Forest", "Stochastic Gradient␣
 ↪Descent", "SVM Classifier"])

results
```

[13]:

	Accuracy	Recall	F1-score
Logistic Regression	0.853244	0.521932	0.610353
Random Forest	0.842875	0.369014	0.508456
Stochastic Gradient Descent	0.844470	0.362173	0.506329
SVM Classifier	0.853864	0.514688	0.608034

[14]:
```
fig, axs = plt.subplots(1,3, figsize=(18, 10))
sns.barplot(data=results, y="Accuracy",x=results.index,ax=axs[0])
sns.barplot(data=results, y="Recall",x=results.index,ax=axs[1])
sns.barplot(data=results, y="F1-score",x=results.index,ax=axs[2])
axs[0].tick_params(axis="x", rotation=90)
axs[1].tick_params(axis="x", rotation=90)
axs[2].tick_params(axis="x", rotation=90)
axs[0].set_ylim([0,1])
axs[1].set_ylim([0,1])
axs[2].set_ylim([0,1])

plt.show()
```



Wybór najlepszego spośród tych modeli nie jest jednoznaczny jednak na pierwszy rzut oka SVM wydaje się być najlepszy, ewentualnie regresja logistyczna. Wybór zależy jednak również od tego, co jest dla nas najbardziej wartościowe. Jeżeli kosztowne dla nas będzie przewidzenie dnia suchego, podczas gdy faktycznie będzie padał deszcz, to powinniśmy optymalizować `recall`. Jeśli zaś sytuacja jest odwrotna: najwięcej kosztuje przewidzenie deszczu, gdy go nie będzie, to model powinniśmy optymalizować pod względem `precision`. Miarą, która daje nam najwięcej informacji o całokształcie modelu jest F1, więc jeśli wszystkie pomyłki są tak samo kosztowne, można podejmować decyzje na jej podstawie.