milestone2

March 30, 2021

1 WUM - projekt 1 - feature engineering, wstępne modelowanie

Przewidywanie oceny końcoworocznej w portugalskich szkołach

Mikołaj Spytek, Artur Żółkowski

W tej części pracy nad projektem przygotowaliśmy odpowiednio zmienne, aby można było za ich pomocą nauczyć modele.

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import requests

from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import FunctionTransformer
from sklearn.metrics import accuracy_score

pd.set_option("display.max_columns", None, "display.width", 1000)
```

[2]:
```
#download the data
r = requests.get('https://api.apispreadsheets.com/api/dataset/school-grades/')
data = r.json()
df = pd.DataFrame(data['data'])

df.head()
```

[2]:
```
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  reason
guardian  traveltime  studytime  failures schoolsup famsup paid activities
nursery higher internet romantic  famrel  freetime  goout  Dalc  Walc  health
absences  G1  G2  G3
0     GP   F   18       U     GT3       A     4     4  at_home   teacher  course
mother           2          2         0       yes     no   no         no     yes
```

```
yes       no       no       4         3      4     1     1       3         4   0
11  11
1     GP   F   17       U     GT3       T     1     1  at_home     other  course
father           1          2         0        no    yes   no         no      no
yes      yes       no       5         3      3     1     1       3         2   9
11  11
2     GP   F   15       U     LE3       T     1     1  at_home     other   other
mother           1          2         0       yes     no   no         no     yes
yes      yes       no       4         3      2     2     3       3         6  12
13  12
3     GP   F   15       U     GT3       T     4     2   health  services    home
mother           1          3         0        no    yes   no        yes     yes
yes      yes      yes       3         2      2     1     1       5         0  14
14  14
4     GP   F   16       U     GT3       T     3     3    other     other    home
father           1          2         0        no    yes   no         no     yes
yes       no       no       4         3      2     1     2       5         0  11
13  13
```

W ramce danych, na której pracujemy znajdują się kolumny `G1` i `G2`, oznaczające oceny z poprzednich semestrów. Zdecydowaliśmy, że nie będziemy ich wykorzystywać, do predykcji zmiennej `G3`, gdyż taki model jest mało użyteczny, a uzyskanie dobrych wyników nie jest trudne. Postanowiliśmy jednak, że użyjemy tych zmiennych, aby wyznaczyć pewnego rodzaju baseline - jeśli za pomocą złożonego modelu korzystającego z innych zmiennych uda nam się osiągnąć wynik, jaki dostajemy na prostym modelu, ale ze zmiennymi `G1` i `G2`, to będziemy zadowoleni.

[3]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# model wytrenowany tylko na zmiennych G1, G2
X_simple = pd.DataFrame()
y_simple = pd.DataFrame()

X_simple[["G1"]] = df[["G1"]].copy()
X_simple[["G2"]] = df[["G2"]].copy()
y_simple[["G3"]] = df[["G3"]].copy()


X_simple_train, X_simple_test, y_simple_train, y_simple_test =␣
 ↪train_test_split(X_simple, y_simple, test_size = 0.2, random_state=42)

simple_model = LinearRegression()
simple_model.fit(X_simple_train, y_simple_train)
simple_pred = simple_model.predict(X_simple_test)

simple_rmse = np.sqrt(mean_squared_error(y_simple_test, simple_pred))
```

```
print("RMSE (of simple model with G1 and G2) = {:.3f}".format(simple_rmse))
```

```
RMSE (of simple model with G1 and G2) = 1.169
```

Naszym kolejnym pomysłem było sprawdzenie, jakie wyniki będzie osiągał model wytrenowany na wszystkich dostępnych zmiennych, oprócz wyżej wymienionych kolumn. Nie można jednak było ich użyć od razu. Część z nich jest typu kategorycznego, i wymaga zakodowania. Niektóre zaś są podane w skali od 1 do 5 - więc przeskalujemy je dzieląc przez maksimum tak, aby przyjmowały wartości z zakresu $[0, 1]$.

[4]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.naive_bayes import BernoulliNB
from sklearn.ensemble import AdaBoostClassifier

enc = OneHotEncoder(drop="if_binary", sparse=False)

categorical_variables = enc.fit_transform(df.iloc[:,␣
↪[0,1,3,4,5,8,9,10,11,15,16,17,18,19,20,21,22]])

variables_1_5 = df.iloc[:, [6,7,12,13,23,24,25,26,27,28,29]] / 5

not_changed = df.iloc[:, [2,14,30]]

temp = np.append(categorical_variables, variables_1_5, axis=1)
X_all = np.append(temp, not_changed, axis=1)
y = df["G3"]

X_train, X_test, y_train, y_test = train_test_split(X_all, y, test_size=0.2,␣
↪random_state=42)

lr = LinearRegression()

lr.fit(X_train,y_train)
lr_pred = lr.predict(X_test)

lr_rmse = np.sqrt(mean_squared_error(y_test, lr_pred))

rf = RandomForestClassifier(random_state=42)

rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)

rf_rmse = np.sqrt(mean_squared_error(y_test, rf_pred))

s = svm.SVR()
```

```
s.fit(X_train, y_train)
s_predict = s.predict(X_test)

s_rmse = np.sqrt(mean_squared_error(y_test, s_predict))

gnb = BernoulliNB()

gnb.fit(X_train, y_train)
gnb_predict = gnb.predict(X_test)

gnb_rmse = np.sqrt(mean_squared_error(y_test, gnb_predict))

ada = AdaBoostClassifier(n_estimators=100)

ada.fit(X_train, y_train)
ada_pred = ada.predict(X_test)

ada_rmse = np.sqrt(mean_squared_error(y_test, ada_pred))


print("RMSE (of linear regression) = {:.3f}".format(lr_rmse))
print("RMSE (of random forest) = {:.3f}".format(rf_rmse))
print("RMSE (of SVR) = {:.3f}".format(s_rmse))
print("RMSE (of bernouli naive bayes) = {:.3f}".format(gnb_rmse))
print("RMSE (of adaboost) = {:.3f}".format(ada_rmse))


## Powinniśmy jeszcze skorzysta z jakiegś prostego baselinu, czy te wyniki są w␣
 ↪ogóle dobre - niech to będzie średnia

sr = y_train.mean()
y_baseline = [sr for i in range(len(y_test))]
rmse_baseline = np.sqrt(mean_squared_error(y_test, y_baseline))
print("RMSE (of baseline) = {:.3f}".format(rmse_baseline))
```

```
RMSE (of linear regression) = 1.786
RMSE (of random forest) = 1.897
RMSE (of SVR) = 1.697
RMSE (of bernouli naive bayes) = 3.304
RMSE (of adaboost) = 2.490
RMSE (of baseline) = 3.173
```

Jak widzimy osiągamy całkiem dobre wyniki po tak podstawowym feature engineeringu. Postanowiliśmy jednak sprawdzić, czy na mniejszej ilości cech nie osiągniemy lepszych wyników. Usunęliśmy zmienne mocno skorelowane (przyjrzeliśmy się macierzy korelacji z naszego EDA) oraz niektóre, które naszym zdaniem mają niewielki wpływ.

[5]:
```
enc = OneHotEncoder(drop="if_binary", sparse=False)

categorical_variables = enc.fit_transform(df.iloc[:, [0,1,5,9,17,19,20,21]])

variables_1_5 = df.iloc[:, [6,7,12,13,25,28,29]] / 5

not_changed = df.iloc[:, [2,30]]

temp = np.append(categorical_variables, variables_1_5, axis=1)
X_all = np.append(temp, not_changed, axis=1)
y = df["G3"]

X_train, X_test, y_train, y_test = train_test_split(X_all, y, test_size=0.2,␣
 ↪random_state=42)

lr = LinearRegression()

lr.fit(X_train,y_train)
lr_pred = lr.predict(X_test)

lr_rmse = np.sqrt(mean_squared_error(y_test, lr_pred))

rf = RandomForestClassifier(random_state=42)

rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)

rf_rmse = np.sqrt(mean_squared_error(y_test, rf_pred))

s = svm.SVR()

s.fit(X_train, y_train)
s_predict = s.predict(X_test)

s_rmse = np.sqrt(mean_squared_error(y_test, s_predict))

gnb = BernoulliNB()

gnb.fit(X_train, y_train)
gnb_predict = gnb.predict(X_test)

gnb_rmse = np.sqrt(mean_squared_error(y_test, gnb_predict))

ada = AdaBoostClassifier(n_estimators=100)

ada.fit(X_train, y_train)
ada_pred = ada.predict(X_test)
```

```
ada_rmse = np.sqrt(mean_squared_error(y_test, ada_pred))

print("RMSE (of linear regression with less variables) = {:.3f}".
 ↪format(lr_rmse))
print("RMSE (of random forest with less variables) = {:.3f}".format(rf_rmse))
print("RMSE (of SVR) = {:.3f}".format(s_rmse))
print("RMSE (of bernouli naive bayes) = {:.3f}".format(gnb_rmse))
print("RMSE (of adaboost) = {:.3f}".format(ada_rmse))


sr = y_train.mean()
y_baseline = [sr for i in range(len(y_test))]
rmse_baseline = np.sqrt(mean_squared_error(y_test, y_baseline))
print("RMSE (of baseline) = {:.3f}".format(rmse_baseline))
```

```
RMSE (of linear regression with less variables) = 1.731
RMSE (of random forest with less variables) = 1.871
RMSE (of SVR) = 1.708
RMSE (of bernouli naive bayes) = 3.357
RMSE (of adaboost) = 2.490
RMSE (of baseline) = 3.173
```

Po takich modyfikacjach otrzymaliśmy trochę lepszy wynik przy prostych modelach, przynajmniej patrząc na tę miarę, którą wybraliśmy. Na modelach bardziej skomplikowanych miara błędu pozostaje na podobnym poziomie

2 Część klasyfikacyjna

W poleceniu zadania (na repo przedmiotu) znaleźliśmy informację, że pierwszy projekt ma dotyczyć klasyfikacji. Można to zadanie potraktować jako 20-sto klasową klasyfikację, lecz nie jest to naszym zdaniem najlepsze podejście do tego problemu. Patrząc na inne tematy tego projektu klasyfikacja w tamtych problemach jest binarna, i bardziej naturalna.

Problemem przy traktowaniu tego jako klasyfikację, jest również miara, której mielibyśmy używać do ewaluacji modelu. Gdyby miało to być po prostu accuracy, to pomyłka oceny o 1 traktowana by była tak samo, jak pomyłka o 19.

Postanowiliśmy przewidywać czy dana osoba zdała, tzn. czy ocena jest `>=10`.

Na tym etapie dodaliśmy kolejne techniki feature engineeringu. Zmienne które były w skali od 1 do 5, przeskalowaliśmy dzieląc przez 5, tak, aby dostawać wartości z przedziału $[0, 1]$

Stwierdziliśmy też, że warto dodać zmienną, która mówi o tym, czy ktoś wcześniej nie zdał. Oraz taką, która określa, czy ktoś ma więcej niż 5 nieobecności.

[6]:
```
df['pass']= np.where(df['G3']<10, 0, 1)
df['Pedu'] = df['Fedu'] + df['Medu']
df["genrel"] = df["sex"]+df["romantic"]
df[["Pedu"]]  = df[["Pedu"]] / 5
```

```
df[["studytime"]]  = df[["studytime"]] / 5
df[["age"]] = df[["age"]] # /22
df[["health"]] = df[["health"]]/5
df[["goout"]] = df[["goout"]]/5
df[["Dalc"]] = df[["Dalc"]]/5

df[["absences"]] = df[["absences"]]/df['absences'].max()
df[["absenc"]] = np.where(df['absences']<5, 0, 1)

fail = pd.DataFrame([(1 if a > 0 else 0) for a in df['failures']],␣
↪columns=["fail"])

df = df.join(fail)
```

Następnie wybraliśmy według nas najistotniejsze zmienne dzieląc je na kategoryczne i numeryczne.

[7]:
```
cat_features = ["Mjob", "higher", "internet", "romantic", "address", "reason",
                "activities", "famsup", "schoolsup", "school"]
num_features = ["Pedu", "health", "studytime", "Dalc", "Walc", "traveltime",␣
↪"freetime", "goout", "age", "fail"]
```

[8]:
```
# Separate features and predicted value
features = num_features + cat_features
X = df.drop(["pass"], axis=1)[features]
y = df["pass"]
```

[9]:
```
# Preprocess numerical feats:
num_transformer = SimpleImputer(strategy="constant")

# Preprocessing for categorical features:
cat_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="constant", fill_value="Unknown")),
    ("onehot", OneHotEncoder(handle_unknown='ignore'))])

# Bundle preprocessing for numerical and categorical features:
preprocessor = ColumnTransformer(transformers=[("num", num_transformer,␣
↪num_features),
                                               ("cat", cat_transformer,␣
↪cat_features)],
                                 remainder = 'passthrough')
```

[10]:
```
rf_model_enh = RandomForestClassifier(n_estimators=10,
                                      max_features=0.4,
                                      min_samples_split=2,
                                      n_jobs=-1,
                                      random_state=33)
```

```
model_pipe = Pipeline(steps=[('preprocessor', preprocessor),
                             ('model', rf_model_enh)])

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2, random_state=42)

model_pipe.fit(X_train, y_train)
```

[10]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('num',
SimpleImputer(strategy='constant'),
                                                  ['Pedu', 'health',
                                                   'studytime', 'Dalc', 'Walc',
                                                   'traveltime', 'freetime',
                                                   'goout', 'age', 'fail']),
                                                 ('cat',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(fill_value='Unknown',
 strategy='constant')),
                                                                  ('onehot',
OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['Mjob', 'higher', 'internet',
                                                   'romantic', 'address',
                                                   'reason', 'activities',
                                                   'famsup', 'schoolsup',
                                                   'school'])])),
                ('model',
                 RandomForestClassifier(max_features=0.4, n_estimators=10,
                                        n_jobs=-1, random_state=33))])
```

[11]:
```
y_predict = model_pipe.predict(X_test)
```

[12]:
```
accuracy_score(y_test, y_predict)
```

[12]:
```
0.8923076923076924
```

Po tak dobranych zmiennych wytrenowaliśmy model Random Forest Classifier. Jak widizmy, udało nam się osiągnąć dobry rezultat przy accuracy wynoszącym ponad 89%.

Następnie korzystając z biblioteki dalex chcieliśmy sprawdzić czy nasz model nie jest zvyt stronniczy, lub czy nie występują w nim żadne zmienne dominujące mogące samodzielnie zdecydować o wartości predykcji.

[13]:
```
import dalex as dx

explainer = dx.Explainer(model_pipe, X_train, y_train)
```

```
Preparation of a new explainer is initiated

  -> data              : 519 rows 20 cols
  -> target variable   : Parameter 'y' was a pandas.Series. Converted to a
numpy.ndarray.
  -> target variable   : 519 values
  -> model_class       : sklearn.ensemble._forest.RandomForestClassifier
(default)
  -> label             : Not specified, model's class short name will be used.
(default)
  -> predict function  : <function yhat_proba_default at 0x000001C14202E1F0>
will be used (default)
  -> predict function  : Accepts only pandas.DataFrame, numpy.ndarray causes
problems.
  -> predicted values  : min = 0.0, mean = 0.842, max = 1.0
  -> model type        : classification will be used (default)
  -> residual function : difference between y and yhat (default)
  -> residuals         : min = -0.6, mean = -0.00565, max = 0.5
  -> model_info        : package sklearn

A new explainer has been created!
```

Przeanalizujmy teraz przykładową predykcję 1 i 0 (zdał, nie zdał). Weźmy zatem pierwsze wystąpienia tych wartości z naszego zbioru testowego (odpowiednio obserwacja 220 i 131)

[14]:
```
y_test.head(10)
```

[14]:
```
636    1
220    1
594    1
429    1
72     1
448    1
181    1
131    0
231    1
277    1
Name: pass, dtype: int32
```

[15]:
```
X_test.iloc[1,:]
```

[15]:
```
Pedu              0.8
health            0.8
studytime         0.4
Dalc              0.4
Walc                2
traveltime          1
freetime            3
```

```
goout              0.6
age                 16
fail                 0
Mjob          services
higher              yes
internet            yes
romantic             no
address               U
reason           course
activities           no
famsup              yes
schoolsup            no
school               GP
Name: 220, dtype: object
```

[16]:
```
pp_1 = explainer.predict_parts(X_test.iloc[1,:])
pp_1
```

[16]:
```
<dalex.predict_explanations._break_down.object.BreakDown at 0x1c14206c280>
```

[17]:
```
pp_1.plot()
```

Jak możemy zauważyć nasz model prawidłowo przewidział wartość predykcyjną. Nie widzimy tutaj żdanych dominujących czynników wpływających na wynik. W tym przypadku największe znaczenie miało to, że uczeń nie miał wcześniejszych porażek oraz to, że chodził do szkoły oznaczonej jako GP.

[18]:
```
pp_0 = explainer.predict_parts(X_test.iloc[7,:])
pp_0
```

[18]:
```
<dalex.predict_explanations._break_down.object.BreakDown at 0x1c14206c1f0>
```

[19]:
```
pp_0.plot()
```

Również w drugim przypadku algorytm poprawnie rozpoznał czy uczeń zda. W tym przypadku największe znaczenie miała poprzednia porażka oraz brak chęci dalszej edukacji.