Final_modeling

April 20, 2021

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import category_encoders as ce
from sklearn import metrics
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
np.random.seed(123)
```

0.1 Encoding i transformacje

[2]:
```python
grades_df = pd.read_csv('school_grades_dataset.csv')
grades_df = grades_df[grades_df['G3'] != 0]

cat_cols = ['school', 'sex', 'address', 'famsize', 'Mjob', 'Fjob', 'reason',␣
 ↪'guardian', 'Pstatus', 'sex', 'school']
bin_cols = ['famsup', 'activities', 'nursery', 'internet', 'romantic',␣
 ↪'higher', 'paid', 'schoolsup']

grades_df_new = grades_df.drop(columns = (cat_cols + bin_cols))

for i in cat_cols:
    means = grades_df.groupby(i)['G3'].mean()
    grades_df_new[i] = grades_df[i].map(means)

for i in bin_cols:
    encoder = ce.OrdinalEncoder(mapping = [{'col': i, 'mapping': {'yes': 1,␣
 ↪'no': 0}},])
    grades_df_new[i] = encoder.fit_transform(grades_df)[i]

grades_df_new['result'] = pd.cut(grades_df_new['G3'],
                                 bins=[-1, 9, 11, 13, 15, 21],
                                 labels=['1', '2', '3', '4', '5'])
```

[3]:
```
from sklearn.ensemble import RandomForestClassifier
import xgboost
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import RFE

# funkca do mierzenia poprawności

def simple_models(X_train, X_test, y_train, y_test):

    lr = LogisticRegression(random_state=1, max_iter=100)
    lr.fit(X_train, y_train)
    print(f'Logistic regression accuracy: {lr.score(X_test, y_test)}')
    selector = RFE(lr, n_features_to_select=7, step=1)
    selector = selector.fit(X_train, y_train)
    print(f'Logistic regression accuracy: {selector.score(X_test, y_test)}. (po␣
 ↪zastosowaniu RFE)')

    tree_model = DecisionTreeClassifier()
    tree_model.fit(X_train, y_train)
    print(f'Decision Tree accuracy: {tree_model.score(X_test, y_test)}')
    selector = RFE(tree_model, n_features_to_select=7, step=1)
    selector = selector.fit(X_train, y_train)
    print(f'Decision Tree accuracy: {selector.score(X_test, y_test)}. (po␣
 ↪zastosowaniu RFE)')


    rf = RandomForestClassifier()
    rf.fit(X_train, y_train)
    print(f'Random Forest accuracy: {rf.score(X_test, y_test)}')
    selector = RFE(rf, n_features_to_select=7, step=1)
    selector = selector.fit(X_train, y_train)
    print(f'Random Forest accuracy: {selector.score(X_test, y_test)}. (po␣
 ↪zastosowaniu RFE)')


    #svc = SVC()
    #svc.fit(X_train,y_train)
    #print(f'SVC accuracy: {svc.score(X_test, y_test)}')
    #selector = RFE(svc, n_features_to_select=7, step=1)
    #selector = selector.fit(X_train, y_train)
    #print(f'SVC accuracy: {selector.score(X_test, y_test)}. (po zastosowaniu␣
 ↪RFE)')

    xgb = xgboost.XGBClassifier(eval_metric = 'merror')
    xgb.fit(X_train,y_train)
```

```
    print(f'XGBoost accuracy: {xgb.score(X_test, y_test)}')
    selector = RFE(xgb, n_features_to_select=7, step=1)
    selector = selector.fit(X_train, y_train)
    print(f'XGBoost accuracy: {selector.score(X_test, y_test)}. (po␣
↪zastosowaniu RFE)')
```

0.2 Klasyfikacja konkretnego wyniku

Sprawdzimy możliwość przewidywania oceny końcowej na dwa sposoby: przewidywanie dokładniej oceny oraz przewidywnanie jej przedziału (kubełki 0-9, 10-11, 12-13, 14-15, 16-21).

Użyjemy też różnych sposobówprzewidywania to znaczy będziemy używać G1 i G2, które jest mocno skorelowane z G3 lub też nie.

0.2.1 Dane łącznie z G1 i G2

[4]:
```
X = grades_df_new.drop(['G3', 'result'], axis = 1)
y = grades_df_new['G3']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
```

[5]:
```
simple_models(X_train, X_test, y_train, y_test)
```

```
Logistic regression accuracy: 0.3298429319371728
Logistic regression accuracy: 0.3717277486910995. (po zastosowaniu RFE)
Decision Tree accuracy: 0.3193717277486911
Decision Tree accuracy: 0.27225130890052357. (po zastosowaniu RFE)
Random Forest accuracy: 0.39267015706806285
Random Forest accuracy: 0.32460732984293195. (po zastosowaniu RFE)
XGBoost accuracy: 0.35602094240837695
XGBoost accuracy: 0.39790575916230364. (po zastosowaniu RFE)
```

Jak widać modele radzą sobie bardzo słabo z odgadnięciem konkretnej liczby punktów zdobytej przez ucznia.

0.2.2 Dane bez G1 i G2

[6]:
```
X = grades_df_new.drop(['G1', 'G2', 'G3', 'result'], axis = 1)
y = grades_df_new['G3']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
```

[7]:
```
simple_models(X_train, X_test, y_train, y_test)
```

```
Logistic regression accuracy: 0.11518324607329843
Logistic regression accuracy: 0.16230366492146597. (po zastosowaniu RFE)
Decision Tree accuracy: 0.08900523560209424
Decision Tree accuracy: 0.10471204188481675. (po zastosowaniu RFE)
Random Forest accuracy: 0.17801047120418848
Random Forest accuracy: 0.13612565445026178. (po zastosowaniu RFE)
XGBoost accuracy: 0.16230366492146597
XGBoost accuracy: 0.1518324607329843. (po zastosowaniu RFE)
```

Bez tych danych jest w ogóle tragicznie.

0.2.3 Regresja liniowa

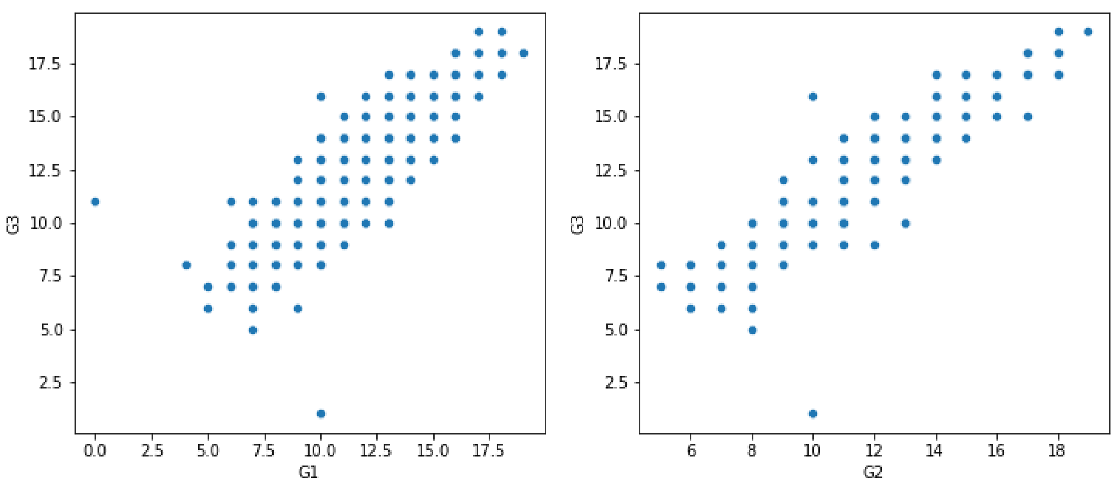
Użyjmy regresji liniowej do przywidywania wyników na podstawie samych G1 i G2, które są mocno skorelowane z G3

[8]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (12, 5))

sns.scatterplot(data = grades_df, x = 'G1', y = 'G3', ax = ax1)
sns.scatterplot(data = grades_df, x = 'G2', y = 'G3', ax = ax2)

plt.show()

# liniowa zalezcnosc miedzy G1, G2, i G3
```



[9]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

X = grades_df_new[['G1', 'G2']]
```

```
y = grades_df_new['G3']
linear_reg = LinearRegression()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
 ↪random_state = 1)
linear_reg.fit(X_train, y_train)
y_test_predicted = linear_reg.predict(X_test)

print(f'RMSE: {mean_squared_error(y_test, y_test_predicted, squared = False)}')
print(f'R-squared: {linear_reg.score(X_test, y_test)}')
```

```
RMSE: 0.9450989370465289
R-squared: 0.8596527332651904
```

Trzeba pamiętać, że regresja liniowa przewiduje wartości ciągłe, spróbujemy zatem zaokrągliz wynik i sprawdźmy ile odpowiedzi zostało odgadniętych:

[10]:
```
print(f'Odesetek dobrze predykowanych zaokrąglonych wyników:\
      {(linear_reg.predict(X_test).round() == y_test).sum() / len(y_test)}')
```

```
Odesetek dobrze predykowanych zaokrąglonych wyników:      0.4816753926701571
```

Nie jest to zachwycająca odpowiedź, ale lepsza od modeli klasyfikujących.

0.3 Klasyfikacja przedziały wyniku

0.3.1 Dane bez G1 i G2

[11]:
```
X = grades_df_new.drop(['G1', 'G2', 'G3', 'result'], axis = 1)
y = grades_df_new['result']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

[12]:
```
simple_models(X_train, X_test, y_train, y_test)
```

```
Logistic regression accuracy: 0.27225130890052357
Logistic regression accuracy: 0.3089005235602094. (po zastosowaniu RFE)
Decision Tree accuracy: 0.2879581151832461
Decision Tree accuracy: 0.25654450261780104. (po zastosowaniu RFE)
Random Forest accuracy: 0.33507853403141363
Random Forest accuracy: 0.28272251308900526. (po zastosowaniu RFE)
XGBoost accuracy: 0.2931937172774869
XGBoost accuracy: 0.2617801047120419. (po zastosowaniu RFE)
```

0.3.2 Dane z G1 i G2

[13]:
```
X = grades_df_new.drop(['G3', 'result'], axis = 1)
y = grades_df_new['result']

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y,␣
↪test_size=0.3, random_state=42)
```

[14]:
```
simple_models(X_train, X_test, y_train, y_test)
```

```
Logistic regression accuracy: 0.6544502617801047
Logistic regression accuracy: 0.6910994764397905. (po zastosowaniu RFE)
Decision Tree accuracy: 0.5602094240837696
Decision Tree accuracy: 0.6178010471204188. (po zastosowaniu RFE)
Random Forest accuracy: 0.6858638743455497
Random Forest accuracy: 0.680628272251309. (po zastosowaniu RFE)
XGBoost accuracy: 0.6649214659685864
XGBoost accuracy: 0.6492146596858639. (po zastosowaniu RFE)
```

[15]:
```
from sklearn.model_selection import GridSearchCV

C = np.arange(0, 2, 0.2)
class_weight = [None, 'balanced']
fit_intercept = [True, False]
l1_ratio = np.arange(0, 1, 0.1)
solver = ["newton-cg", "sag", "saga", "lbfgs", "liblinear"]

lr = LogisticRegression(random_state=1, max_iter=100)

param_grid = dict(C = C, class_weight = class_weight, fit_intercept =␣
↪fit_intercept, l1_ratio = l1_ratio, solver = solver)
grid = GridSearchCV(estimator=lr, param_grid=param_grid, cv = 3, n_jobs=-1)
grid_result = grid.fit(X_train, y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.717718 using {'C': 0.2, 'class_weight': None, 'fit_intercept': True,
'l1_ratio': 0.0, 'solver': 'newton-cg'}
```

[16]:
```
best_model=grid_result.best_estimator_
score_max = 0

for i in range(30):
    selector = RFE(best_model, n_features_to_select=i+1, step=1)
    selector = selector.fit(X_train, y_train)
    if (selector.score(X_test, y_test) > score_max):
        feature_number = i+1
```

```
        selector_best = selector
        score_max = selector.score(X_test, y_test)
```

[17]:
```
print(f'Wynik dla regresji logistycznej: {selector_best.score(X_test, y_test)}.␣
↪(po zastosowaniu RFE dla {feature_number} zmiennych)')
```

```
Wynik dla regresji logistycznej: 0.7329842931937173. (po zastosowaniu RFE dla 9
zmiennych)
```

[18]:
```
y_pred = selector_best.predict(X_test)

from sklearn.metrics import f1_score
print(f'F1-score: {f1_score(y_test, y_pred, average = "weighted")}')

from sklearn.metrics import precision_score
print(f'Precision: {precision_score(y_test, y_pred, average = "weighted")}')

from sklearn.metrics import recall_score
print(f'Recall: {recall_score(y_test, y_pred, average = "weighted")}')
```

```
F1-score: 0.7272386623852753
Precision: 0.7345616330948538
Recall: 0.7329842931937173
```

[19]:
```
rf = RandomForestClassifier()

criterion = ['gini', 'balanced']
class_weight = ['balanced', 'balanced_subsample']
max_depth = [3, 4, 5]


param_grid = dict(criterion=criterion, class_weight=class_weight,␣
↪max_depth=max_depth)
grid = GridSearchCV(estimator=rf, param_grid=param_grid, cv = 3, n_jobs=-1)
grid_result = grid.fit(X_train, y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))

best_model=grid_result.best_estimator_
score_max = 0

for i in range(30):
    selector = RFE(best_model, n_features_to_select=i+1, step=1)
    selector = selector.fit(X_train, y_train)
    if (selector.score(X_test, y_test) > score_max):
        feature_number = i+1
        selector_best = selector
```

```
        score_max = selector.score(X_test, y_test)

y_pred = selector_best.predict(X_test)

from sklearn.metrics import f1_score
print(f'F1-score: {f1_score(y_test, y_pred, average = "weighted")}')

from sklearn.metrics import precision_score
print(f'Precision: {precision_score(y_test, y_pred, average = "weighted")}')

from sklearn.metrics import recall_score
print(f'Recall: {recall_score(y_test, y_pred, average = "weighted")}')
```

```
Best: 0.744837 using {'class_weight': 'balanced_subsample', 'criterion': 'gini',
'max_depth': 5}
F1-score: 0.7279610573294034
Precision: 0.7364131716510501
Recall: 0.7329842931937173
```

[20]:
```
print(f'Wynik dla lasu losowego: {selector_best.score(X_test, y_test)}. (po␣
↪zastosowaniu RFE dla {feature_number} zmiennych)')
```

```
Wynik dla lasu losowego: 0.7329842931937173. (po zastosowaniu RFE dla 23
zmiennych)
```