spytek_zolkowski_projekt1_final

April 20, 2021

```python
[1]: import pandas as pd
     import numpy as np
     import requests

     from sklearn.model_selection import train_test_split, KFold, cross_validate,
      ↪cross_val_score
     from sklearn.pipeline import Pipeline
     from sklearn.compose import ColumnTransformer
     from sklearn.preprocessing import LabelEncoder, OneHotEncoder
     from sklearn.preprocessing import PolynomialFeatures
     from sklearn.feature_selection import SelectKBest, chi2, mutual_info_classif,
      ↪RFE, SelectFromModel
     from sklearn.impute import SimpleImputer
     from sklearn.preprocessing import FunctionTransformer
     from sklearn.metrics import accuracy_score
     from sklearn.metrics import mean_squared_error

     from sklearn.linear_model import LogisticRegression
     from sklearn.linear_model import LinearRegression
     from sklearn.ensemble import GradientBoostingRegressor
     from sklearn.svm import SVR
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.ensemble import RandomForestRegressor


     import dalex as dx

     import warnings
     warnings.filterwarnings('ignore')

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

0.0.1 Wczytanie danych

```python
[2]: 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
```

0.1 Podejście 1. - klasyfikacja

Dobór zmiennych i ich przetworzenie

[3]:
```python
df = pd.DataFrame(data['data'])

df['pass']= np.where(df['G3']<10, 0, 1)

### Stworzenie nowych kolumn korzystając z dostępnych danych
df['Pedu'] = df['Fedu'] + df['Medu']
df["genrel"] = df["sex"]+df["romantic"]
df["Alc"] = (df["Dalc"]+df["Walc"]) / 10
df[["absenc"]] = np.where(df['absences']<8, 0, 1)
fail = pd.DataFrame([(1 if a > 0 else 0) for a in df['failures']],
 ↪columns=["fail"])
df = df.join(fail)

### Przeskalowanie kolumn
df[["Pedu"]]   = df[["Pedu"]] /df['Pedu'].max()
df[["studytime"]]  = df[["studytime"]] /df['studytime'].max()
```

```python
df[["age"]] = df[["age"]] /df['age'].max()
df[["health"]] = df[["health"]]/df['health'].max()
df[["goout"]] = df[["goout"]]/df['goout'].max()
df[["freetime"]]  = df[["freetime"]] /df['freetime'].max()
df[["Dalc"]] = df[["Dalc"]]/df['Dalc'].max()
df[["absences"]] = df[["absences"]]/df['absences'].max()
```

```python
[4]: cat_features = ["Mjob", "higher", "genrel", "address", "reason", "school",
     →'internet']
     num_features = ["Pedu", "studytime", "goout", "age", "fail"]
```

```python
[5]: features = num_features + cat_features
     X = df.drop(["pass"], axis=1)[features]
     y = df["pass"]
```

```python
[6]: # 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')
```

```python
[7]: 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)
```

```
[7]: Pipeline(steps=[('preprocessor',
                      ColumnTransformer(remainder='passthrough',
```

```
                                          transformers=[('num',
     SimpleImputer(strategy='constant'),
                                                 ['Pedu', 'studytime', 'goout',
                                                  'age', 'fail']),
                                               ('cat',
                                                Pipeline(steps=[('imputer',
     SimpleImputer(fill_value='Unknown',
      strategy='constant')),
                                                               ('onehot',
     OneHotEncoder(handle_unknown='ignore'))]),
                                                 ['Mjob', 'higher', 'genrel',
                                                  'address', 'reason',
                                                  'school', 'internet'])])),
                          ('model',
                           RandomForestClassifier(max_features=0.4, n_estimators=10,
                                                  n_jobs=-1, random_state=33))])
```

[8]:
```
y_predict = model_pipe.predict(X_test)
accuracy_score(y_test, y_predict)
```

[8]: 0.8615384615384616

0.2 Podejście 2. - regresja

0.2.1 Feature engineering

[9]:
```
enc = OneHotEncoder(drop="if_binary", sparse=False)

alldf = enc.fit_transform(df.iloc[:,␣
 ↪[0,1,3,4,5,8,9,10,11,15,16,17,18,19,20,21,22]])
non_encoded = df.iloc[:, [2,14,29]]
scaled = df.iloc[:, [6,7,12,13,23,24,25,26,27,28]]/5

non_encoded.iloc[:,0] = non_encoded.iloc[:,0]/22
non_encoded.iloc[:,1] = non_encoded.iloc[:,1]/max(non_encoded.iloc[:,1])
non_encoded.iloc[:,2] = non_encoded.iloc[:,2]/max(non_encoded.iloc[:,2])



X_all = np.append(alldf, non_encoded, axis=1)
X_all = np.append(X_all, scaled, axis=1)

encoded_names = enc.get_feature_names(input_features= df.iloc[:,␣
 ↪[0,1,3,4,5,8,9,10,11,15,16,17,18,19,20,21,22]].columns)
enc_names_list = encoded_names.tolist() + non_encoded.columns.tolist()+ scaled.
 ↪columns.tolist()
enc_names_list
```

```python
X_all = pd.DataFrame(X_all, columns=enc_names_list)
y = df[["G3"]]

X_all
```

[9]:
```
     school_MS  sex_M  address_U  famsize_LE3  Pstatus_T  Mjob_at_home
Mjob_health  Mjob_other  Mjob_services  Mjob_teacher  Fjob_at_home  Fjob_health
Fjob_other  Fjob_services  Fjob_teacher  reason_course  reason_home
reason_other  reason_reputation  guardian_father  guardian_mother
guardian_other  schoolsup_yes  famsup_yes  paid_yes  activities_yes  nursery_yes
higher_yes  internet_yes  romantic_yes       age  failures  absences  Medu  Fedu
traveltime  studytime  famrel  freetime  goout  Dalc  Walc  health
0          0.0    0.0        1.0          0.0        0.0           1.0
0.0          0.0            0.0          0.0           0.0           0.0
0.0            0.0            1.0            1.0          0.0            0.0
0.0            0.0              1.0          0.0            1.0        0.0
0.0            0.0              1.0          1.0          0.0            0.0
0.037190  0.000000    0.1250    0.8    0.8           0.4        0.10    0.8
0.12   0.16  0.04   0.2    0.12
1          0.0    0.0        1.0          0.0        1.0           1.0
0.0          0.0            0.0          0.0           0.0           0.0
1.0          0.0            0.0            1.0          0.0            0.0
0.0            1.0              0.0          0.0            0.0        1.0
0.0            0.0              0.0          1.0          1.0            0.0
0.035124  0.000000    0.0625    0.2    0.2           0.2        0.10    1.0
0.12   0.12  0.04   0.2    0.12
2          0.0    0.0        1.0          1.0        1.0           1.0
0.0          0.0            0.0          0.0           0.0           0.0
1.0          0.0            0.0            0.0          0.0            1.0
0.0            0.0              1.0          0.0            1.0        0.0
0.0            0.0              1.0          1.0          1.0            0.0
0.030992  0.000000    0.1875    0.2    0.2           0.2        0.10    0.8
0.12   0.08  0.08   0.6    0.12
3          0.0    0.0        1.0          0.0        1.0           0.0
1.0          0.0            0.0          0.0           0.0           0.0
0.0            1.0            0.0            0.0          1.0            0.0
0.0            0.0              1.0          0.0            0.0        1.0
0.0            1.0              1.0          1.0          1.0            1.0
0.030992  0.000000    0.0000    0.8    0.4           0.2        0.15    0.6
0.08   0.08  0.04   0.2    0.20
4          0.0    0.0        1.0          0.0        1.0           0.0
0.0          1.0            0.0          0.0           0.0           0.0
1.0          0.0            0.0            0.0          1.0            0.0
0.0            1.0              0.0          0.0            0.0        1.0
0.0            0.0              1.0          1.0          0.0            0.0
0.033058  0.000000    0.0000    0.6    0.6           0.2        0.10    0.8
```

```
0.12    0.08  0.04   0.4     0.20
..            ...   ...              ...               ...               ...                   ...
...              ...                    ...               ...               ...                  ...
...                ...               ...                    ...             ...                   ...
...                 ...                     ...                ...               ...                 ...
...                 ...               ...               ...                  ...                  ...
...            ...       ...    ...    ...               ...                ...       ...    ...
...     ...    ...       ...
644        1.0     0.0         0.0          0.0         1.0          0.0
0.0          0.0                 1.0          0.0           0.0          0.0
1.0            0.0           0.0             1.0           0.0            0.0
0.0             0.0             1.0            0.0            0.0          0.0
0.0            1.0           0.0           1.0           1.0           0.0
0.039256 0.333333    0.1250   0.4   0.6           0.2        0.15     1.0
0.16   0.08  0.04   0.4    0.20
645        1.0     0.0         1.0          1.0         1.0          0.0
0.0          0.0             0.0            1.0            0.0          0.0
0.0            1.0           0.0             1.0           0.0            0.0
0.0             0.0             1.0            0.0            0.0         1.0
0.0             0.0           1.0            1.0           1.0           0.0
0.037190 0.000000    0.1250   0.6   0.2           0.2        0.10     0.8
0.12   0.16  0.04   0.2     0.04
646        1.0     0.0         1.0          0.0         1.0          0.0
0.0           1.0           0.0            0.0            0.0          0.0
1.0            0.0           0.0             1.0           0.0            0.0
0.0             0.0             1.0            0.0            0.0          0.0
0.0            1.0          1.0            1.0            0.0            0.0
0.037190 0.000000    0.1875   0.2   0.2           0.4        0.10     0.2
0.04   0.04  0.04   0.2     0.20
647        1.0     1.0         1.0          1.0         1.0          0.0
0.0           0.0             1.0            0.0            0.0          0.0
0.0            1.0           0.0             1.0           0.0            0.0
0.0             0.0             1.0            0.0            0.0          0.0
0.0             0.0           0.0            1.0           1.0           0.0
0.035124 0.000000    0.1875   0.6   0.2           0.4        0.05     0.4
0.16   0.20  0.12   0.8     0.08
648        1.0     1.0         0.0          1.0         1.0          0.0
0.0          0.0             1.0            0.0            0.0          0.0
1.0            0.0           0.0             1.0           0.0            0.0
0.0             0.0             1.0            0.0            0.0          0.0
0.0             0.0           0.0            1.0           1.0           0.0
0.037190 0.000000    0.1250   0.6   0.4           0.6        0.05     0.8
0.16   0.04  0.12   0.8     0.20

[649 rows x 43 columns]


Dobór zmiennych korzystając z gotowych funkcji
```

```
[10]: def feature_names(selector):
          return np.array(pf.get_feature_names(X_train.columns))[selector.
      ↪get_support()]
```

Podział na zbiór testowy i treningowy

```
[11]: X_train, X_test, y_train, y_test= train_test_split(X_all, y, test_size = 0.1,␣
      ↪random_state=42)
```

Wyznaczenie baseline'u

```
[12]: mn = np.mean(y_train)
      baseline = [mn for i in range(len(y_test))]

      np.sqrt(mean_squared_error(y_test, baseline))
```

```
[12]: 3.070836358822292
```

Polynomial features

```
[13]: pf = PolynomialFeatures(degree=2)
      X_features = pf.fit_transform(X_train)
      X_test = pf.fit_transform(X_test)
```

chi selector

```
[14]: chi2_selector = SelectKBest(chi2, k=12)
      chi2_selector.fit_transform(X_features, y_train)
      feature_names(chi2_selector)
```

```
[14]: array(['failures', 'school_MS guardian_mother', 'address_U failures',
             'Pstatus_T failures', 'Mjob_at_home Fjob_health',
             'Mjob_services failures', 'Fjob_health reason_home',
             'Fjob_services failures', 'reason_other failures',
             'guardian_mother failures', 'internet_yes failures',
             'romantic_yes failures'], dtype='<U33')
```

mi selector

```
[15]: mi_selector = SelectKBest(mutual_info_classif, k=12)
      mi_selector.fit(X_features, y_train)
      feature_names(mi_selector)
```

```
[15]: array(['failures', 'school_MS famrel', 'guardian_mother failures',
             'higher_yes Medu', 'age failures', 'failures absences',
             'failures Medu', 'failures traveltime', 'failures famrel',
             'failures Dalc', 'failures Walc', 'goout Dalc'], dtype='<U33')
```

rfe selector

```
[16]:  # estimator = LogisticRegression(max_iter=2000)
       # rfe_selector = RFE(estimator, n_features_to_select=10, step=1)
       # rfe_selector = rfe_selector.fit(X_features, y_train)
       # feature_names(rfe_selector)
```

L1-based feature selection

```
[17]:  def selection(num_features):
           r = 0.1
           l = 0.000000000001
           c = (l+r)/2
           while True:
               model_selector = SelectFromModel(
                   LogisticRegression(penalty="l1", C=c, solver="liblinear",␣
       ↪random_state=42)
               )
               model_selector.fit_transform(X_features, y_train)
               feat = len(feature_names(model_selector))
               if feat > num_features:
                   r = c
                   c = (r+l)/2
               elif feat < num_features:
                   l = c
                   c = (l+r)/2
               else:
                   break
               print("Currently on ", len(feature_names(model_selector)), " features.")
           print("Selected ", len(feature_names(model_selector)), " features.")
           return(model_selector)


       mod = selection(12)
       feature_names(mod)
```

```
Selected  12  features.
```

```
[17]: array(['1', 'address_U', 'Pstatus_T', 'guardian_mother', 'nursery_yes',
             'higher_yes', 'address_U^2', 'address_U higher_yes', 'Pstatus_T^2',
             'nursery_yes^2', 'higher_yes^2', 'higher_yes internet_yes'],
            dtype='<U33')
```

Wytrenowanie 4 wybranych modeli na automatycznie przygotowanych danych

```
[18]:  X_chi2 = X_features[:, chi2_selector.get_support()]
       X_mi = X_features[:, mi_selector.get_support()]
       # X_rfe = X_features[:, rfe_selector.get_support()]
       X_msel = X_features[:, mod.get_support()]


       X_chi2_t = X_test[:, chi2_selector.get_support()]
```

```
X_mi_t = X_test[:, mi_selector.get_support()]
# X_rfe_t = X_test[:, rfe_selector.get_support()]
X_msel_t = X_test[:, mod.get_support()]


X_list = [X_chi2, X_mi, X_rfe, X_msel]
X_list_t = [X_chi2_t, X_mi_t, X_rfe_t, X_msel_t]

res = []

for i in range(4):
    lr = LinearRegression()
    sv = SVR(C=1.5)
    rf = RandomForestRegressor(n_estimators=20, max_features=0.5,␣
 ↪min_samples_split=3, n_jobs=-1, random_state=0)
    gb = GradientBoostingRegressor(learning_rate=0.045, n_estimators=100,␣
 ↪criterion='mse', random_state=0)

    lr.fit(X_list[i], y_train)
    lr_pred = lr.predict(X_list_t[i])
    lr_err = np.sqrt(mean_squared_error(y_test, lr_pred))

    sv.fit(X_list[i], y_train)
    sv_pred = sv.predict(X_list_t[i])
    sv_err = np.sqrt(mean_squared_error(y_test, sv_pred))

    rf.fit(X_list[i], y_train)
    rf_pred = rf.predict(X_list_t[i])
    rf_err = np.sqrt(mean_squared_error(y_test, rf_pred))

    gb.fit(X_list[i], y_train)
    gb_pred = gb.predict(X_list_t[i])
    gb_err = np.sqrt(mean_squared_error(y_test, gb_pred))

    temp_res = [lr_err, sv_err, rf_err, gb_err]
    res.append(temp_res)
```

0.2.2 Trenowanie modeli na wybranych i przetworzonych przez nas zmiennych

```
[19]: df = pd.DataFrame(data['data'])

      ### Stworzenie nowych kolumn korzystając z dostępnych danych
      df['Pedu'] = df['Fedu'] + df['Medu']
      df["genrel"] = df["sex"]+df["romantic"]
      df["Alc"] = (df["Dalc"]+df["Walc"]) / 10
      df[["absenc"]] = np.where(df['absences']<8, 0, 1)
```

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

### Przeskalowanie kolumn
df[["Pedu"]]  = df[["Pedu"]] /df['Pedu'].max()
df[["studytime"]]  = df[["studytime"]] /df['studytime'].max()
df[["age"]] = df[["age"]] /df['age'].max()
df[["health"]] = df[["health"]]/df['health'].max()
df[["goout"]] = df[["goout"]]/df['goout'].max()
df[["freetime"]]  = df[["freetime"]] /df['freetime'].max()
df[["Dalc"]] = df[["Dalc"]]/df['Dalc'].max()
df[["absences"]] = df[["absences"]]/df['absences'].max()
```

Dobór kategorycznych i numerycznych zmiennych

```
[20]: cat_features = ["Mjob", "higher", "genrel", "address", "reason", "school",
 ↪'internet']
num_features = ["Pedu", "studytime", "goout", "age", "fail"]
```

```
[21]: features = num_features + cat_features
X = df.drop(["G3"], axis=1)[features]
y = df["G3"]
```

```
[22]: # Preprocesssing numerycznych zmiennych:
num_transformer = SimpleImputer(strategy="constant")

# Preprocesssing kategorycznych zmiennych:
cat_transformer = Pipeline(steps=[
                                 ("imputer",
 ↪SimpleImputer(strategy="constant", fill_value="Unknown")),
                                 ("onehot",
 ↪OneHotEncoder(handle_unknown='ignore'))
                                 ]
                           )

preprocessor = ColumnTransformer(transformers=[("num", num_transformer,
 ↪num_features),
                                              ("cat", cat_transformer,
 ↪cat_features)],
                                remainder = 'passthrough')
```

Wytrenowanie wybrancyh modeli z dobranymi ręcznie hiperparametrami

```
[23]: gb = GradientBoostingRegressor(learning_rate=0.045, n_estimators=100,
 ↪criterion='mse', random_state=0)
rf = RandomForestRegressor(n_estimators=20, max_features=0.5,
 ↪min_samples_split=3, n_jobs=-1, random_state=0)
```

```python
lr = LinearRegression()
svr = SVR(C=1.5)

model_pipe_gb = Pipeline(steps=[('preprocessor', preprocessor),
                                ('model', gb)])
model_pipe_rf = Pipeline(steps=[('preprocessor', preprocessor),
                                ('model', rf)])
model_pipe_lr = Pipeline(steps=[('preprocessor', preprocessor),
                                ('model', lr)])
model_pipe_svr = Pipeline(steps=[('preprocessor', preprocessor),
                                 ('model', svr)])

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.1, random_state=42)

model_pipe_gb.fit(X_train, y_train)
model_pipe_rf.fit(X_train, y_train)
model_pipe_lr.fit(X_train, y_train)
model_pipe_svr.fit(X_train, y_train)
```

```
[23]: Pipeline(steps=[('preprocessor',
                       ColumnTransformer(remainder='passthrough',
                                         transformers=[('num',
      SimpleImputer(strategy='constant'),
                                                        ['Pedu', 'studytime', 'goout',
                                                         'age', 'fail']),
                                                       ('cat',
                                                        Pipeline(steps=[('imputer',
      SimpleImputer(fill_value='Unknown',
       strategy='constant')),
                                                                        ('onehot',
      OneHotEncoder(handle_unknown='ignore'))]),
                                                        ['Mjob', 'higher', 'genrel',
                                                         'address', 'reason',
                                                         'school', 'internet'])])),
                      ('model', SVR(C=1.5))])
```

Sprawdzenie wyników i porównanie modeli

```python
[24]: y_predict_gb = model_pipe_gb.predict(X_test)
      y_predict_rf = model_pipe_rf.predict(X_test)
      y_predict_lr = model_pipe_lr.predict(X_test)
      y_predict_svr = model_pipe_svr.predict(X_test)

      res.append([
          np.sqrt(mean_squared_error(y_test, y_predict_lr)),
          np.sqrt(mean_squared_error(y_test, y_predict_svr)),
```

```
        np.sqrt(mean_squared_error(y_test, y_predict_rf)),
        np.sqrt(mean_squared_error(y_test, y_predict_gb))
])


#      )], columns=["Linear Regression", "SVR", "Random Forest","Gradient
 ↪Boosting"])


results = pd.DataFrame(res, columns=["Linear Regression", "SVR", "Random
 ↪Forest","Gradient Boosting"], index=["SelectKBest (chi2)","SelectKBest
 ↪(mutual information)", "RFE", "L1 Based Model Selection", "Hand-prepared
 ↪features"])
```

[25]: `results`

[25]:
	Linear Regression	SVR	Random Forest
Gradient Boosting			
SelectKBest (chi2)	3.139358	2.884703	2.926594
2.875707			
SelectKBest (mutual information)	2.796916	2.831569	2.786297
2.774692			
RFE	2.949454	2.758008	2.716849
2.791437			
L1 Based Model Selection	2.801049	2.813967	2.905404
2.834317			
Hand-prepared features	2.639725	2.680089	2.770634
2.591747			

Analiza najlepszego modelu - Gradient Boosting

[26]: `explainer = dx.Explainer(model_pipe_gb,X,y)`

```
Preparation of a new explainer is initiated

  -> data                : 649 rows 12 cols
  -> target variable     : Parameter 'y' was a pandas.Series. Converted to a
numpy.ndarray.
  -> target variable     : 649 values
  -> model_class         : sklearn.ensemble._gb.GradientBoostingRegressor
(default)
  -> label               : Not specified, model's class short name will be used.
(default)
  -> predict function    : <function yhat_default at 0x000001EB96053CA0> will be
used (default)
  -> predict function    : Accepts only pandas.DataFrame, numpy.ndarray causes
problems.
  -> predicted values    : min = 3.25, mean = 11.9, max = 16.1
  -> model type          : regression will be used (default)
  -> residual function   : difference between y and yhat (default)
```

```
-> residuals          : min = -10.1, mean = 0.0444, max = 6.45
-> model_info         : package sklearn
```

A new explainer has been created!

Zbadanie wpływu zmiennych na predykcję dla dwóch losowych rekordów - przeciętnego i słabego wyniku

```
[27]: y[10]
```

```
[27]: 14
```

```
[28]: explainer.predict_parts(X.loc[[10],:]).plot()
```

```
[29]: y[432]
```

```
[29]: 7
```

```
[30]: explainer.predict_parts(X.loc[[432],:]).plot()
```