Congressional_Voting_Tuning

April 20, 2021

1 Congressional voting

1.1 *Hubert Ruczyński, Bartosz Sawicki*

2 Przygotowanie danych i środowiska

2.1 Importy

[1]:
```
import numpy as np
import pandas as pd
import requests
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

from matplotlib import pyplot as plt
plt.style.use('ggplot')

from sklearn import metrics
from sklearn import tree
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import cross_validate
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import plot_roc_curve
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix

from xgboost import XGBClassifier
```

2.2 Wczytanie danych

[2]:
```python
url = 'https://api.apispreadsheets.com/api/dataset/congressional-voting/'

r = requests.get(url)
data = r.json()
df = pd.DataFrame.from_dict(data['data'], orient='columns')
df.sample(5)
```

[2]:
```
    handicapped_infants water_project_cost_sharing  \
104                   ?                          ?
16                    y                          n
314                   n                          y
290                   y                          n
18                    n                          y

    adoption_of_the_budget_resolution physician_fee_freeze el_salvador_aid  \
104                                 ?                    ?               n
16                                  y                    n               n
314                                 n                    y               y
290                                 y                    n               ?
18                                  n                    y               y

    religious_groups_in_schools anti_satellite_test_ban  \
104                           y                       y
16                            y                       n
314                           y                       y
290                           y                       ?
18                            y                       n

    aid_to_nicaraguan_contras mx_missile immigration  \
104                         y          y           y
16                          y          ?           y
314                         y          n           n
290                         y          y           y
18                          n          n           n

    synfuels_corporation_cutback education_spending superfund_right_to_sue  \
104                            ?                  n                      y
16                             y                  y                      ?
314                            y                  y                      y
290                            n                  n                      y
18                             n                  ?                      y

    crime duty_free_exports export_administration_act_south_africa  \
104     y                  n                                       ?
16      n                  n                                       y
```

```
314     y                y                                       y
290     y                n                                       y
18      y                n                                       n

    political_party
104        democrat
16         democrat
314      republican
290        democrat
18       republican
```

[3]:
```
X = df.drop('political_party', axis=1)
y = df.iloc[:,16]
X.info()
y
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 435 entries, 0 to 434
Data columns (total 16 columns):
 #   Column                                 Non-Null Count  Dtype
---  ------                                 --------------  -----
 0   handicapped_infants                    435 non-null    object
 1   water_project_cost_sharing             435 non-null    object
 2   adoption_of_the_budget_resolution      435 non-null    object
 3   physician_fee_freeze                   435 non-null    object
 4   el_salvador_aid                        435 non-null    object
 5   religious_groups_in_schools            435 non-null    object
 6   anti_satellite_test_ban                435 non-null    object
 7   aid_to_nicaraguan_contras              435 non-null    object
 8   mx_missile                             435 non-null    object
 9   immigration                            435 non-null    object
 10  synfuels_corporation_cutback           435 non-null    object
 11  education_spending                     435 non-null    object
 12  superfund_right_to_sue                 435 non-null    object
 13  crime                                  435 non-null    object
 14  duty_free_exports                      435 non-null    object
 15  export_administration_act_south_africa 435 non-null    object
dtypes: object(16)
memory usage: 54.5+ KB
```

[3]:
```
0      republican
1      republican
2        democrat
3        democrat
4        democrat
          …
430    republican
```

```
431      democrat
432    republican
433    republican
434    republican
Name: political_party, Length: 435, dtype: object
```

Rozważymy 2 zestawy zmiennych objaśniających. Wszystkie dostępne dane oraz dane z usuniętymi kolumnami, które w czasie EDA uznaliśmy za nieróżnicujące.

[4]:
```
df_dropped = df.
↪drop(['water_project_cost_sharing','immigration','export_administration_act_south_africa'],
↪axis=1)
X_dropped = df_dropped.iloc[:,:12]
y_dropped = df_dropped.iloc[:,13]
```

2.3 Podział na zbiór testowy i walidacyjny

[5]:
```
x_train, x_test, y_train, y_test = train_test_split(df_dropped.iloc[:,:12],␣
↪df_dropped.iloc[:,13], random_state=43)
x_train_o, x_test_o, y_train_o, y_test_o = train_test_split(X, y,␣
↪random_state=43)
```

2.4 One Hot Encoding

Zgodnie z wcześniejszymi wynikami naszej pracy, do uzyskania dobrych wyników skorzytamy z One Hot Encodingu.

[6]:
```
!pip install category_encoders
```

```
Requirement already satisfied: category_encoders in
/home/sawcio/Studia/4sem/Wstęp_do_U_M/venv/lib/python3.8/site-packages (2.2.2)
Requirement already satisfied: patsy>=0.5.1 in
/home/sawcio/Studia/4sem/Wstęp_do_U_M/venv/lib/python3.8/site-packages (from
category_encoders) (0.5.1)
Requirement already satisfied: scikit-learn>=0.20.0 in
/home/sawcio/Studia/4sem/Wstęp_do_U_M/venv/lib/python3.8/site-packages (from
category_encoders) (0.24.1)
Requirement already satisfied: statsmodels>=0.9.0 in
/home/sawcio/Studia/4sem/Wstęp_do_U_M/venv/lib/python3.8/site-packages (from
category_encoders) (0.12.2)
Requirement already satisfied: numpy>=1.14.0 in
/home/sawcio/Studia/4sem/Wstęp_do_U_M/venv/lib/python3.8/site-packages (from
category_encoders) (1.20.1)
Requirement already satisfied: scipy>=1.0.0 in
/home/sawcio/Studia/4sem/Wstęp_do_U_M/venv/lib/python3.8/site-packages (from
category_encoders) (1.6.1)
Requirement already satisfied: pandas>=0.21.1 in
```

```
/home/sawcio/Studia/4sem/Wstęp_do_U_M/venv/lib/python3.8/site-packages (from
category_encoders) (1.2.2)
Requirement already satisfied: pytz>=2017.3 in
/home/sawcio/Studia/4sem/Wstęp_do_U_M/venv/lib/python3.8/site-packages (from
pandas>=0.21.1->category_encoders) (2021.1)
Requirement already satisfied: python-dateutil>=2.7.3 in
/home/sawcio/Studia/4sem/Wstęp_do_U_M/venv/lib/python3.8/site-packages (from
pandas>=0.21.1->category_encoders) (2.8.1)
Requirement already satisfied: six in
/home/sawcio/Studia/4sem/Wstęp_do_U_M/venv/lib/python3.8/site-packages (from
patsy>=0.5.1->category_encoders) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/home/sawcio/Studia/4sem/Wstęp_do_U_M/venv/lib/python3.8/site-packages (from
scikit-learn>=0.20.0->category_encoders) (2.1.0)
Requirement already satisfied: joblib>=0.11 in
/home/sawcio/Studia/4sem/Wstęp_do_U_M/venv/lib/python3.8/site-packages (from
scikit-learn>=0.20.0->category_encoders) (1.0.1)
```

[7]:
```python
import category_encoders as ce

one_hot_encoder = ce.OneHotEncoder()

one_hot = one_hot_encoder.fit_transform(X,y)

one_hot_encoder_dropped = ce.OneHotEncoder()
one_hot_dropped = one_hot_encoder_dropped.fit_transform(X_dropped,y_dropped)
```

3 Gradient Boosting Clasifier

Pierwszym z trzech modeli które postanowiliśmy dostroić jest Gradient Boosting. Poniżej znajdują się jego wyniki przed strojeniem.

[8]:
```python
clf = GradientBoostingClassifier(n_estimators=1000, learning_rate=.05,
    max_depth=3, random_state=997)

pipe_one_hot = Pipeline(
[
    ('transformer_one_hot', one_hot_encoder),
    ('classifier', clf)
])

pipe_one_hot_dropped = Pipeline(
[
    ('transformer_one_hot', one_hot_encoder_dropped),
    ('classifier', clf)
])
```

[9]:
```
print(np.mean(cross_validate(pipe_one_hot, X, y, cv=11, scoring='accuracy').
 ↪get('test_score')))
print(np.mean(cross_validate(pipe_one_hot_dropped, X_dropped, y_dropped, cv=11,␣
 ↪scoring='accuracy').get('test_score')))
```

```
0.9514568764568765
0.9491841491841492
```

3.1 Strojenie ręczne

W tym podrozdziale znajduje się najlepszy wynik otrzymany za pomocą ręcznego strojenia parametrów.

[10]:
```
clf = GradientBoostingClassifier(loss='deviance',n_estimators=1000,␣
 ↪learning_rate=.05, max_depth=3,criterion='friedman_mse', random_state=997)

pipe_one_hot = Pipeline(
[
    ('transformer_one_hot', one_hot_encoder),
    ('classifier', clf)
])

pipe_one_hot_dropped = Pipeline(
[
    ('transformer_one_hot', one_hot_encoder_dropped),
    ('classifier', clf)
])
```

[11]:
```
print(np.mean(cross_validate(pipe_one_hot, X, y, cv=11, scoring='accuracy').
 ↪get('test_score')))
print(np.mean(cross_validate(pipe_one_hot_dropped, X_dropped, y_dropped, cv=11,␣
 ↪scoring='accuracy').get('test_score')))
```

```
0.9514568764568765
0.9491841491841492
```

3.2 GridSearchCV

Użyjemy funkcji GridSearchCV, aby znaleźć najlepsze parametry. Będziemy przeszukiwać po parametrach `n_estimators` oraz `max_depth`. Ustawiony został `random_state` aby uzyskać reprodukowalność wyników.

[13]:
```
# Ustalamy siatke parametrów, z której wszystkie kombinacje parametrów będą␣
 ↪użyte\

# przy uzyciu w pipeline do parametru dodajemy przedrostek
# {nazwa klasyfikatora w pipeline}__ żeby było wiadomo czego parametry zmieniamy
parameters = {
```

```python
    'gbc_name__n_estimators': [50, 100, 150, 200, 400],
    'gbc_name__max_depth': [x for x in range(1,6)],
    'gbc_name__random_state': [997]
}
gbc = GradientBoostingClassifier()

pipe_clf = Pipeline([('ohe', one_hot_encoder),('gbc_name', gbc)])

clf = GridSearchCV(pipe_clf, param_grid=parameters, n_jobs=-1)
clf.fit(X,y)
```

[13]:
```
GridSearchCV(estimator=Pipeline(steps=[('ohe',
OneHotEncoder(cols=['handicapped_infants',
'water_project_cost_sharing',
'adoption_of_the_budget_resolution',
'physician_fee_freeze',
                                                            'el_salvador_aid',
'religious_groups_in_schools',
'anti_satellite_test_ban',
'aid_to_nicaraguan_contras',
                                                            'mx_missile',
                                                            'immigration',
'synfuels_corporation_cutback',
'education_spending',
'superfund_right_to_sue',
                                                            'crime',
                                                            'duty_free_exports',
'export_administration_act_south_africa'])),
                                       ('gbc_name',
                                        GradientBoostingClassifier())]),
             n_jobs=-1,
             param_grid={'gbc_name__max_depth': [1, 2, 3, 4, 5],
                         'gbc_name__n_estimators': [50, 100, 150, 200, 400],
                         'gbc_name__random_state': [997]})
```

[14]:
```python
clf.best_score_
```

[14]:
```
0.9678160919540231
```

[15]:
```python
clf.best_params_
```

[15]:
```
{'gbc_name__max_depth': 1,
 'gbc_name__n_estimators': 400,
 'gbc_name__random_state': 997}
```

Najlepszy znaleziony klasyfikator osiągnął dokładność 96,78% z parametrami: - max_depth': 1 - n_estimators': 400

4 XGBClassifier

Drugim z trzech modeli które postanowiliśmy dostroić jest XGB. Poniżej znajdują się jego wyniki przed strojeniem. Ponadto przedstawione zostały Confusion Matrixes, które obrazują ile danych zostało 'przestrzelonych'.

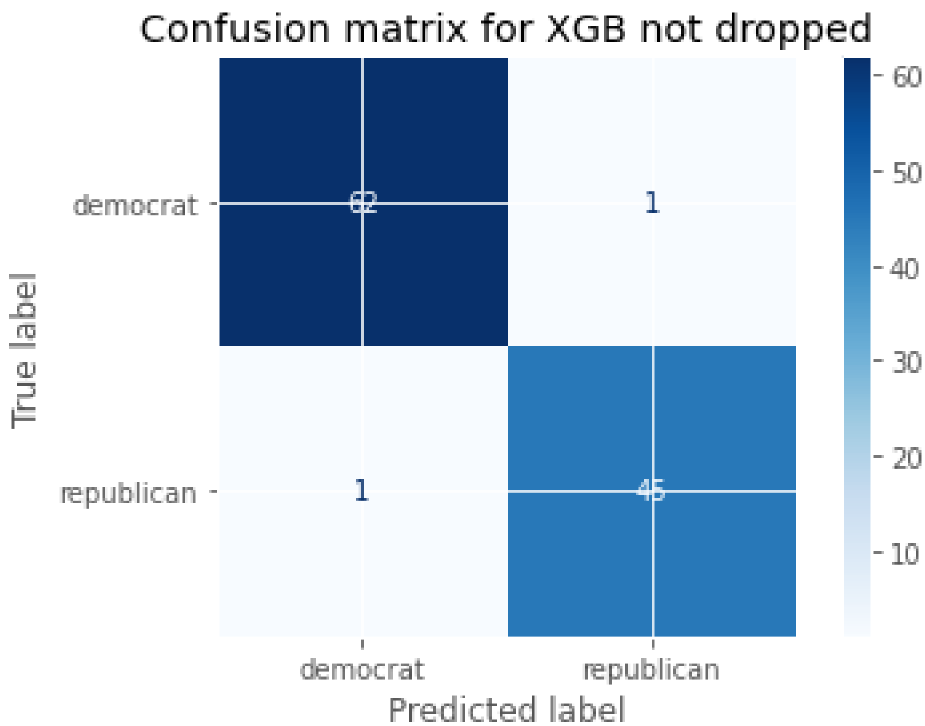
https://xgboost.readthedocs.io/en/latest/parameter.html#global-configuration

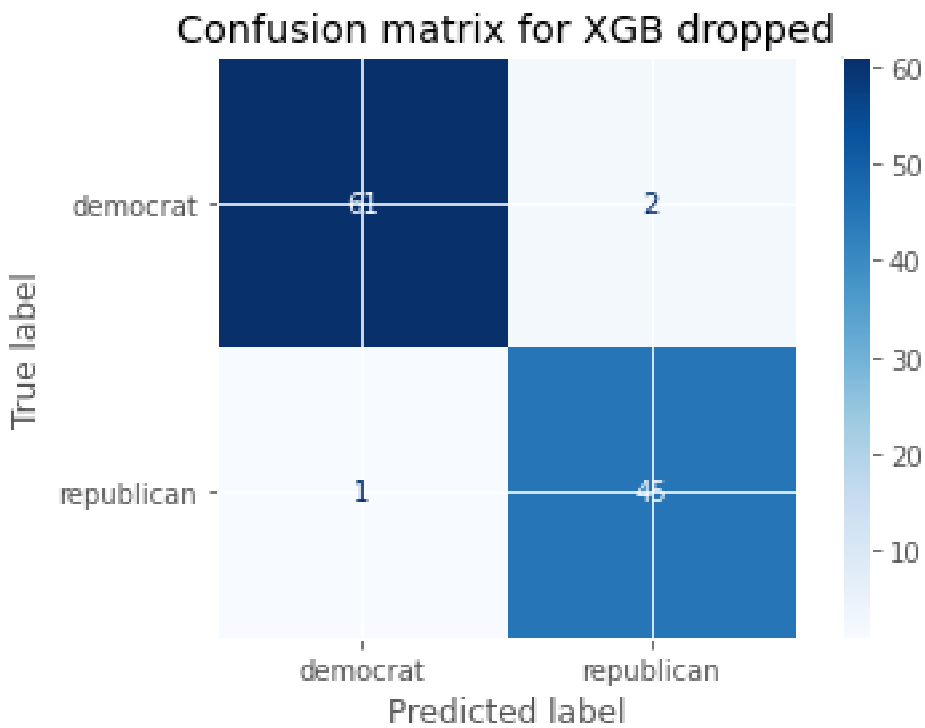
[25]:
```
model=XGBClassifier(random_state=1,
                    learning_rate=0.01, # Szybkość "uczenia" się
                    booster='gbtree', # Jaki model wykorzystujemy (drzewo -␣
 ↪gbtree, liniowe - gblinear)
                    nround = 1000, # Ilość itereacji boosingowych
                    max_depth=3, # Maksymalna głębokość drzewa
                    verbosity = 0
                    )
XGB_one_hot = Pipeline(
[
    ('transformer_one_hot', one_hot_encoder),
    ('classifier', model)
])

XGB_one_hot_dropped = Pipeline(
[
    ('transformer_one_hot', one_hot_encoder_dropped),
    ('classifier', model)
])
XGB_one_hot.fit(x_train_o,y_train_o)
prediction_test=XGB_one_hot.predict(x_test_o)
print(metrics.classification_report(y_test_o, prediction_test))
disp=plot_confusion_matrix(XGB_one_hot, x_test_o, y_test_o,cmap=plt.cm.Blues)
disp.ax_.set_title('Confusion matrix for XGB not dropped')
XGB_one_hot_dropped.fit(x_train,y_train)
prediction_test=XGB_one_hot_dropped.predict(x_test)
print(metrics.classification_report(y_test, prediction_test))
disp=plot_confusion_matrix(XGB_one_hot_dropped, x_test, y_test,cmap=plt.cm.
 ↪Blues)
disp.ax_.set_title('Confusion matrix for XGB dropped')
```

```
              precision    recall  f1-score   support

    democrat       0.98      0.98      0.98        63
  republican       0.98      0.98      0.98        46

    accuracy                           0.98       109
   macro avg       0.98      0.98      0.98       109
weighted avg       0.98      0.98      0.98       109
```

```
              precision    recall  f1-score   support

    democrat       0.98      0.97      0.98        63
  republican       0.96      0.98      0.97        46

    accuracy                           0.97       109
   macro avg       0.97      0.97      0.97       109
weighted avg       0.97      0.97      0.97       109
```

[25]: Text(0.5, 1.0, 'Confusion matrix for XGB dropped')






```
[26]: print(np.mean(cross_validate(XGB_one_hot, X, y, cv=11, scoring='accuracy').
       ↪get('test_score')))
      print(np.mean(cross_validate(XGB_one_hot_dropped, X_dropped, y_dropped, cv=11,␣
       ↪scoring='accuracy').get('test_score')))
```

```
0.9539044289044291
0.9561188811188811
```

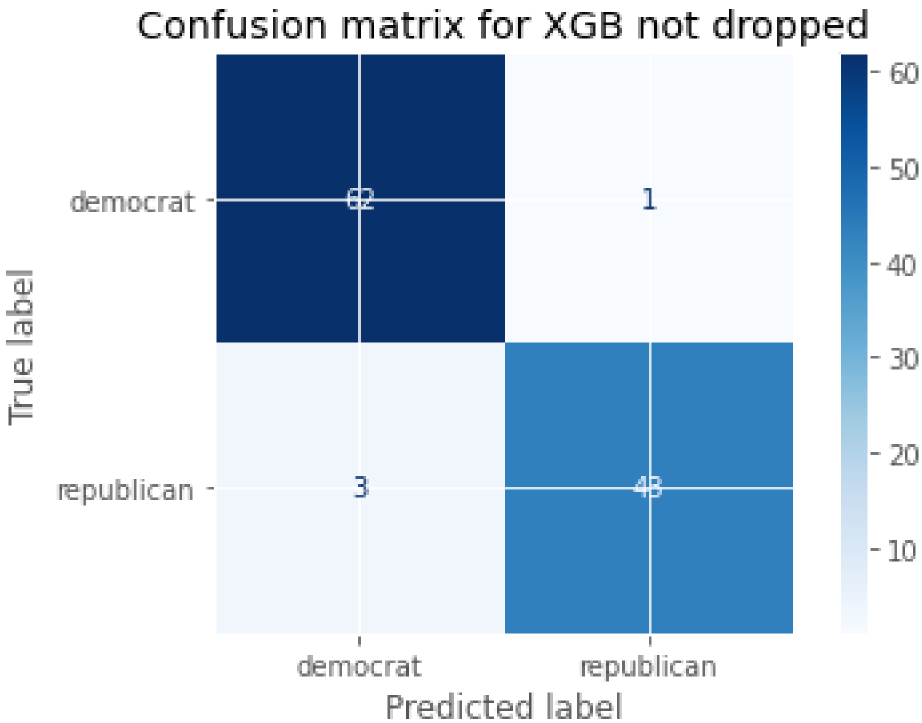
4.1 Strojenie ręczne

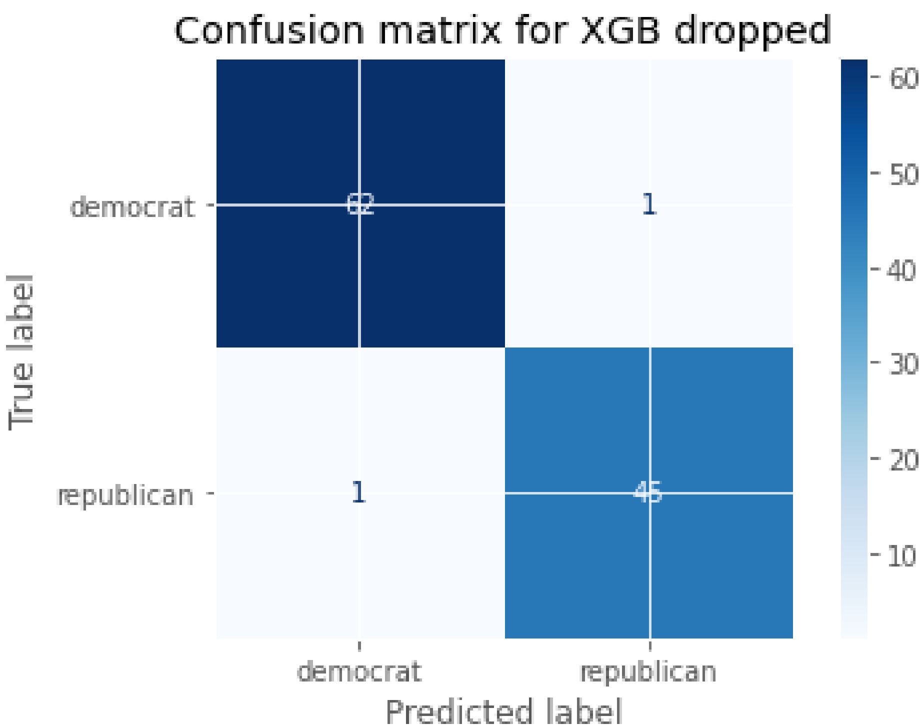
```
[27]: model=XGBClassifier(random_state=1,
                          learning_rate=0.09, # Szybkość "uczenia" się
                          booster='gbtree', # Jaki model wykorzystujemy (drzewo -␣
       ↪gbtree, liniowe - gblinear, 'dart' - drzewo)
                          nround = 1000, # Ilość itereacji boosingowych
                          max_depth=25, # Maksymalna głębokość drzewa
                          feature_selector ='shuffle'
                          )
      XGB_one_hot_2 = Pipeline(
      [
          ('transformer_one_hot', one_hot_encoder),
          ('classifier', model)
      ])
```

```
XGB_one_hot_2_dropped = Pipeline(
[
    ('transformer_one_hot', one_hot_encoder_dropped),
    ('classifier', model)
])

XGB_one_hot_2.fit(x_train_o,y_train_o)
prediction_test=XGB_one_hot_2.predict(x_test_o)
print(metrics.classification_report(y_test_o, prediction_test))
disp=plot_confusion_matrix(XGB_one_hot_2, x_test_o, y_test_o,cmap=plt.cm.Blues)
disp.ax_.set_title('Confusion matrix for XGB not dropped')

XGB_one_hot_2_dropped.fit(x_train,y_train)
prediction_test=XGB_one_hot_2_dropped.predict(x_test)
print(metrics.classification_report(y_test, prediction_test))
disp=plot_confusion_matrix(XGB_one_hot_2_dropped, x_test, y_test,cmap=plt.cm.
 ↪Blues)
disp.ax_.set_title('Confusion matrix for XGB dropped')
```

```
              precision    recall  f1-score   support

    democrat       0.95      0.98      0.97        63
  republican       0.98      0.93      0.96        46

    accuracy                           0.96       109
   macro avg       0.97      0.96      0.96       109
weighted avg       0.96      0.96      0.96       109

              precision    recall  f1-score   support

    democrat       0.98      0.98      0.98        63
  republican       0.98      0.98      0.98        46

    accuracy                           0.98       109
   macro avg       0.98      0.98      0.98       109
weighted avg       0.98      0.98      0.98       109
```

[27]: Text(0.5, 1.0, 'Confusion matrix for XGB dropped')


Confusion matrix for XGB not dropped
True label
democrat
republican
62
1
3
43
Predicted label
democrat
republican
60
50
40
30
20
10

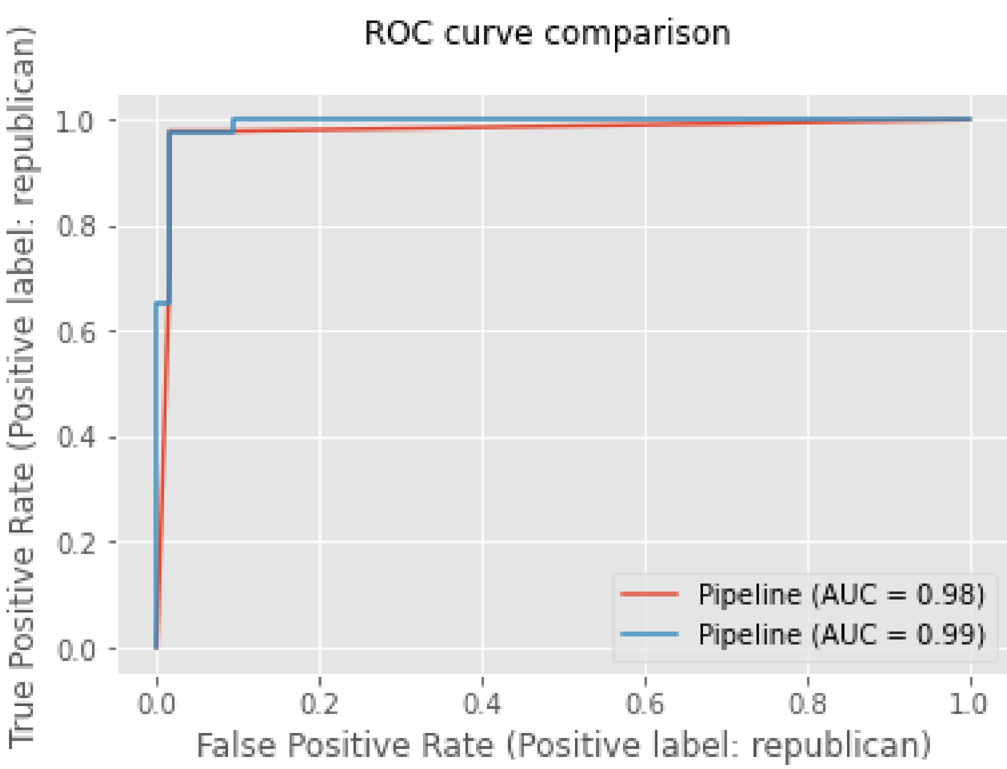

Confusion matrix for XGB dropped
True label
democrat
republican
62
1
1
45
Predicted label
democrat
republican
60
50
40
30
20
10

[28]:
```
print(np.mean(cross_validate(XGB_one_hot_2, X, y, cv=11, scoring='accuracy').
 ↪get('test_score')))
print(np.mean(cross_validate(XGB_one_hot_2_dropped, X_dropped, y_dropped,␣
 ↪cv=11, scoring='accuracy').get('test_score')))
```

```
0.954020979020979
0.9699883449883451
```

Najlepszy ręcznie znaleziony model ma dokładność 97%. Jest oparty na zbiorze danych z usuniętymi kolumnami.

[29]:
```
XGB_one_hot.fit(x_train_o,y_train_o)
XGB_one_hot_2.fit(x_train_o,y_train_o)

XGB_one_hot_disp = plot_roc_curve(XGB_one_hot, x_test_o, y_test_o)
XGB_one_hot_2_disp = plot_roc_curve(XGB_one_hot_2, x_test_o, y_test_o,␣
 ↪ax=XGB_one_hot_disp.ax_)
XGB_one_hot_2_disp.figure_.suptitle("ROC curve comparison")

plt.figure(figsize=(20,10))
plt.show()
```




```
<Figure size 1440x720 with 0 Axes>
```

4.2 GridSearchCV

Będziemy przeszukiwać parametry: - `n_estimators` - `max_depth` - `learning_rate`

oraz ustawimy `random_state` dla uzyskania reprodukowalności.

[22]:
```
parameters = {
    'xgb__n_estimators': [50, 150, 400, 600],
    'xgb__max_depth': [x for x in range(1,4)],
    'xgb__learning_rate': [.001, .01, .05],
    'xgb__random_state': [997]
}
xgb = XGBClassifier()

pipe_xgb = Pipeline([('ohe', one_hot_encoder),('xgb', xgb)])

xgb_clf_gs = GridSearchCV(pipe_xgb, param_grid=parameters, n_jobs=-1)
xgb_clf_gs.fit(X,y)
```

```
[02:04:45] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was changed
from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore
the old behavior.
```

[22]:
```
GridSearchCV(estimator=Pipeline(steps=[('ohe',
OneHotEncoder(cols=['handicapped_infants',
'water_project_cost_sharing',
'adoption_of_the_budget_resolution',
'physician_fee_freeze',
                                                              'el_salvador_aid',
'religious_groups_in_schools',
'anti_satellite_test_ban',
'aid_to_nicaraguan_contras',
                                                              'mx_missile',
                                                              'immigration',
'synfuels_corporation_cutback',
                                                              'education_spe…
                                                        monotone_constraints=None,
                                                        n_estimators=100,
                                                        n_jobs=None,
                                                        num_parallel_tree=None,
                                                        random_state=None,
                                                        reg_alpha=None,
                                                        reg_lambda=None,
                                                        scale_pos_weight=None,
                                                        subsample=None,
                                                        tree_method=None,
                                                        validate_parameters=None,
                                                        verbosity=None))]),
```

```
                n_jobs=-1,
                param_grid={'xgb__learning_rate': [0.001, 0.01, 0.05],
                            'xgb__max_depth': [1, 2, 3],
                            'xgb__n_estimators': [50, 150, 400, 600],
                            'xgb__random_state': [997]})
```

Najlepszy znaleziony klasyfikator miał skuteczność 96.55%. Jest to lepszy wynik niż wyjściowy klasyfikator, ale gorszy niż klasyfikator znaleziony ręcznie.

Parametry najlepszego modelu : - `learning_rate`: 0.05 - `max_depth`: 1 - `n_estimators`: 600

[23]:
```
xgb_clf_gs.best_params_
```

[23]:
```
{'xgb__learning_rate': 0.05,
 'xgb__max_depth': 1,
 'xgb__n_estimators': 600,
 'xgb__random_state': 997}
```

[24]:
```
xgb_clf_gs.best_score_
```

[24]:
```
0.9655172413793105
```

4.3 GridSearchCV dropped

Dokonaliśmy przeszukiwania parametrów także dla modelu opartego o mniejszą liczbę kolumn. Siatka parametrów w tym wyszukiwaniu była taka sama jak w przypadku pełnych danych.

[ ]:
```
parameters = {
    'xgb__n_estimators': [100 * x for x in range(1,11)],
    'xgb__max_depth': [x for x in range(1,7)],
    'xgb__learning_rate': [.001, .005, .01, .05, .1, .25],
    'xgb__random_state': [997]
}
xgb_dropped = XGBClassifier()

pipe_xgb_dropped = Pipeline([('ohe', one_hot_encoder_dropped),('xgb',␣
 ↪xgb_dropped)])

dropped_xgb_clf_gs = GridSearchCV(pipe_xgb_dropped, param_grid=parameters,␣
 ↪n_jobs=-1)
dropped_xgb_clf_gs.fit(X_dropped,y_dropped)
```

[ ]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('ohe',
OneHotEncoder(cols=['handicapped_infants',
'adoption_of_the_budget_resolution',
'physician_fee_freeze',
```

```
                                                                       'el_salvador_aid',
'religious_groups_in_schools',
'anti_satellite_test_ban',
'aid_to_nicaraguan_contras',
                                                                       'mx_missile',
'synfuels_corporation_cutback',
'education_spending',…
                                                                seed=None, silent=None,
                                                                subsample=1,
                                                                verbosity=1))],
                                  verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'xgb__learning_rate': [0.001, 0.005, 0.01, 0.05, 0.1,
                                                 0.25],
                         'xgb__max_depth': [1, 2, 3, 4, 5, 6],
                         'xgb__n_estimators': [100, 200, 300, 400, 500, 600,
                                               700, 800, 900, 1000],
                         'xgb__random_state': [997]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

Najlepszy model XGB oparty na wybranych kolumnach uzyskał skuteczność 97%. To najlepszy liczbowo wynik, natomiast przewaga nad modelem strojonym ręcznie jest niewielka i może wynikać z różnego podziału zbioru do CV, jak również z różnych parametrów `k` w CV.

Parametry najlepszego modelu XGB opartego na wybranych kolumnach: - `learning_rate`: 0.05 - `max_depth`: 4 - `n_estimators`: 200

[ ]:
```
dropped_xgb_clf_gs.best_score_
```

[ ]:
```
0.9701149425287356
```

[ ]:
```
dropped_xgb_clf_gs.best_params_
```

[ ]:
```
{'xgb__learning_rate': 0.05,
 'xgb__max_depth': 4,
 'xgb__n_estimators': 200,
 'xgb__random_state': 997}
```

5 AdaBoostClassifier

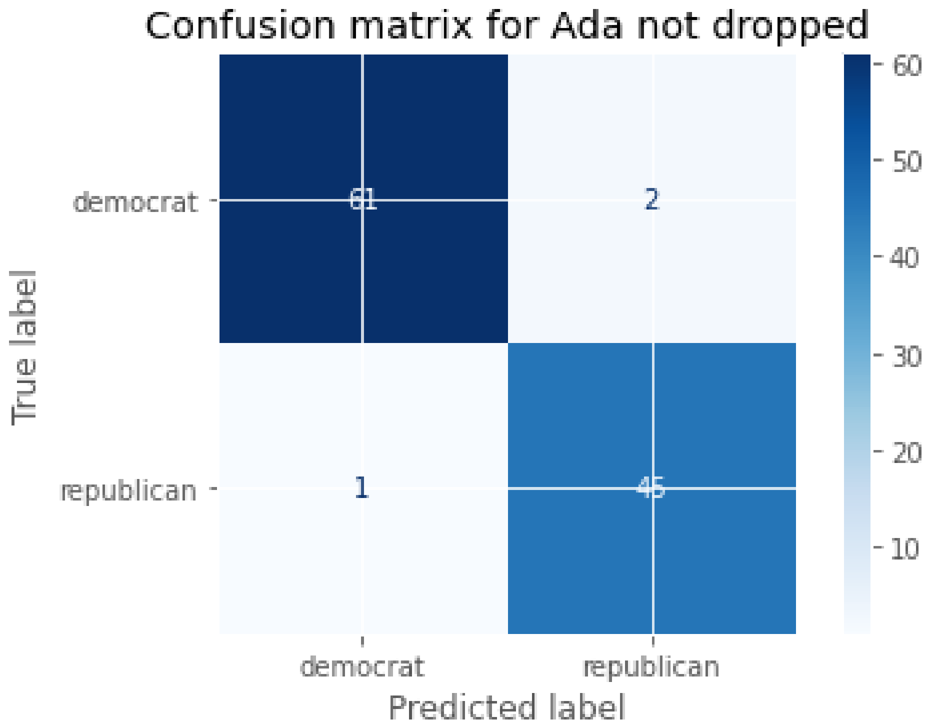
Trzecim z trzech modeli które postanowiliśmy dostroić jest Ada Boost. Poniżej znajdują się jego wyniki przed strojeniem. Ponadto przedstawione zostały Confusion Matrixes, które obrazują ile danych zostało 'przestrzelonych'.

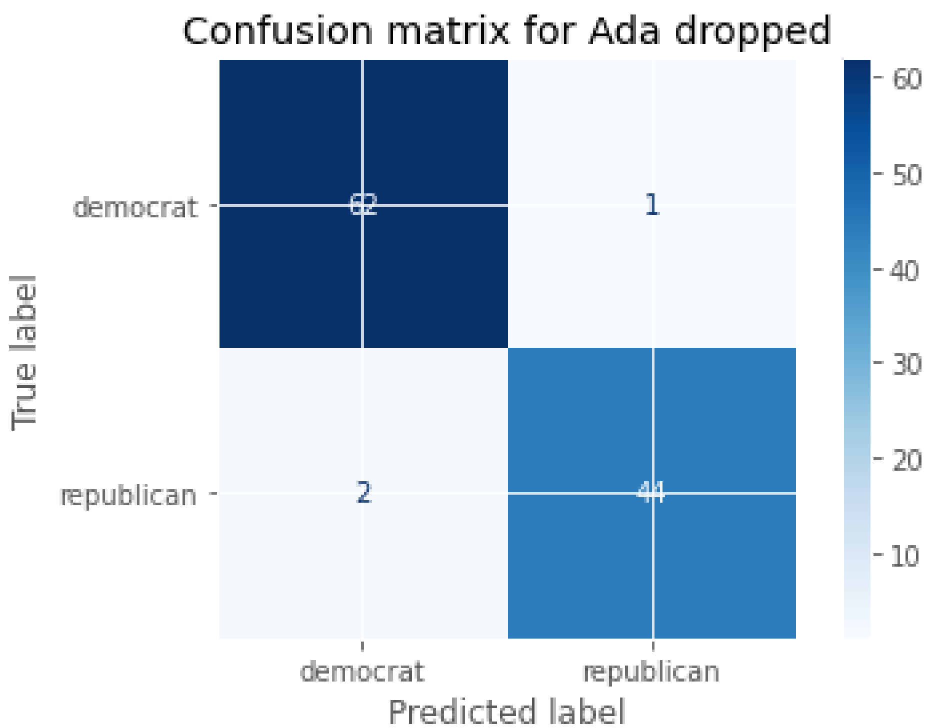
[ ]:
```
```

```
model = AdaBoostClassifier(random_state=1,n_estimators=50,learning_rate=0.
 ↪1,algorithm ='SAMME.R')
Ada_one_hot = Pipeline(
[
    ('transformer_one_hot', one_hot_encoder),
    ('classifier', model)
])

Ada_one_hot_dropped = Pipeline(
[
    ('transformer_one_hot', one_hot_encoder_dropped),
    ('classifier', model)
])
Ada_one_hot.fit(x_train_o,y_train_o)
prediction_test=Ada_one_hot.predict(x_test_o)
print(metrics.classification_report(y_test_o, prediction_test))
disp=plot_confusion_matrix(Ada_one_hot, x_test_o, y_test_o,cmap=plt.cm.Blues)
disp.ax_.set_title('Confusion matrix for Ada not dropped')

Ada_one_hot_dropped.fit(x_train,y_train)
prediction_test=Ada_one_hot_dropped.predict(x_test)
print(metrics.classification_report(y_test, prediction_test))
disp=plot_confusion_matrix(Ada_one_hot_dropped, x_test, y_test,cmap=plt.cm.
 ↪Blues)
disp.ax_.set_title('Confusion matrix for Ada dropped')
```

```
              precision    recall  f1-score   support

    democrat       0.98      0.97      0.98        63
  republican       0.96      0.98      0.97        46

    accuracy                           0.97       109
   macro avg       0.97      0.97      0.97       109
weighted avg       0.97      0.97      0.97       109

              precision    recall  f1-score   support

    democrat       0.97      0.98      0.98        63
  republican       0.98      0.96      0.97        46

    accuracy                           0.97       109
   macro avg       0.97      0.97      0.97       109
weighted avg       0.97      0.97      0.97       109
```

[ ]:
```
Text(0.5, 1.0, 'Confusion matrix for Ada dropped')
```


Confusion matrix for Ada not dropped
True label
democrat
republican
61
2
1
45
Predicted label
60
50
40
30
20
10


Confusion matrix for Ada dropped
True label
democrat
republican
62
1
2
44
Predicted label
60
50
40
30
20
10

```
[ ]: print(np.mean(cross_validate(Ada_one_hot, X, y, cv=7, scoring='accuracy').
      ↪get('test_score')))
     print(np.mean(cross_validate(Ada_one_hot_dropped, X_dropped, y_dropped, cv=7,␣
      ↪scoring='accuracy').get('test_score')))
```

```
0.951649476995099
0.9562577719259747
```

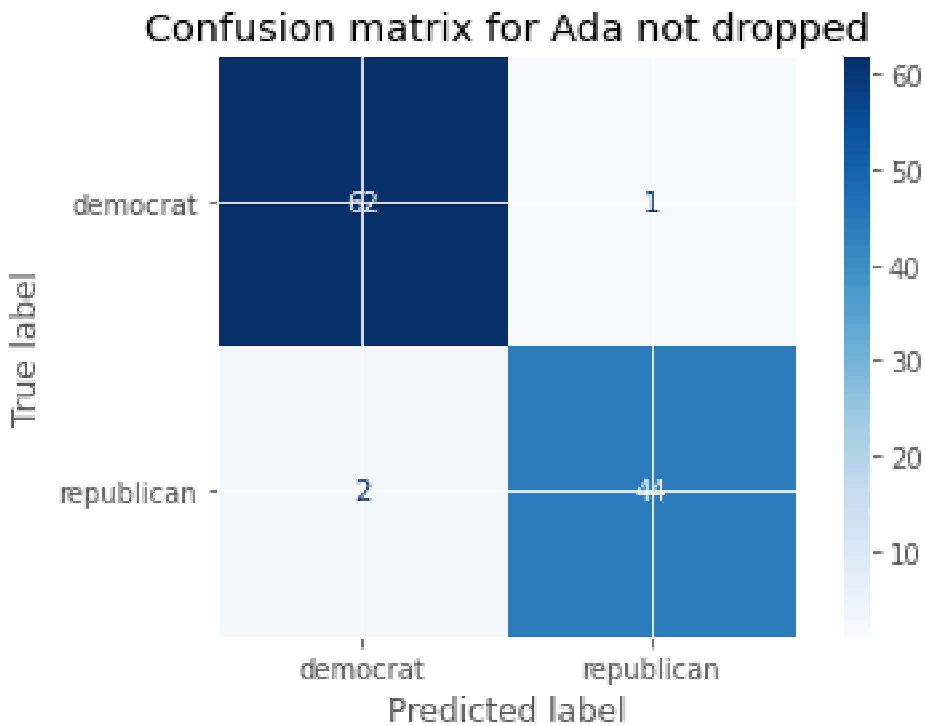
5.1 Strojenie ręczne

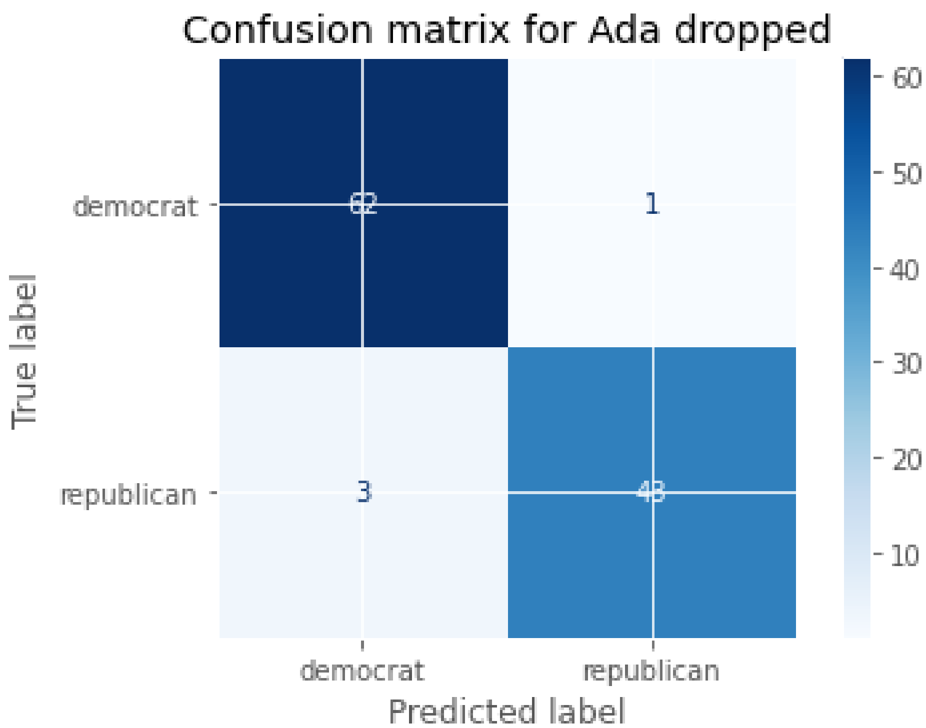
```
[ ]: model = AdaBoostClassifier(random_state=1,n_estimators=500,learning_rate=0.
      ↪1,algorithm ='SAMME.R')
     Ada_one_hot_2 = Pipeline(
     [
         ('transformer_one_hot', one_hot_encoder),
         ('classifier', model)
     ])

     Ada_one_hot_dropped_2 = Pipeline(
     [
         ('transformer_one_hot', one_hot_encoder_dropped),
         ('classifier', model)
     ])
     Ada_one_hot_2.fit(x_train_o,y_train_o)
     prediction_test=Ada_one_hot_2.predict(x_test_o)
     print(metrics.classification_report(y_test_o, prediction_test))
     disp=plot_confusion_matrix(Ada_one_hot_2, x_test_o, y_test_o,cmap=plt.cm.Blues)
     disp.ax_.set_title('Confusion matrix for Ada not dropped')

     Ada_one_hot_dropped_2.fit(x_train,y_train)
     prediction_test=Ada_one_hot_dropped_2.predict(x_test)
     print(metrics.classification_report(y_test, prediction_test))
     disp=plot_confusion_matrix(Ada_one_hot_dropped_2, x_test, y_test,cmap=plt.cm.
      ↪Blues)
     disp.ax_.set_title('Confusion matrix for Ada dropped')
```

```
              precision    recall  f1-score   support

    democrat       0.97      0.98      0.98        63
  republican       0.98      0.96      0.97        46

    accuracy                           0.97       109
   macro avg       0.97      0.97      0.97       109
weighted avg       0.97      0.97      0.97       109

              precision    recall  f1-score   support
```

```
  democrat       0.95      0.98      0.97        63
republican       0.98      0.93      0.96        46

  accuracy                           0.96       109
 macro avg       0.97      0.96      0.96       109
weighted avg     0.96      0.96      0.96       109
```

[ ]: Text(0.5, 1.0, 'Confusion matrix for Ada dropped')

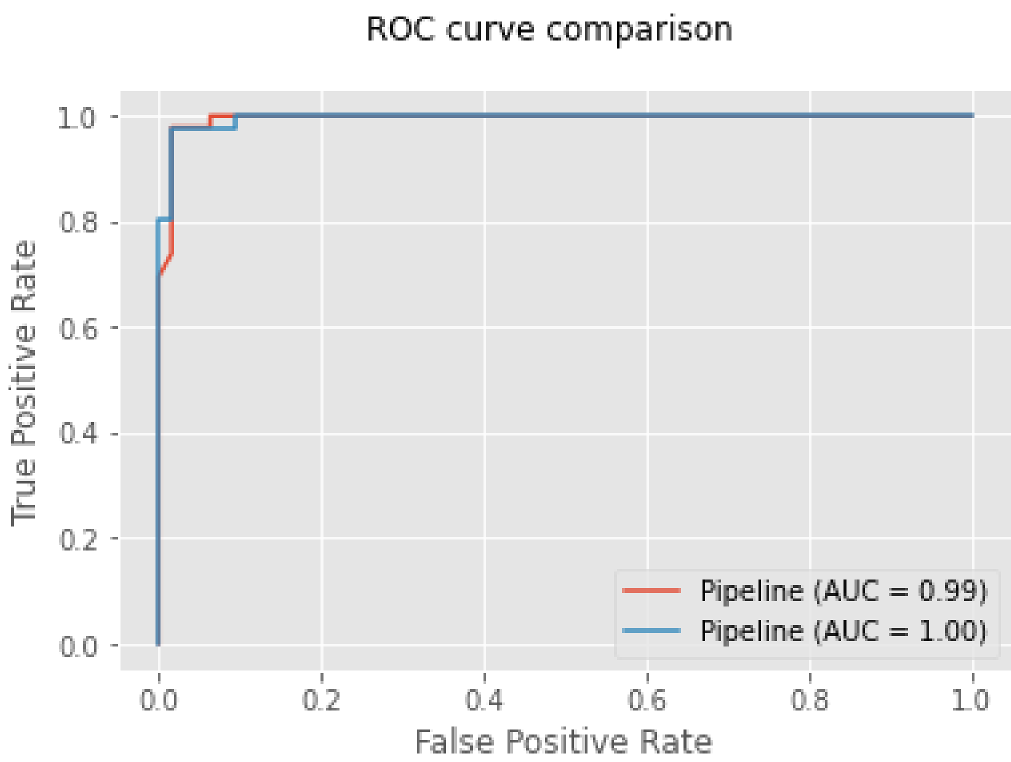




[ ]:
```
print(np.mean(cross_validate(Ada_one_hot_2, X, y, cv=7, scoring='accuracy').
 ↪get('test_score')))
print(np.mean(cross_validate(Ada_one_hot_dropped_2, X_dropped, y_dropped, cv=7,␣
 ↪scoring='accuracy').get('test_score')))
```

```
0.9654743617877258
0.9516494769950992
```

[ ]:
```
Ada_one_hot.fit(x_train_o,y_train_o)
Ada_one_hot_2.fit(x_train_o,y_train_o)

Ada_one_hot_disp = plot_roc_curve(Ada_one_hot, x_test_o, y_test_o)
Ada_one_hot_2_disp = plot_roc_curve(Ada_one_hot_2, x_test_o, y_test_o,␣
 ↪ax=Ada_one_hot_disp.ax_)
Ada_one_hot_2_disp.figure_.suptitle("ROC curve comparison")

plt.figure(figsize=(20,10))
plt.show()
```




```
<Figure size 1440x720 with 0 Axes>
```

5.2 GridSearchCV

Będziemy przeszukiwać parametry: - `n_estimators` - `learning_rate`

oraz ustawimy `random_state` dla uzyskania reprodukowalności.

[ ]:
```
parameters = {
    'ada__n_estimators': [50 * x for x in range(1,11)],
    'ada__learning_rate': [.001, .005, .01, .05, .1, .25],
    'ada__random_state': [997]
}
ada = AdaBoostClassifier()

pipe_ada = Pipeline([('ohe', one_hot_encoder),('ada', ada)])

ada_clf_gs = GridSearchCV(pipe_ada, param_grid=parameters, n_jobs=-1)
ada_clf_gs.fit(X,y)
```

[ ]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('ohe',
```

```
OneHotEncoder(cols=['handicapped_infants',
'water_project_cost_sharing',
'adoption_of_the_budget_resolution',
'physician_fee_freeze',
                                                            'el_salvador_aid',
'religious_groups_in_schools',
'anti_satellite_test_ban',
'aid_to_nicaraguan_contras',
                                                            'mx_missile',
                                                            'immigration',
                                                            'synfuels…
                                                           base_estimator=None,
                                                           learning_rate=1.0,
                                                           n_estimators=50,
                                                           random_state=None))],
                                  verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'ada__learning_rate': [0.001, 0.005, 0.01, 0.05, 0.1,
                                                 0.25],
                         'ada__n_estimators': [50, 100, 150, 200, 250, 300, 350,
                                                400, 450, 500],
                         'ada__random_state': [997]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

Najlepszy klasyfikator AdaBoost oparty o dane z wszystkich kolumn uzyskał skuteczność 96,78%. To wynik nieznacznie lepszy niż uzyskany w czasie strojenia ręcznego.

Parametry najlepszego AdaBoost'a: - `learning_rate`: 0.05 - `n_estimators`: 450

[ ]:
```
print(ada_clf_gs.best_params_)
print(ada_clf_gs.best_score_)
```

```
{'ada__learning_rate': 0.05, 'ada__n_estimators': 450, 'ada__random_state': 997}
0.9678160919540231
```

5.3 GridSearchCV dropped

w tym przypadku stroimy klasyfikator AdaBoost oparty o wybrane kolumny. Siatka parametrów jes taka sama jak w poprzednim przypadku (AdaBoost)

[ ]:
```
parameters = {
    'ada__n_estimators': [50 * x for x in range(1,11)],
    'ada__learning_rate': [.001, .005, .01, .05, .1, .25],
    'ada__random_state': [997]
}
ada = AdaBoostClassifier()
```

```
pipe_ada_dropped = Pipeline([('ohe', one_hot_encoder_dropped),('ada', ada)])

dropped_ada_clf_gs = GridSearchCV(pipe_ada_dropped, param_grid=parameters,␣
 ↪n_jobs=-1)
dropped_ada_clf_gs.fit(X_dropped,y_dropped)
```

```
[ ]: GridSearchCV(cv=None, error_score=nan,
                  estimator=Pipeline(memory=None,
                                     steps=[('ohe',
     OneHotEncoder(cols=['handicapped_infants',
     'adoption_of_the_budget_resolution',
     'physician_fee_freeze',
                                                                 'el_salvador_aid',
     'religious_groups_in_schools',
     'anti_satellite_test_ban',
     'aid_to_nicaraguan_contras',
                                                                 'mx_missile',
     'synfuels_corporation_cutback',
     'education_spending',…
                                                                base_estimator=None,
                                                                learning_rate=1.0,
                                                                n_estimators=50,
                                                                random_state=None))],
                                     verbose=False),
                  iid='deprecated', n_jobs=-1,
                  param_grid={'ada__learning_rate': [0.001, 0.005, 0.01, 0.05, 0.1,
                                                     0.25],
                              'ada__n_estimators': [50, 100, 150, 200, 250, 300, 350,
                                                    400, 450, 500],
                              'ada__random_state': [997]},
                  pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                  scoring=None, verbose=0)
```

Najlepszy klasyfikator AdaBoost oparty o dane z wybranych kolumn uzyskał skuteczność 96.09%. To wynik nieznacznie lepszy niż uzyskany w czasie strojenia ręcznego.

Parametry najlepszego AdaBoost'a: - `learning_rate`: 0.01 - `n_estimators`: 500

```
[ ]: print(dropped_ada_clf_gs.best_params_)
     print(dropped_ada_clf_gs.best_score_)
```

```
{'ada__learning_rate': 0.01, 'ada__n_estimators': 500, 'ada__random_state': 997}
0.960919540229885
```

6 Długotrwałe wyszukiwanie najlepszych parametrów

Zauważyliśmy, że tuning hiperparametrów wymaga dużej mocy obliczeniowej, a w konsekwencji spotrzeba sporo czasu do jego wykonania. Podjęliśmy decyzję o uruchomieniu obliczeń wieczorem i pozostawieniu pracującego komputera na noc. Tworząc ten notebook pracowaliśmy w Google Colab, więc postanowiliśmy wykorzystać możliwość podpięcia się do Google Drive w celu zapisania wyników, na wypadek gdyby sesja w Colab się zakończyła.

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
path = '/content/drive/MyDrive/Studia_dysk/WUM/grid_search_results.txt'
```

Poniżej napisana została funkcja, która przyjmuje model, siatkę parametrów, X, y, nazwę modelu oraz używany encoder. Funkcja tworzy Pipeline z encoderem i modele, następnie wywołuje GridSearchCV z zadaną siatką parametrów i na końcu zapisuje parametry oraz wyniki najlepszego modelu do pliku, aby nie utracić danych w przypadku wyłączenia sesji Google Colab.

[ ]:
```
def grid_search_model(model, params, X, y, name, one_hot_encoder):

    pipe = Pipeline([('ohe', one_hot_encoder),(name, model)])
    clf_gs = GridSearchCV(pipe, param_grid=params, n_jobs=-1)
    clf_gs.fit(X,y)
    with open(path, 'a') as file:
        file.writelines(name + '\n')
        file.writelines(str(clf_gs.best_params_) + '\n')
        file.writelines('score' + str(clf_gs.best_score_) + '\n')
        file.writelines('**********************\n')
```

Tym razem siatki parametrów są gęstsze i pokrywają większy zakres przestrzeni parametrów niż w poprzednich sekcjach.

Każdy z trzech modeli będzie tuningowany w dwóch wariantach: - opartym na wszystkich kolumnach, - opartym na wybranych kolumnach (*dropped*)

[ ]:
```
#ADA

ada_parameters = {
    'ada__n_estimators': [50 * i for i in range(1,21)],
    'ada__learning_rate': [.001, .005, .01, .025, .05, .1, .25, .5],
    'ada__random_state': [997]
}
ada = AdaBoostClassifier()
grid_search_model(ada, ada_parameters, X, y, 'ada', one_hot_encoder)

#ADA drop

dropped_ada_parameters = {
    'dropped_ada__n_estimators': [50 * i for i in range(1,21)],
    'dropped_ada__learning_rate': [.001, .005, .01, .025, .05, .1, .25, .5],
    'dropped_ada__random_state': [997]
```

```
}
dropped_ada = AdaBoostClassifier()
grid_search_model(dropped_ada, dropped_ada_parameters, X_dropped, y_dropped,␣
↪'dropped_ada', one_hot_encoder_dropped)

#XGB

xgb_parameters = {
    'xgb__n_estimators': [50 * x for x in range(1,21)],
    'xgb__max_depth': [x for x in range(1,10)],
    'xgb__learning_rate': [.001, .005, .01, .025, .05, .1, .25],
    'xgb__random_state': [997]
}
xgb = XGBClassifier()
grid_search_model(xgb, xgb_parameters, X, y, 'xgb', one_hot_encoder)

#XGB drop

dropped_xgb_parameters = {
    'xgb_dropped__n_estimators': [50 * x for x in range(1,21)],
    'xgb_dropped__max_depth': [x for x in range(1,10)],
    'xgb_dropped__learning_rate': [.001, .005, .01, .025, .05, .1, .25],
    'xgb_dropped__random_state': [997]
}
dropped_xgb = XGBClassifier()
grid_search_model(dropped_xgb, dropped_xgb_parameters, X_dropped, y_dropped,␣
↪'xgb_dropped', one_hot_encoder_dropped)


# GB
gb_parameters = {
    'gbc__n_estimators': [50 * x for x in range(1,21)],
    'gbc__max_depth': [x for x in range(1,7)],
    'gbc__random_state': [997]
}
gbc = GradientBoostingClassifier()
grid_search_model(gbc, gb_parameters, X, y, 'gbc', one_hot_encoder)

# GB drop
gb_parameters_dropped = {
    'gbc_dropped__n_estimators': [50 * x for x in range(1,21)],
    'gbc_dropped__max_depth': [x for x in range(1,7)],
    'gbc_dropped__random_state': [997]
}
gbc_dropped = GradientBoostingClassifier()
grid_search_model(gbc_dropped, gb_parameters_dropped, X_dropped, y_dropped,␣
↪'gbc_dropped', one_hot_encoder_dropped)
```

6.1 Wyniki przeszukiwania

6.1.1 AdaBoost

- 'ada___learning_rate': 0.025
- 'ada___n_estimators': 900
- 'ada___random_state': 997

score **96.78**160919540231% ********************** ### AdaBoost dropped - 'dropped_ada___learning_rate': 0.005 - 'dropped_ada___n_estimators': 850 - 'dropped_ada___random_state': 997

score **96.09**19540229885% ********************** ### XGBoost - 'xgb___learning_rate': 0.1 - 'xgb___max_depth': 3 - 'xgb___n_estimators': 50 - 'xgb___random_state': 997

score **96.78**160919540231% ********************** ### XGBoost dropped - 'xgb_dropped___learning_rate': 0.025 - 'xgb_dropped___max_depth': 4 - 'xgb_dropped___n_estimators': 400 - 'xgb_dropped___random_state': 997

score **97.01**1494252873356% ********************** ### Gradient Boosting Classifier - 'gbc___max_depth': 1 - 'gbc___n_estimators': 300 - 'gbc___random_state': 997

score **96.78**160919540231% ********************** ### Gradient Boosting Classifier dropped - 'gbc_dropped___max_depth': 3 - 'gbc_dropped___n_estimators': 50 - 'gbc_dropped___random_state': 997

score **97.01**1494252873356% **********************

Zauważmy, że najlepszy wynik został ex-equo osiągnięty przez dwa modele: - XGBoost oparty o dane z wybranych kolumn - GBC oparty o dane z wybranych kolumn

Poniżej stworzyliśmy modele o najlepszych parametrach w celu głębszego ich zbadania.

6.2 Gradient Boosting Classifier

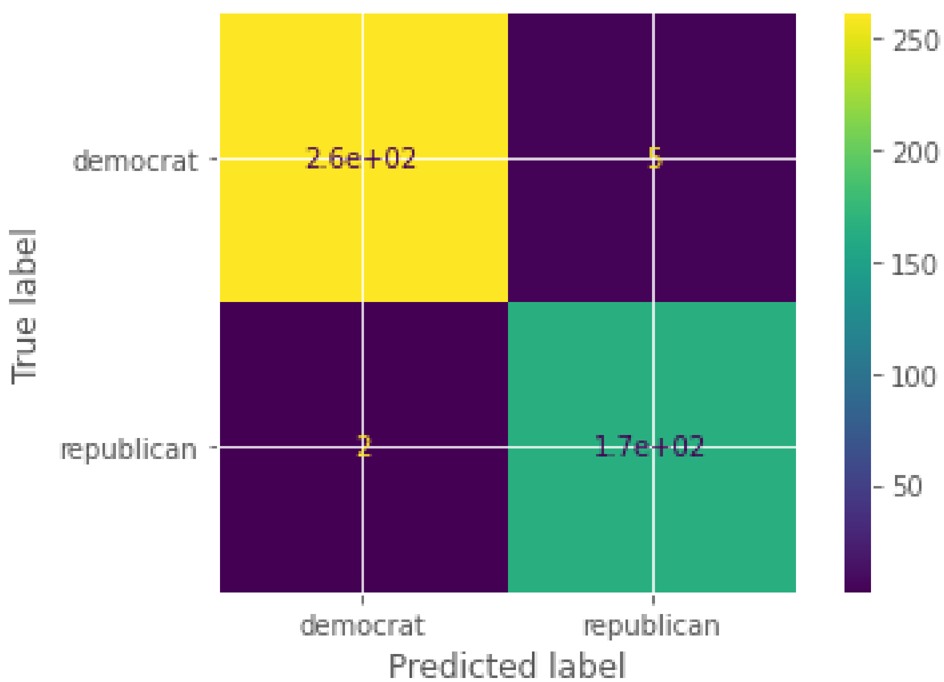
[ ]:
```
gbc = GradientBoostingClassifier(max_depth=3, n_estimators=50, random_state=997)

gbc_pipe = Pipeline([
                    ('one_hot', one_hot_encoder_dropped),
                    ('gbc', gbc)
])

gbc_pipe.fit(x_train, y_train)
```

[ ]:
```
Pipeline(memory=None,
         steps=[('one_hot',
                 OneHotEncoder(cols=['handicapped_infants',
                                     'adoption_of_the_budget_resolution',
                                     'physician_fee_freeze', 'el_salvador_aid',
                                     'religious_groups_in_schools',
                                     'anti_satellite_test_ban',
                                     'aid_to_nicaraguan_contras', 'mx_missile',
```

```
                                          'synfuels_corporation_cutback',
                                          'education_spending',
                                          'superfund_right_to_sue', 'crime'],
                                    drop_inva…
                                                 learning_rate=0.1, loss='deviance',
                                                 max_depth=3, max_features=None,
                                                 max_leaf_nodes=None,
                                                 min_impurity_decrease=0.0,
                                                 min_impurity_split=None,
                                                 min_samples_leaf=1,
                                                 min_samples_split=2,
                                                 min_weight_fraction_leaf=0.0,
                                                 n_estimators=50,
                                                 n_iter_no_change=None,
                                                 presort='deprecated',
                                                 random_state=997, subsample=1.0,
                                                 tol=0.0001, validation_fraction=0.1,
                                                 verbose=0, warm_start=False))],
         verbose=False)
```

[ ]:
```
plot_confusion_matrix(gbc_pipe, X_dropped, y_dropped)
```

[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fa5ac462550>
```



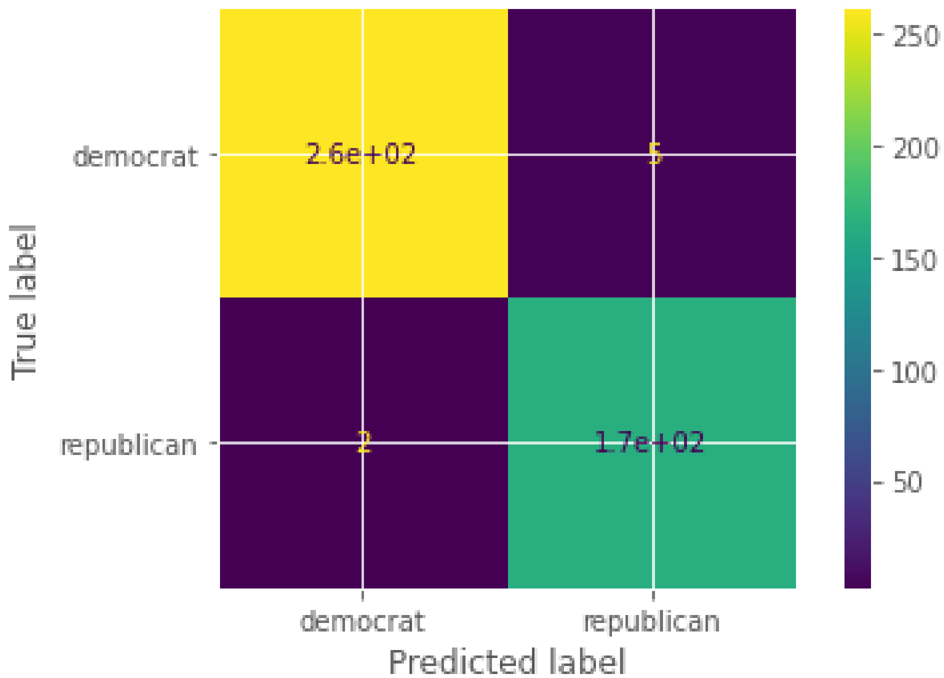
6.3 XGBoost

[ ]:
```
xgb = GradientBoostingClassifier(learning_rate=0.025, max_depth=4,␣
↪n_estimators=400, random_state=997)

xgb_pipe = Pipeline([
                    ('one_hot', one_hot_encoder_dropped),
                    ('xgb', xgb)
])

xgb_pipe.fit(x_train, y_train)
```

[ ]:
```
Pipeline(memory=None,
         steps=[('one_hot',
                 OneHotEncoder(cols=['handicapped_infants',
                                     'adoption_of_the_budget_resolution',
                                     'physician_fee_freeze', 'el_salvador_aid',
                                     'religious_groups_in_schools',
                                     'anti_satellite_test_ban',
                                     'aid_to_nicaraguan_contras', 'mx_missile',
                                     'synfuels_corporation_cutback',
                                     'education_spending',
                                     'superfund_right_to_sue', 'crime'],
                               drop_inva…
                                            loss='deviance', max_depth=4,
                                            max_features=None,
                                            max_leaf_nodes=None,
                                            min_impurity_decrease=0.0,
                                            min_impurity_split=None,
                                            min_samples_leaf=1,
                                            min_samples_split=2,
                                            min_weight_fraction_leaf=0.0,
                                            n_estimators=400,
                                            n_iter_no_change=None,
                                            presort='deprecated',
                                            random_state=997, subsample=1.0,
                                            tol=0.0001, validation_fraction=0.1,
                                            verbose=0, warm_start=False))],
         verbose=False)
```

[ ]:
```
plot_confusion_matrix(xgb_pipe, X_dropped, y_dropped)
```

[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fa5a7283fd0>
```



6.4 Porównanie modeli

Wyniki w macierzy pomyłek są takie same dla obu klasyfikatorów. Sprawdźmy, czy pomyliły się w tych samych obserwacjach.

[ ]:
```
xgb_pred = pd.DataFrame(xgb_pipe.predict(X_dropped))
gbc_pred = pd.DataFrame(gbc_pipe.predict(X_dropped))

xgb_errors = xgb_pred[xgb_pred[0] != y_dropped]
display(xgb_errors)

gbc_errors = gbc_pred[gbc_pred[0] != y_dropped]
display(gbc_errors)
```

```
              0
103  republican
242    democrat
315    democrat
375  republican
382  republican
388  republican
407  republican

              0
168  republican
```

```
242      democrat
267      democrat
375    republican
382    republican
388    republican
407    republican
```

W 5 z 7 przypadków modele popełniły błędy w tych samych obserwacjach. Sprawdźmy z jakimi prawdopodobieństwami modele przewidywały błędnie.

[ ]:
```
xgb_pred_proba = pd.DataFrame(xgb_pipe.predict_proba(X_dropped))
display(xgb_pred_proba[xgb_pred[0] != y_dropped])

gbc_pred_proba = pd.DataFrame(gbc_pipe.predict_proba(X_dropped))
gbc_pred_proba[gbc_pred[0] != y_dropped]
```

	0	1
103	0.499581	0.500419
242	0.994049	0.005951
315	0.839319	0.160681
375	0.496698	0.503302
382	0.480286	0.519714
388	0.006474	0.993526
407	0.168661	0.831339

[ ]:

	0	1
168	0.452113	0.547887
242	0.941438	0.058562
267	0.657365	0.342635
375	0.355217	0.644783
382	0.410204	0.589796
388	0.094742	0.905258
407	0.185291	0.814709

Widać, że w niektórych przypadkach predykcja była blisko granicy 50% (np. obserwacja 382, 168, 103), ale zdarzało się że model pomylił się zupełnie (np. obserwacja 242, 407).

Spójrzmy, co to za obserwacje i skonfrontujmy je z wnioskami z EDA.

[ ]:
```
X_dropped.iloc[242],y_dropped.iloc[242]
```

[ ]:
```
(handicapped_infants                     n
 adoption_of_the_budget_resolution       n
 physician_fee_freeze                    n
 el_salvador_aid                         y
 religious_groups_in_schools             y
 anti_satellite_test_ban                 y
 aid_to_nicaraguan_contras               n
 mx_missile                              n
```

```
synfuels_corporation_cutback         n
education_spending                   ?
superfund_right_to_sue               n
crime                                y
Name: 242, dtype: object, 'republican')
```

W tym przypadku dla modeli prawdopodobnie mylące okazała się kolumna `physician_fee_freeze`. Na wykresie powyżej widać, że tylko ułamek kongresmenów będących republikanami głosowało przeciwko tej ustawie. Poniżej widać, że było ich tylko dwóch.

```
[ ]: X_dropped[(X_dropped['physician_fee_freeze'] == 'n') & (y_dropped ==␣
      ↪'republican')]
```

```
[ ]:    handicapped_infants  … crime
     242                   n  …     y
     267                   y  …     y

     [2 rows x 12 columns]
```

```
[ ]: X_dropped.iloc[407],y_dropped.iloc[407]
```

```
[ ]: (handicapped_infants                  n
      adoption_of_the_budget_resolution    n
      physician_fee_freeze                 y
      el_salvador_aid                      y
      religious_groups_in_schools          y
      anti_satellite_test_ban              n
      aid_to_nicaraguan_contras            n
      mx_missile                           n
      synfuels_corporation_cutback         y
      education_spending                   y
      superfund_right_to_sue               y
      crime                                y
      Name: 407, dtype: object, 'democrat')
```

Obserwacja o indeksie 407 opisuje demokratę, który głosował za ustawą `physician_fee_freeze`, czyli przeciwko większości swojej partii i to było mylące dla modeli.

6.4.1 Wnioski

- model, który na pierwszy rzut oka nie wydawał się najlepszy (Gradient Boosting Classifier), po tuningu hipeparametrów uzyskał najlepsze wyniki. Warto przeprowadzać tuning hiperparametrów wielu modeli.
- w zbiorach danych mogą występować obserwacje, które będą sprawiać problemy z klasyfikacją dla różnych modeli.