Projekt1_Solawa_Smolen_final

April 20, 2021

1 Wstęp do Uczenia Maszynowego - Projekt 1

1.1 Autorzy: Katarzyna Solawa, Jan Smoleń

Tematem naszego projektu jest przewidywanie przynależności partyjnej członka Izby Reprezentantów amerykańskiego kongresu w 1986 roku na podstawie dokonanych przez niego wyborów podczas głosowań. Naszym zbiorem danych jest ramka zawierająca dane o przynależności partyjnej poszczególnych reprezentantów i ich głosach podczas 16 kluczowych w tym roku głosowań.

2 Importy

[1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import sklearn.metrics
import random
from sklearn import manifold
random.seed(42)
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, chi2, mutual_info_classif
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel
from sklearn.svm import LinearSVC
import xgboost as xgb
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import plot_precision_recall_curve
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
import xgboost as xgb
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

```
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import RandomizedSearchCV
import xgboost as xgb
import sklearn.metrics as metrics
import statistics
```

3 EDA

[2]:
```
df=pd.read_csv("congressional_voting_dataset.csv")
```

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 435 entries, 0 to 434
Data columns (total 17 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
 16  political_party                        435 non-null    object
dtypes: object(17)
memory usage: 57.9+ KB
```

[4]:
```
df.head()
```

[4]:

	handicapped_infants	water_project_cost_sharing	\
0	n	y	
1	n	y	
2	?	y	
3	n	y	
4	y	y	

```
  adoption_of_the_budget_resolution physician_fee_freeze el_salvador_aid  \
0                                 n                    y              y
1                                 n                    y              y
2                                 y                    ?              y
3                                 y                    n              ?
4                                 y                    n              y

  religious_groups_in_schools anti_satellite_test_ban  \
0                           y                       n
1                           y                       n
2                           y                       n
3                           y                       n
4                           y                       n

  aid_to_nicaraguan_contras mx_missile immigration  \
0                         n          n           y
1                         n          n           n
2                         n          n           n
3                         n          n           n
4                         n          n           n

  synfuels_corporation_cutback education_spending superfund_right_to_sue  \
0                            ?                  y                      y
1                            n                  y                      y
2                            y                  n                      y
3                            y                  n                      y
4                            y                  ?                      y

  crime duty_free_exports export_administration_act_south_africa  \
0     y                 n                                      y
1     y                 n                                      ?
2     y                 n                                      n
3     n                 n                                      y
4     y                 y                                      y

  political_party
0      republican
1      republican
2        democrat
3        democrat
4        democrat
```

3.1 Objaśnienie zmiennych

Kolumny 0-15 zawierają wyniki głosowań na tematy skrótowo opisane w nazwach kolumn. Każdy rząd odpowiada jednemu reprezentantowi. Możliwe wartości: **y** - głos na tak **n** - głos na nie **?** - brak głosu - niewzięcie udziału w głosowaniu lub wstrzymanie się od głosu Ostatnia kolumna

zawiera informacje o przynależności partyjnej reprezentanta - **republican** albo **democrat**. W naszej ramce danych nie występuje bezpośrednio problem braku danych, ale zapewne będzie trzeba jakoś rozwiązać kwestię wartości **?**.

[5]:
```
df.describe()
```

[5]:

	handicapped_infants	water_project_cost_sharing
count	435	435
unique	3	3
top	n	y
freq	236	195

	adoption_of_the_budget_resolution	physician_fee_freeze	el_salvador_aid
count	435	435	435
unique	3	3	3
top	y	n	y
freq	253	247	212

	religious_groups_in_schools	anti_satellite_test_ban
count	435	435
unique	3	3
top	y	y
freq	272	239

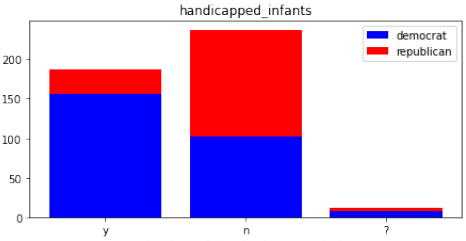
	aid_to_nicaraguan_contras	mx_missile	immigration
count	435	435	435
unique	3	3	3
top	y	y	y
freq	242	207	216

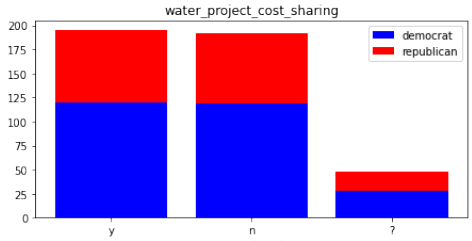
	synfuels_corporation_cutback	education_spending	superfund_right_to_sue
count	435	435	435
unique	3	3	3
top	n	n	y
freq	264	233	209

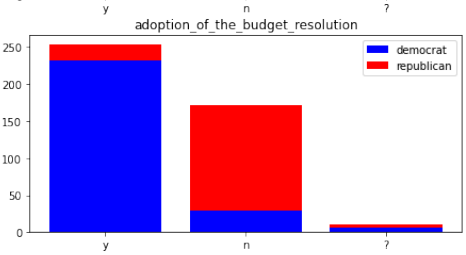
	crime	duty_free_exports	export_administration_act_south_africa
count	435	435	435
unique	3	3	3
top	y	n	y
freq	248	233	269

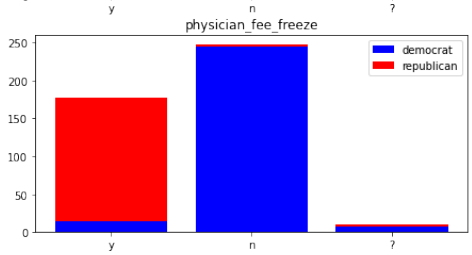
	political_party
count	435
unique	2
top	democrat
freq	267

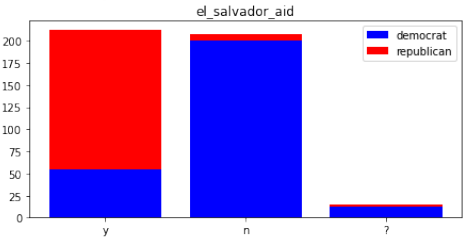
[6]:
```
labels=["y", "n", "?"]
fig, axs = plt.subplots(ncols=2, nrows=8, figsize=(16, 32))
for i in range(len(df.columns)-1):
    col=df.columns[i]
    tmp=df[[col, "political_party"]].groupby(["political_party", col]).size().
 ↪tolist()
    r, c= i//2, i%2
    axs[r,c].bar(labels, list(reversed(tmp[0:3])), label='democrat',␣
 ↪color="blue")
    axs[r,c].bar(labels, list(reversed(tmp[3:6])), bottom=list(reversed(tmp[0:
 ↪3])),
        label='republican', color="red")
    axs[r,c].legend()
    axs[r,c].set_title(col)
```

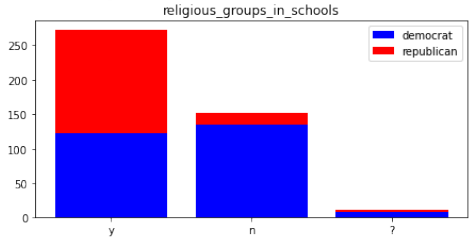

handicapped_infants
democrat
republican
y
n
?

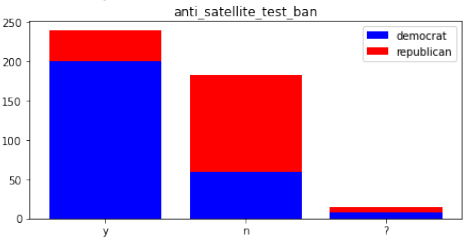

water_project_cost_sharing
democrat
republican
y
n
?

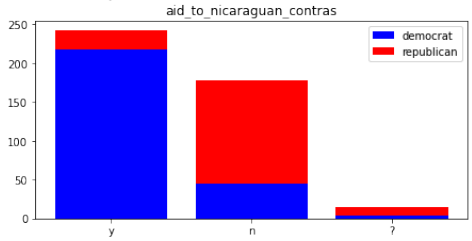

adoption_of_the_budget_resolution
democrat
republican
y
n
?

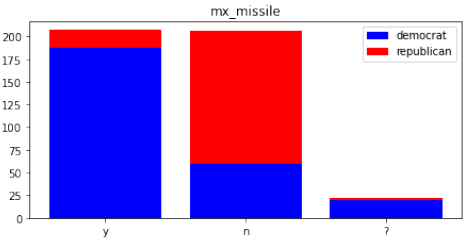

physician_fee_freeze
democrat
republican
y
n
?

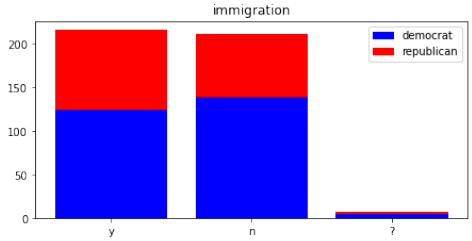

el_salvador_aid
democrat
republican
y
n
?

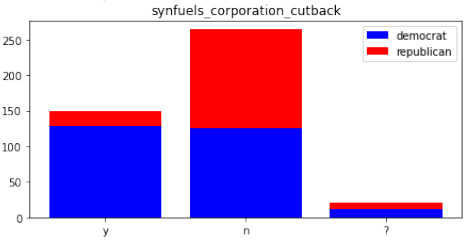

religious_groups_in_schools
democrat
republican
y
n
?

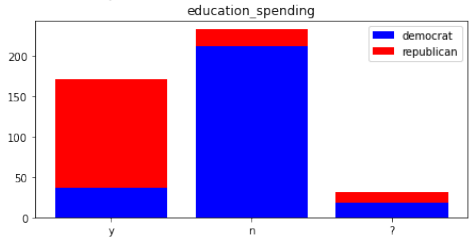

anti_satellite_test_ban
democrat
republican
y
n
?

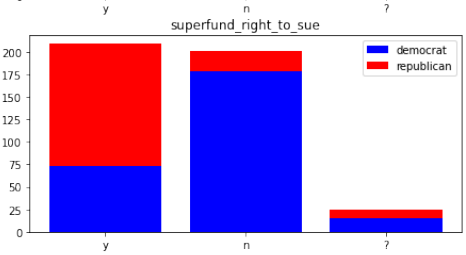

aid_to_nicaraguan_contras
democrat
republican
y
n
?

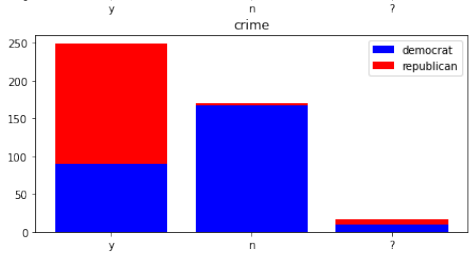

mx_missile
democrat
republican
y
n
?

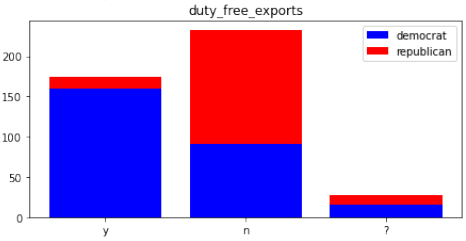

immigration
democrat
republican
y
n
?

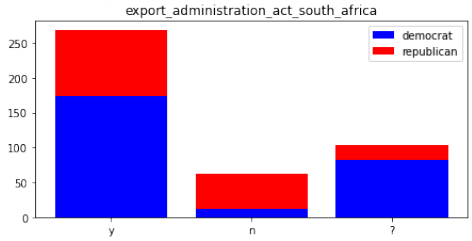

synfuels_corporation_cutback
democrat
republican
y
n
?


education_spending
democrat
republican
y
n
?


superfund_right_to_sue
democrat
republican
y
n
?

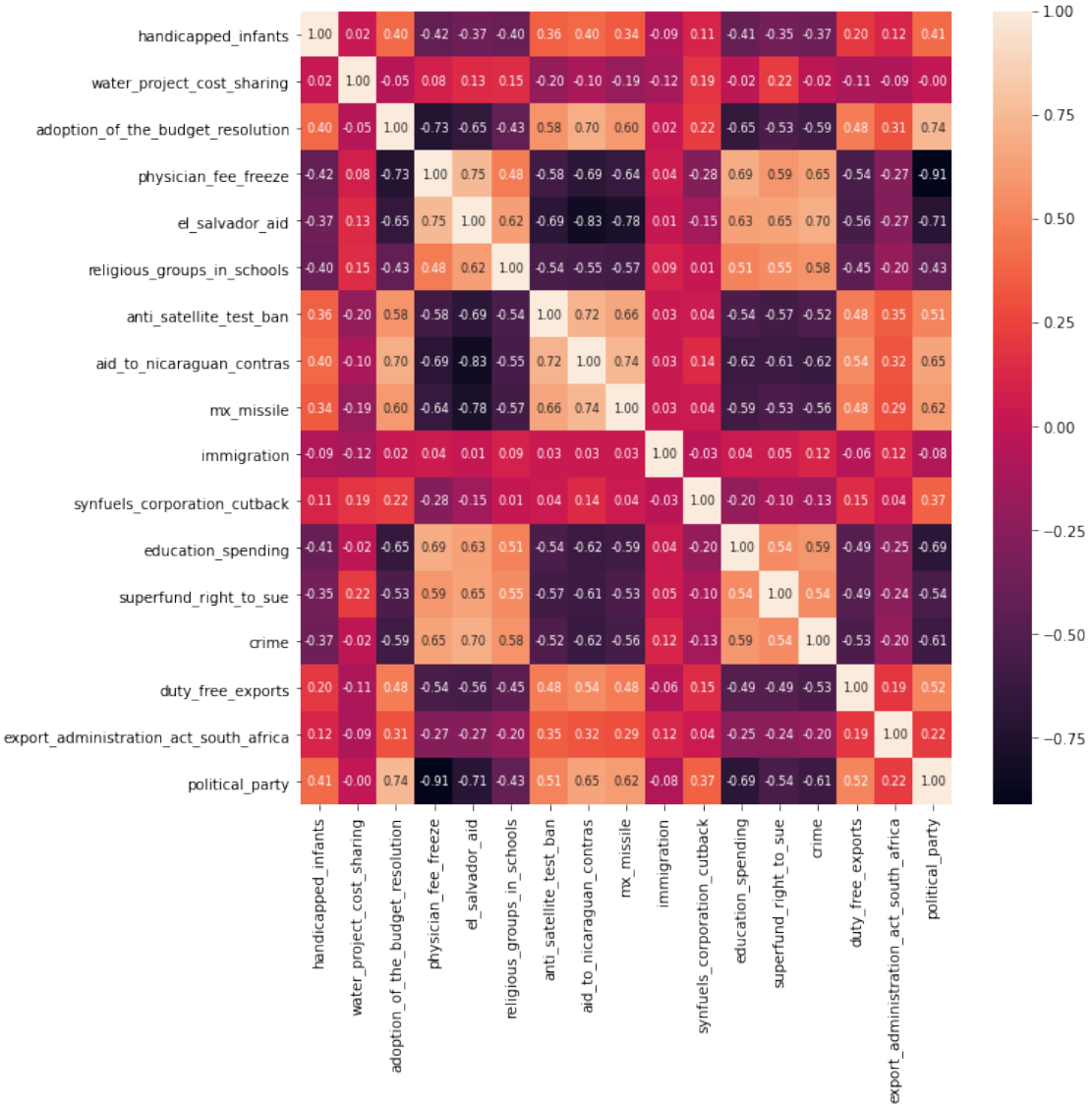

crime
democrat
republican
y
n
?


duty_free_exports
democrat
republican
y
n
?


export_administration_act_south_africa
democrat
republican
y
n
?

Obie partie głosowały podobnie na `water_project_cost_sharing` oraz `imigration`(lecz u demokratów przeważa `no`, a u republikan `yes`) Widoczna róznica głosów dla: - `adoption_of_the_budget_resolution`(r-no, d-yes) - `physician_fee_freeze`(r-yes, d-no) - `el_salvador_aid`(r-yes, d-no) - `education_spending`(r-yes, d-no)

[7]:
```
df=df.replace("n", 0)
df=df.replace("y", 1)
df=df.replace("?",  0.5)    #rozwiązanie tymczasowe
df=df.replace("republican", 0)
df=df.replace("democrat",  1)
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(), annot=True, annot_kws={'size': 8}, fmt='.2f')
plt.show()
```



Jak widzimy, poziom korelacjii pomiędzy głosem a partią bardzo się różni w zależności od tematu głosowania - dla głosowania **water_project_cost_sharing** związek praktycznie nie istnieje, a dla **physician_fee_freeze** jest bardzo duży.

Spróbujemy teraz zobaczyć, na ile głosy poszczególnych reprezentantów przypominają głosy innych członków tej samej partii - w tym celu przekształcimy zapisy głosowań poszczególnych członków na wektory i policzymy odległości pomiędzy każdą parą.

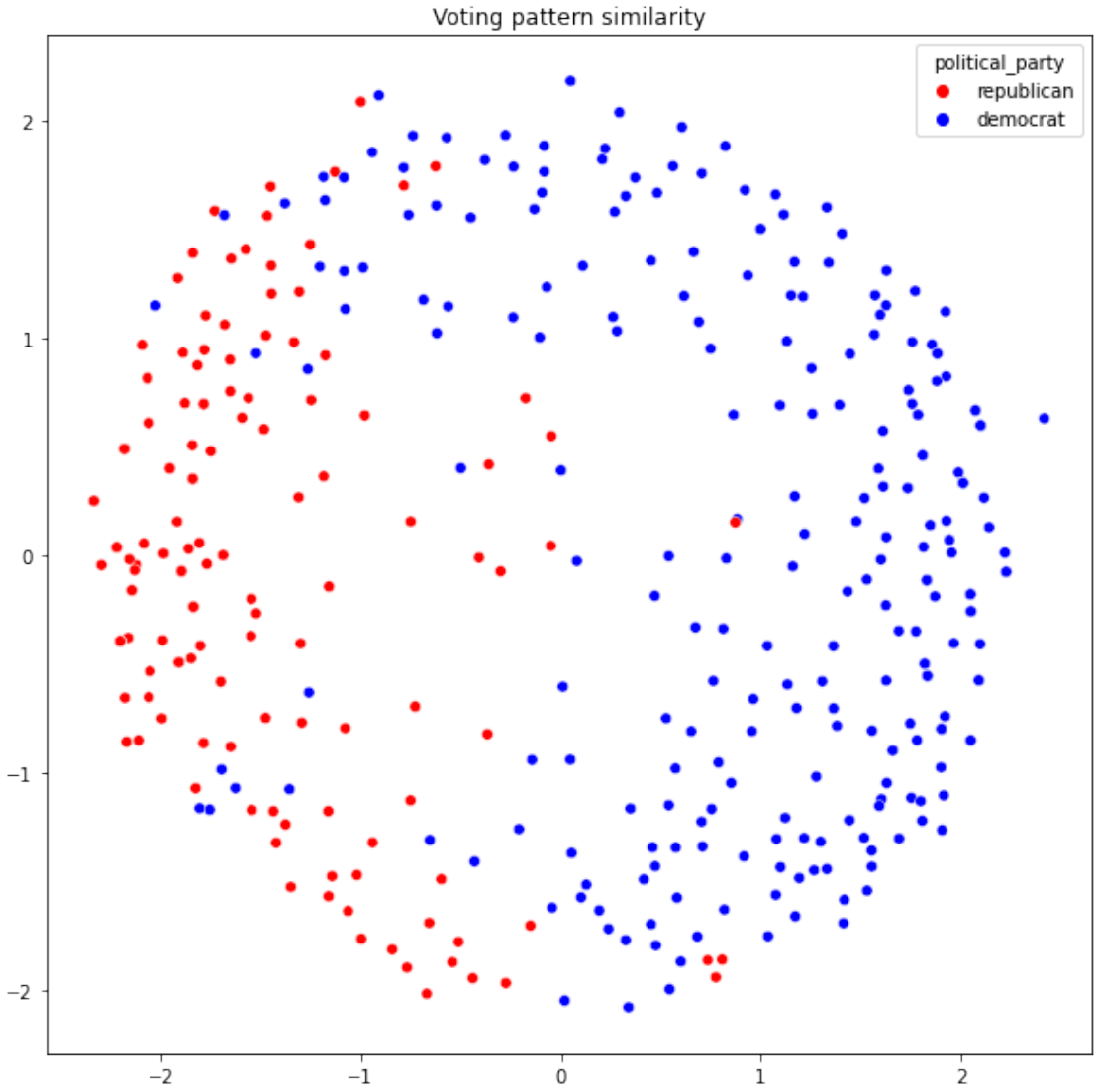
[8]:
```
adist=sklearn.metrics.pairwise_distances(df.drop(["political_party"], axis=1))
adist
```

[8]:
```
array([[0.        , 1.22474487, 2.17944947, …, 1.22474487, 1.5       ,
        1.22474487],
       [1.22474487, 0.        , 1.93649167, …, 1.22474487, 1.5       ,
        1.22474487],
       [2.17944947, 1.93649167, 0.        , …, 1.93649167, 2.54950976,
        2.17944947],
       …,
       [1.22474487, 1.22474487, 1.93649167, …, 0.        , 1.5       ,
        1.87082869],
       [1.5       , 1.5       , 2.54950976, …, 1.5       , 0.        ,
        1.80277564],
       [1.22474487, 1.22474487, 2.17944947, …, 1.87082869, 1.80277564,
        0.        ]])
```

Użyjemy teraz funkcji z pakietu manifold żeby przekształcić ramkę zawierającą wzajemne odległości na zbiór współrzędnych na dwuwymiarowej płaszczyźnie. Jest to rzut, który próbuje przekształcić wielowymiarowe zależności na płaszczyznę 2D.

[9]:
```
df["political_party"]=df["political_party"].replace(0, "republican")
df["political_party"]=df["political_party"].replace(1, "democrat")
adist=np.array(adist)
mds = manifold.MDS(n_components=2, dissimilarity="precomputed", random_state=6)
results = mds.fit(adist)
coords = results.embedding_
fig, ax = plt.subplots(figsize=(10,10))
sns.scatterplot(
    coords[:, 0], coords[:, 1], marker = 'o', hue=df["political_party"],␣
 ↪palette=["red", "blue"]
    )
ax.set_title("Voting pattern similarity")
```

[9]:
```
Text(0.5, 1.0, 'Voting pattern similarity')
```

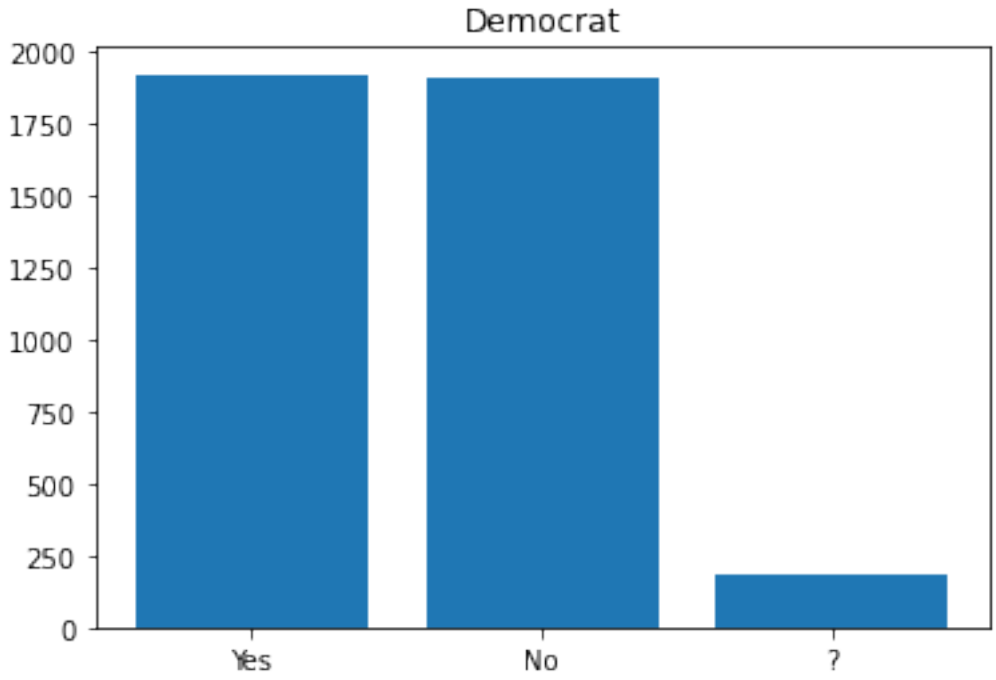


Dodatkowo sprawdźmy czy któraś z parti ma skołonność do głosowania na tak lub nie.

[10]:
```
df=pd.read_csv("congressional_voting_dataset.csv")
democrat_df = df[df['political_party'] == 'democrat']
republican_df = df[df['political_party'] == 'republican']
```

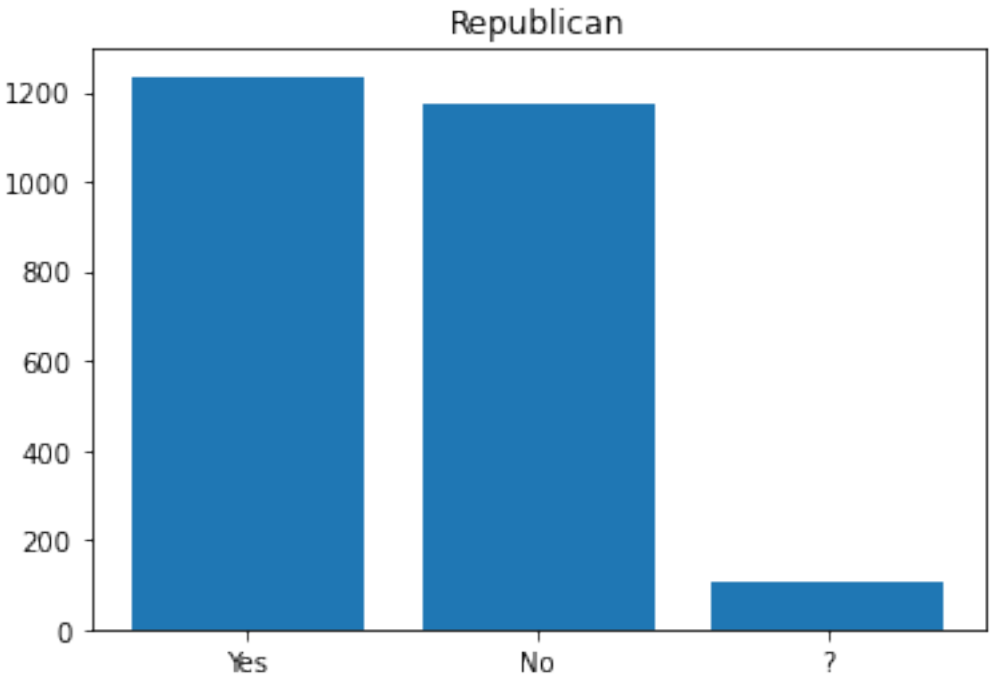
[11]:
```
tak = 0
nie = 0
brak = 0
for i in range(0,15):
    tak += (democrat_df[democrat_df.columns[i]] == "y").sum()
    nie += (democrat_df[democrat_df.columns[i]] == "n").sum()
    brak += (democrat_df[democrat_df.columns[i]] == "?").sum()
```

```
labels = ['Yes', 'No', '?']
sizes = [tak, nie,brak]
plt.title("Democrat")
plt.bar(labels, sizes)
plt.show()
```



[12]:
```
tak = 0
nie = 0
brak = 0
for i in range(0,15):
    tak += (republican_df[republican_df.columns[i]] == "y").sum()
    nie += (republican_df[republican_df.columns[i]] == "n").sum()
    brak += (republican_df[republican_df.columns[i]] == "?").sum()

labels = ['Yes', 'No', '?']
sizes = [tak, nie,brak]
plt.title("Republican")
plt.bar(labels, sizes)
plt.show()
```



W obu przypadkach licza głosów jest dość wyrównana.

4 Feature Engineering

[13]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import sklearn.metrics
import random
from sklearn import manifold
import xgboost as xgb
random.seed(42)
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import chi2
from matplotlib import pyplot
```

[14]:
```
df=pd.read_csv("congressional_voting_dataset.csv")
```

4.0.1 Encoding

W naszych danych kodowanie zmiennych kategorycznych wydaje się nie być dużym wyzwaniem. Głosy na nie oznaczamy jako 0, brak głosu jako 0.5, a głosy na tak to 1. Podobnie intuicyjnie republikanów oznaczamy jako zera, a demokratów jako jedynki.

[15]:
```
df=df.replace("n", 0)
df=df.replace("y", 1)
df=df.replace("?",  0.5)
df=df.replace("republican", 0)
df=df.replace("democrat",  1)
```

4.0.2 Outliers

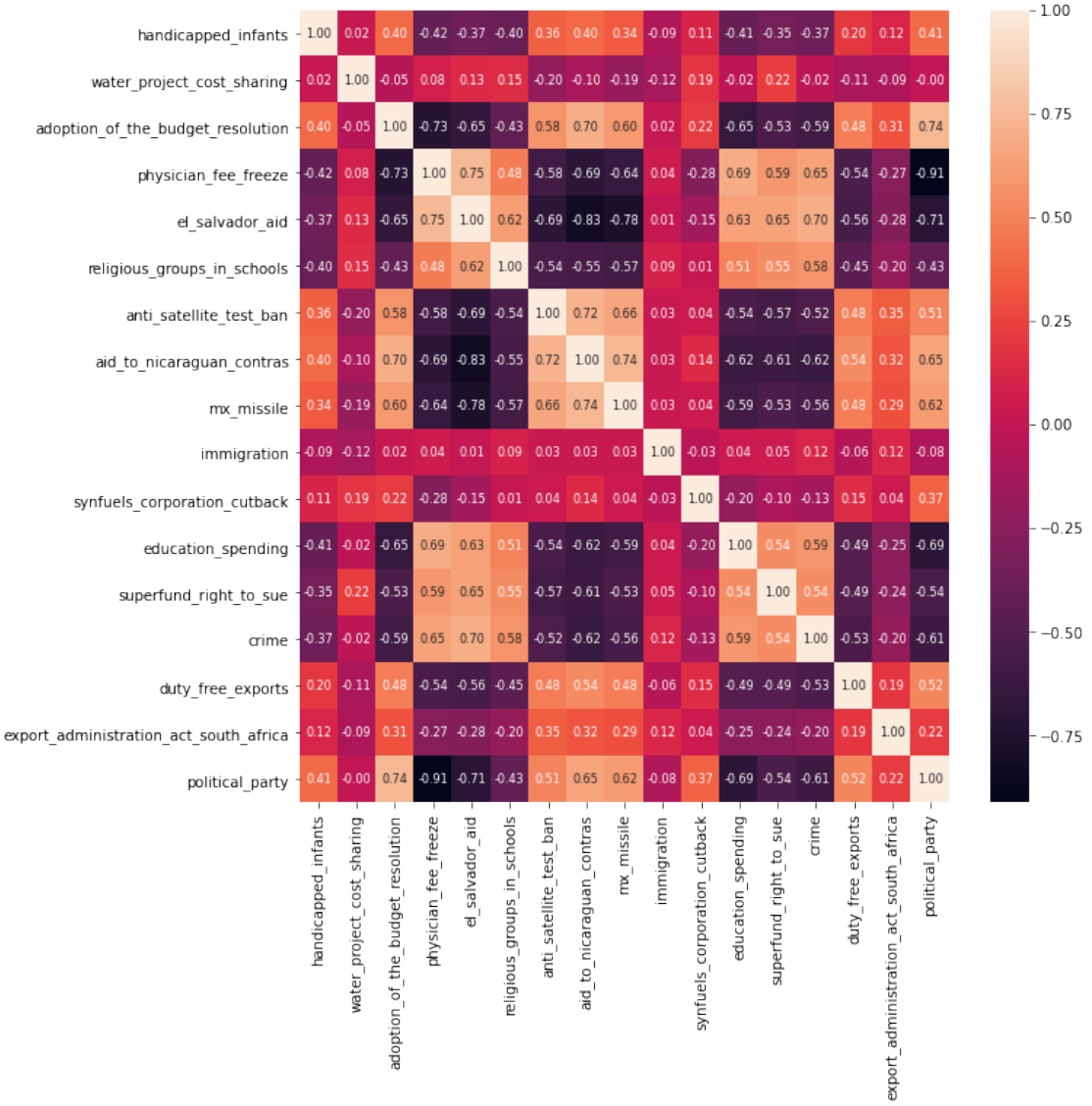
Ze względu na kategoryczne wartości w naszych danych, nie widzimy tu outlierów w postaci rzędów, które się szczególnie wyróżniają jedną wartością. Jedyny rząd, który odrzucimy to ten, w którym wartości wszystkich głosowań wynosiły "?" - jest to prawdopodobnie brak danych, bądź dany reprezentant z jakiś osobliwych powodów nie wziął udziału w żadnym głosowaniu.

[16]:
```
X=df.drop(["political_party"], axis=1)
indexes=[]
colnames=X.columns
for i in range(len(X)):
    for j in range(len(colnames)):
        if X.iloc[i, j]!=0.5:
            break
        if j==len(colnames)-1:
            indexes.append(i)
X=X.drop(248, axis=0)
y=df["political_party"].drop(248, axis=0)
df=df.drop(248, axis=0)
df.to_csv("df_encoded.csv", index=False)
```

5 Feature Selection

Na początku korzystaliśmy z bardziej intycuicyjnych sposobów wyborów cech, a potem zastosowaliśmy metody pokazane na laboratoriach. Spojrzymy najpierw ponownie na macierz korelacji.

[17]:
```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(), annot=True, annot_kws={'size': 8}, fmt='.2f')
plt.show()
```



Usuniemy dwie zmienne, które w porównaniu z innymi są bardzo mało skorelowane z naszym celem - **water_project_cost_sharing** i **immigration**. Spróbujemy też usunąć zmienną **el_salvador_aid** - mimo, że jest silnie związana z celem, jest także najbardziej skorelowana z innymi zmiennymi objaśniającymi.

[18]:
```
drop=["water_project_cost_sharing", "immigration", "el_salvador_aid"]
X=X.drop(drop, axis=1)
```

5.0.1 Model

Naszym modelem bazowym, którego dziś użyjemy, będzie xgboost bez tuningu hiperparametrów.

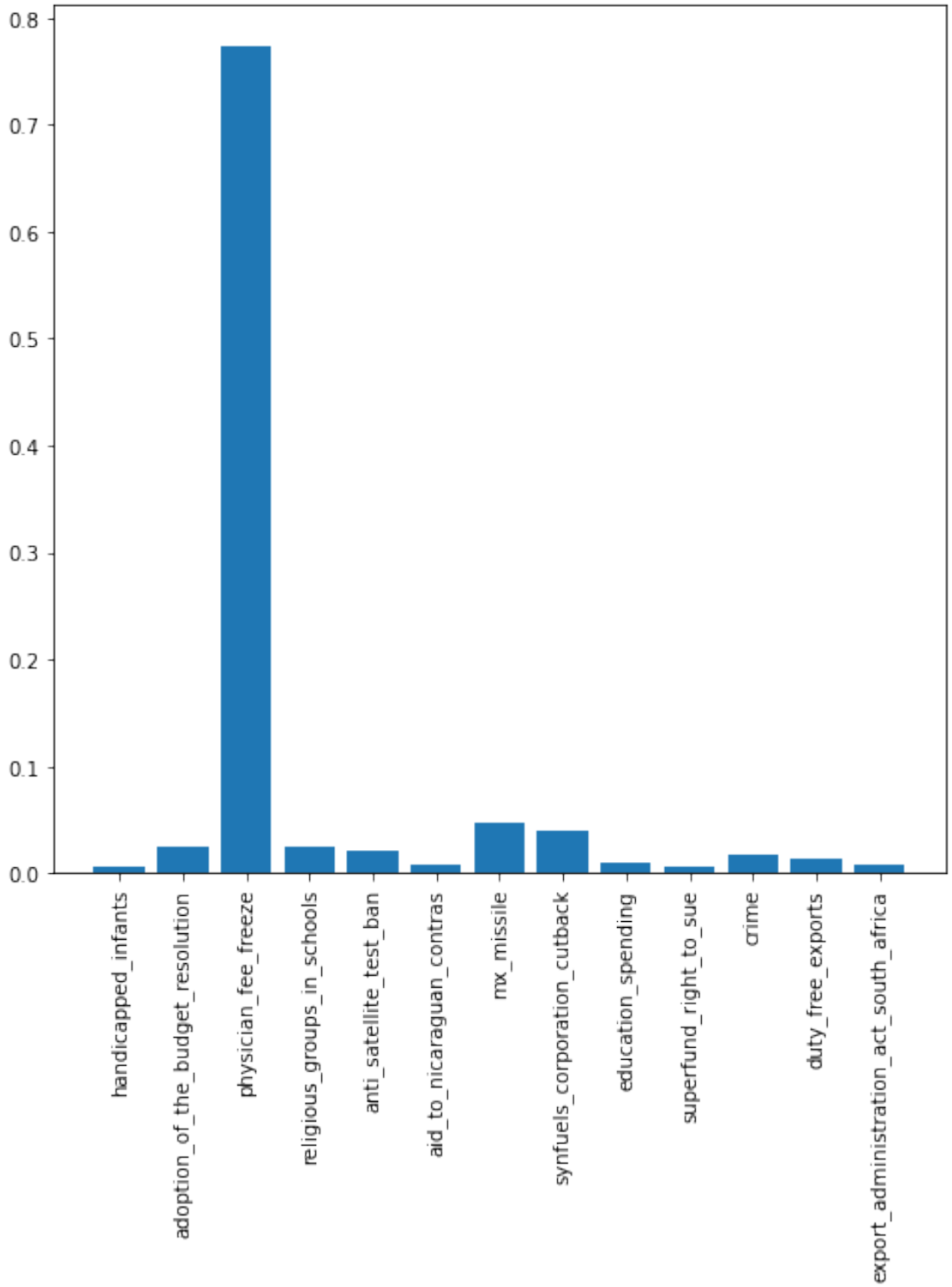
[19]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size␣
↪= 0.2, random_state = 42)
```

[20]:
```
xgb_model = xgb.XGBClassifier(objective = "binary:logistic", seed = 42,␣
↪use_label_encoder=False, verbosity=0)
xgb_model.fit(X_train, y_train)
preds = xgb_model.predict(X_test)
comparison = pd.DataFrame({'actual':y_test, 'predicted':preds})
print("Accuracy: " + str(sum(comparison["actual"] == comparison["predicted"]) /␣
↪len(comparison) * 100) + "%")
```

```
Accuracy: 96.55172413793103%
```

Jak widzimy, nasz model całkiem dobrze sobie radzi z przewidywaniem przynależności do danej partii politycznej - osiąga ponad 96% skuteczności. Na koniec spojrzymy, jak ważne dla niego są poszczególne kolumny - użyjemy do tego wbudowanej funkcji modelu xgb.

[21]:
```
plt.figure(figsize=(8,8))
pyplot.bar(X.columns, xgb_model.feature_importances_)
plt.xticks(rotation=90)
pyplot.show()
```



Zgodnie z oczekiwaniami wynikającymi z mapy korelacji, zmienna **physicican_fee_freeze** ma olbrzymi wpływ na predykcje naszego modelu. Na koniec spójrzmy jeszcze, jak wyglądała by

skuteczność modelu, gdybyśy wybrali jedynie 5 najważniejszych wg wykresu cech.

[22]:
```
fts=["physician_fee_freeze", "mx_missile", "synfuels_corporation_cutback",␣
 ↪"religious_groups_in_schools", "crime"]
X=X[fts]
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size␣
 ↪= 0.2, random_state = 42)
xgb_model = xgb.XGBClassifier(objective = "binary:logistic", seed = 42,␣
 ↪use_label_encoder=False, verbosity=0)
xgb_model.fit(X_train, y_train)
preds = xgb_model.predict(X_test)
comparison = pd.DataFrame({'actual':y_test, 'predicted':preds})
print("Accuracy: " + str(sum(comparison["actual"] == comparison["predicted"]) /␣
 ↪len(comparison) * 100) + "%")
```

```
Accuracy: 94.25287356321839%
```

[23]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import sklearn.metrics
import random
from sklearn import manifold
random.seed(42)
```

[24]:
```
df=pd.read_csv("congressional_voting_dataset.csv")
df=df.replace("n", 0)
df=df.replace("y", 1)
df=df.replace("?",  0.5)
df=df.replace("republican", 0)
df=df.replace("democrat",  1)
X=df.drop(["political_party"], axis=1)
indexes=[]
colnames=X.columns
for i in range(len(X)):
    for j in range(len(colnames)):
        if X.iloc[i, j]!=0.5:
            break
        if j==len(colnames)-1:
            indexes.append(i)
X=X.drop(248, axis=0)
y=df["political_party"].drop(248, axis=0)
df.head()
```

[24]:

	handicapped_infants	water_project_cost_sharing	\
0	0.0	1.0	

```
1                  0.0                          1.0
2                  0.5                          1.0
3                  0.0                          1.0
4                  1.0                          1.0

   adoption_of_the_budget_resolution  physician_fee_freeze  el_salvador_aid  \
0                                0.0                   1.0              1.0
1                                0.0                   1.0              1.0
2                                1.0                   0.5              1.0
3                                1.0                   0.0              0.5
4                                1.0                   0.0              1.0

   religious_groups_in_schools  anti_satellite_test_ban  \
0                          1.0                      0.0
1                          1.0                      0.0
2                          1.0                      0.0
3                          1.0                      0.0
4                          1.0                      0.0

   aid_to_nicaraguan_contras  mx_missile  immigration  \
0                        0.0         0.0          1.0
1                        0.0         0.0          0.0
2                        0.0         0.0          0.0
3                        0.0         0.0          0.0
4                        0.0         0.0          0.0

   synfuels_corporation_cutback  education_spending  superfund_right_to_sue  \
0                           0.5                 1.0                     1.0
1                           0.0                 1.0                     1.0
2                           1.0                 0.0                     1.0
3                           1.0                 0.0                     1.0
4                           1.0                 0.5                     1.0

   crime  duty_free_exports  export_administration_act_south_africa  \
0    1.0                0.0                                     1.0
1    1.0                0.0                                     0.5
2    1.0                0.0                                     0.0
3    0.0                0.0                                     1.0
4    1.0                1.0                                     1.0

   political_party
0                0
1                0
2                1
3                1
4                1
```

[25]:
```
drop=["water_project_cost_sharing", "immigration", "el_salvador_aid"]
X=X.drop(drop, axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size␣
↪= 0.2, random_state = 42)
X_best=None
```

[26]:
```
def selectFeature(selector):
    selector.fit(X_train, y_train)
    X_train_fs = selector.transform(X_train)
    X_test_fs = selector.transform(X_test)
    xgb_model = xgb.XGBClassifier(objective = "binary:logistic", seed = 42,␣
↪use_label_encoder=False, verbosity=0)
    xgb_model.fit(X_train_fs, y_train)
    yhat = xgb_model.predict(X_test_fs)
    accuracy = accuracy_score(y_test, yhat)
    print('Accuracy: %.2f' % (accuracy*100))
```

[27]:
```
def selectFeature2(selector):
    clf = Pipeline([
      ('feature_selection', SelectFromModel(selector)),
      ('classification', xgb.XGBClassifier(objective = "binary:logistic", seed␣
↪= 42, use_label_encoder=False, verbosity=0))
    ])
    clf.fit(X_train, y_train)
    yhat = clf.predict(X_test)
    accuracy = accuracy_score(y_test, yhat)
    print('Accuracy: %.2f' % (accuracy*100))
```

5.0.2 CHI2

[28]:
```
for i in range(1,14):
    selectFeature(SelectKBest(chi2, k=i))
```

```
Accuracy: 94.25
Accuracy: 94.25
Accuracy: 94.25
Accuracy: 93.10
Accuracy: 91.95
Accuracy: 90.80
Accuracy: 90.80
Accuracy: 93.10
Accuracy: 94.25
Accuracy: 94.25
Accuracy: 95.40
Accuracy: 95.40
Accuracy: 96.55
```

Najwyższe accuracy dla wyboru wszytkich 13 zmiennych

5.0.3 Recursive Feature Elimination

[29]:
```
from sklearn.feature_selection import RFE
estimator = LogisticRegression()
for i in range(1,14):
    selectFeature(RFE(estimator, n_features_to_select=i, step=1))
```

```
Accuracy: 94.25
Accuracy: 94.25
Accuracy: 95.40
Accuracy: 96.55
Accuracy: 96.55
Accuracy: 94.25
Accuracy: 95.40
Accuracy: 95.40
Accuracy: 95.40
Accuracy: 95.40
Accuracy: 96.55
Accuracy: 96.55
Accuracy: 96.55
```

Najwyższe accuracy przy wyborze min 4 zmiennych

5.0.4 Mutual info classif

[30]:
```
for i in range(1,14):
    selectFeature(SelectKBest(mutual_info_classif, k=i))
```

```
Accuracy: 94.25
Accuracy: 94.25
Accuracy: 94.25
Accuracy: 94.25
Accuracy: 94.25
Accuracy: 91.95
Accuracy: 91.95
Accuracy: 93.10
Accuracy: 94.25
Accuracy: 94.25
Accuracy: 95.40
Accuracy: 95.40
Accuracy: 96.55
```

Najwyższe accuracy przy wyborze min 10 zmiennych

5.0.5 LinearSVC - Linear Support Vector Classification

[31]:
```
# C = 0.005
selectFeature2(LinearSVC(C=0.005, penalty="l1", dual=False))

# C = 0.005
selectFeature2(LinearSVC(C=0.01, penalty="l1", dual=False))

# C = 0.11
selectFeature2(LinearSVC(C=0.11, penalty="l1", dual=False))

# C = 0.17
selectFeature2(LinearSVC(C=0.17, penalty="l1", dual=False))

# C = 0.2
selectFeature2(LinearSVC(C=0.2, penalty="l1", dual=False))
```

```
Accuracy: 94.25
Accuracy: 94.25
Accuracy: 95.40
Accuracy: 95.40
Accuracy: 95.40
```

[32]:
```
selectFeature2(LinearSVC())#domyślnie C = 1, penalty='l2'
```

```
Accuracy: 95.40
```

5.0.6 PolynomialFeatures

[33]:
```
pf = PolynomialFeatures(degree=3)
pf.fit(X_train, y_train)
X_train_fs = pf.transform(X_train)
X_test_fs = pf.transform(X_test)
xgb_model = xgb.XGBClassifier(objective = "binary:logistic", seed = 42,␣
 ↪use_label_encoder=False, verbosity=0)
xgb_model.fit(X_train_fs, y_train)
yhat = xgb_model.predict(X_test_fs)
accuracy = accuracy_score(y_test, yhat)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Accuracy: 96.55
```

[34]:
```
def selectFeaturePF(selector):
    pf = PolynomialFeatures(degree=3)
    pf.fit(X_train, y_train)
    X_train_pf = pf.transform(X_train)
    X_test_pf = pf.transform(X_test)
    selector.fit(X_train_pf, y_train)
```

```
    X_train_fs = selector.transform(X_train_pf)
    X_test_fs = selector.transform(X_test_pf)
    xgb_model = xgb.XGBClassifier(objective = "binary:logistic", seed = 42,␣
 ↪use_label_encoder=False, verbosity=0)
    xgb_model.fit(X_train_fs, y_train)
    yhat = xgb_model.predict(X_test_fs)
    accuracy = accuracy_score(y_test, yhat)
    print('Accuracy: %.2f' % (accuracy*100))
```

[35]:
```
for i in range(1,20):
    print(f'i = {i}')
    selectFeaturePF(SelectKBest(chi2, k=i))

print(f'i = {42}')
selectFeaturePF(SelectKBest(chi2, k=42))
```

```
i = 1
Accuracy: 94.25
i = 2
Accuracy: 94.25
i = 3
Accuracy: 94.25
i = 4
Accuracy: 94.25
i = 5
Accuracy: 94.25
i = 6
Accuracy: 94.25
i = 7
Accuracy: 94.25
i = 8
Accuracy: 94.25
i = 9
Accuracy: 94.25
i = 10
Accuracy: 94.25
i = 11
Accuracy: 94.25
i = 12
Accuracy: 94.25
i = 13
Accuracy: 94.25
i = 14
Accuracy: 94.25
i = 15
Accuracy: 94.25
i = 16
```

```
Accuracy: 94.25
i = 17
Accuracy: 94.25
i = 18
Accuracy: 94.25
i = 19
Accuracy: 94.25
i = 42
Accuracy: 96.55
```

[36]:
```
for i in range(1,20):
    print(f'i = {i}')
    selectFeaturePF(SelectKBest(mutual_info_classif, k=i))

print(f'i = {37}')
selectFeaturePF(SelectKBest(mutual_info_classif, k=37))
```

```
i = 1
Accuracy: 94.25
i = 2
Accuracy: 94.25
i = 3
Accuracy: 94.25
i = 4
Accuracy: 94.25
i = 5
Accuracy: 94.25
i = 6
Accuracy: 94.25
i = 7
Accuracy: 94.25
i = 8
Accuracy: 94.25
i = 9
Accuracy: 94.25
i = 10
Accuracy: 94.25
i = 11
Accuracy: 94.25
i = 12
Accuracy: 94.25
i = 13
Accuracy: 94.25
i = 14
Accuracy: 94.25
i = 15
Accuracy: 94.25
i = 16
```

```
Accuracy: 94.25
i = 17
Accuracy: 94.25
i = 18
Accuracy: 94.25
i = 19
Accuracy: 94.25
i = 37
Accuracy: 94.25
```

Przy wyborze do k najlepszych zmiennych za pomocą chi2 oraz mutual info classif, accuracy jest większe niż 94.25 dla k min równego 42 i 37.

5.1 Wnioski

Metody osiągają max accuracy 96.55, czyli tyle ile udało nam sie osiągnąć przy samodzielnym wyborze zmiennych.
Używając niedużej ilości zmiennych udało nam się zatem osiągnąć wynik tylko nieznacznie gorszy od bazowego XGBoosta, który wynosił 97%. Jednak ze względu na małą liczbę rekordów i krótki czas wykonywania algorytmów, nie widzieliśmy sensu w ograniczaniu w tym przypadku liczby kolumn.

[37]:
```
fts=["physician_fee_freeze","synfuels_corporation_cutback",␣
 ↪"adoption_of_the_budget_resolution", "education_spending"]
X2=X[fts]
X_train, X_test, y_train, y_test = train_test_split(X2, y, stratify=y,␣
 ↪test_size = 0.2, random_state = 42)
xgb_model = xgb.XGBClassifier(objective = "binary:logistic", seed = 42,␣
 ↪use_label_encoder=False, verbosity=0)
xgb_model.fit(X_train, y_train)
preds = xgb_model.predict(X_test)
comparison = pd.DataFrame({'actual':y_test, 'predicted':preds})
print("Accuracy: " + str(sum(comparison["actual"] == comparison["predicted"]) /␣
 ↪len(comparison) * 100) + "%")
```

```
Accuracy: 96.55172413793103%
```

[38]:
```
xgb_tmp=xgb.XGBClassifier(objective = "binary:logistic", seed = 42,␣
 ↪use_label_encoder=False, verbosity=0)
```

[39]:
```
xgb_tmp
```

[39]:
```
XGBClassifier(base_score=None, booster=None, colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=None, gamma=None,
              gpu_id=None, importance_type='gain', interaction_constraints=None,
              learning_rate=None, max_delta_step=None, max_depth=None,
              min_child_weight=None, missing=nan, 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, seed=42, subsample=None, tree_method=None,
                use_label_encoder=False, validate_parameters=None, verbosity=0)
```

[40]:
```
y_train
```

[40]:
```
209    1
32     1
346    0
60     1
426    1
      ..
302    0
284    1
149    1
188    0
75     1
Name: political_party, Length: 347, dtype: int64
```

6 Wybór modelu

[41]:
```
df=pd.read_csv("df_encoded.csv")
```

Chociaż na we wcześniejszym etapie pracy usunęliśmy z naszych danych zduplikowane rzędy, to podczas pracy z modelami okazuje się, że osiągają one lepsze wyniki kiedy je zostawimy.

[42]:
```
from sklearn.model_selection import train_test_split
X=df.drop(["political_party"], axis=1)
y=df["political_party"]
X_train, X_test, y_train, y_test = train_test_split(X, y,␣
 ↪stratify=y,random_state = 42)
```

Wybraliśmy do przetestowania 3 modele - **SVM**, **Random Forest** i **XGBoost**.

6.1 SVM

[43]:
```
svm_base=SVC(random_state=42)
svm_base.fit(X_train, y_train)
preds=svm_base.predict(X_test)
```

[44]:
```
svm_base_acc=accuracy_score(preds,y_test)
print("Accuracy SVM z domyślnymi hiperparametrami: " + str(svm_base_acc))
```

```
Accuracy SVM z domyślnymi hiperparametrami: 0.963302752293578
```

Widzimy, że już domyślny SVM osiąga bardzo dobre accuracy na poziomie ponad 96%. Spróbujemy teraz wykonać tuning hiperparametrów. Ponieważ nasz zbiór danych jest stosunkowo mały, skorzystamy z narzędzia GridSearch.

[45]:
```
svm_tuned=SVC(random_state=42)
c=[]  # wartości parametru C
gamma=[]  #wartości parametru gamma
for i in range(-4, 5):      # orientacyjne wartości na podstawie informacji␣
 ↪znalezionych w internecie
    c.append(10**i)
for i in range(-4, 5):
    gamma.append(10**i)
gamma.append("auto")
gamma.append("scale")
params = [{'C': c,
           "kernel": ["rbf", "linear", "poly"],
        'gamma': gamma}]
gs_svm=GridSearchCV(svm_tuned, param_grid=params, scoring='accuracy', cv=4,␣
 ↪n_jobs=2)
gs_svm.fit(X_train, y_train)
gs_svm.best_params_
```

[45]:
```
{'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
```

[46]:
```
svm_tuned_acc=accuracy_score(gs_svm.predict(X_test),y_test)
print("Accuracy SVM po tuningu hiperparametrów: " + str(svm_tuned_acc))
```

```
Accuracy SVM po tuningu hiperparametrów: 0.9541284403669725
```

Jak widzimy, nie udało nam się polepszyć wyniku, a nawet uzyskaliśmy accuracy trochę gorsze. Mimo że w naszej ramce danych znajdują się także domyślne wartości hiperparametrów, to wypadły one gorzej przy kroswalidacji i dlatego algorytm ich nie wybrał. Wydaje mi się, że taka sytuacja zachodzi ze względu na małą liczbę rekordów i duże accuracy naszych modeli.

6.2 XGBoost

[47]:
```
xgb_base = xgb.XGBClassifier(objective = "binary:logistic", seed = 1613,␣
 ↪use_label_encoder=False, verbosity=0)
xgb_base.fit(X_train, y_train)
preds=xgb_base.predict(X_test)
```

[48]:
```
xgb_base_acc=accuracy_score(preds,y_test)
print("Accuracy XGB z domyślnymi hiperparametrami: " + str(xgb_base_acc))
```

```
Accuracy XGB z domyślnymi hiperparametrami: 0.9724770642201835
```

[49]:
```
xgb_tuned=xgb.XGBClassifier(objective = "binary:logistic", seed = 1613,␣
 ↪use_label_encoder=False, eval_metric="error")
params = {
        'min_child_weight': [1, 5, 10],
        'gamma': [0.5, 1, 1.5, 2, 5],
        'subsample': [0.6, 0.8, 1.0],
```

```
        'colsample_bytree': [0.6, 0.8, 1.0],
        'max_depth': [3, 4, 5]
        }

gs_xgb=GridSearchCV(xgb_tuned, param_grid=params, scoring='accuracy', cv=4,␣
 ↪n_jobs=2)
gs_xgb.fit(X_train, y_train)
gs_xgb.best_params_
```

[49]:
```
{'colsample_bytree': 1.0,
 'gamma': 1,
 'max_depth': 3,
 'min_child_weight': 1,
 'subsample': 0.6}
```

[50]:
```
xgb_tuned_acc=accuracy_score(gs_xgb.predict(X_test), y_test)
print("Accuracy XGB po treningu hiperparametrów: " + str(xgb_tuned_acc))
```

```
Accuracy XGB po treningu hiperparametrów: 0.9724770642201835
```

6.3 Random Forest

[51]:
```
rfc_base = RandomForestClassifier(random_state=16)
rfc_base.fit(X_train, y_train)
preds=rfc_base.predict(X_test)
```

[52]:
```
rfc_base_acc=accuracy_score(preds,y_test)
print("Accuracy RFC z domyślnymi hiperparametrami: " + str(rfc_base_acc))
```

```
Accuracy RFC z domyślnymi hiperparametrami: 0.9724770642201835
```

[53]:
```
rfc_tuned=RandomForestClassifier(random_state=16)
n_estimators = [int(x) for x in np.linspace(start = 50, stop = 1000, num = 5)]␣
 ↪# przykładowe wartości znalezione w internecie
max_depth = [int(x) for x in np.linspace(5, 55, num = 5)]
max_features= ['auto', 'sqrt', 'log2']

params = [{'n_estimators': n_estimators,
        'max_depth': max_depth,
          'max_features': max_features}]
gs_rfc=GridSearchCV(rfc_tuned, param_grid=params, scoring='accuracy', cv=4,␣
 ↪n_jobs=2)
gs_rfc.fit(X_train, y_train)
gs_rfc.best_params_
```

[53]:
```
{'max_depth': 5, 'max_features': 'auto', 'n_estimators': 50}
```

[54]:
```
rfc_tuned_acc=accuracy_score(gs_rfc.predict(X_test), y_test)
print("Accuracy RFC po treningu hiperparametrów: " + str(rfc_tuned_acc))
```

```
Accuracy RFC po treningu hiperparametrów: 0.963302752293578
```

Jak widać, ani w XGBooście, ani w Random Forest nie udało się uzyskać lepszej niż domyślna accuracy. Co więcej, obydwa bazowe modele osiągają dokładnie ten sam wynik.

7 Ocena modeli

7.1 Accuracy score

[55]:
```
scores=[]
labels=[]
scores.append(svm_base_acc)
labels.append("SVM")
scores.append(xgb_base_acc)
labels.append("XGB")
scores.append(rfc_base_acc)
labels.append("RFC")
```

[56]:
```
pd.DataFrame({"Accuracy Score": scores}, index=labels)
```

[56]:

	Accuracy Score
SVM	0.963303
XGB	0.972477
RFC	0.972477

7.2 Confusion matrix

7.2.1 SVM

[58]:
```
tn, fp, fn, tp = confusion_matrix(y_test, svm_base.predict(X_test)).ravel()
pd.DataFrame({"Actual positives": [tp, fp], "Actual negatives": [fn, tn]},␣
↪index = ["Positive predictions", "Negative predictions"])
```

[58]:

	Actual positives	Actual negatives
Positive predictions	64	3
Negative predictions	1	41

7.2.2 XGB

[59]:
```
tn, fp, fn, tp = confusion_matrix(y_test, xgb_base.predict(X_test)).ravel()
pd.DataFrame({"Actual positives": [tp, fp], "Actual negatives": [fn, tn]},␣
↪index = ["Positive predictions", "Negative predictions"])
```

[59]:

	Actual positives	Actual negatives
Positive predictions	65	2
Negative predictions	1	41

7.2.3 RFC

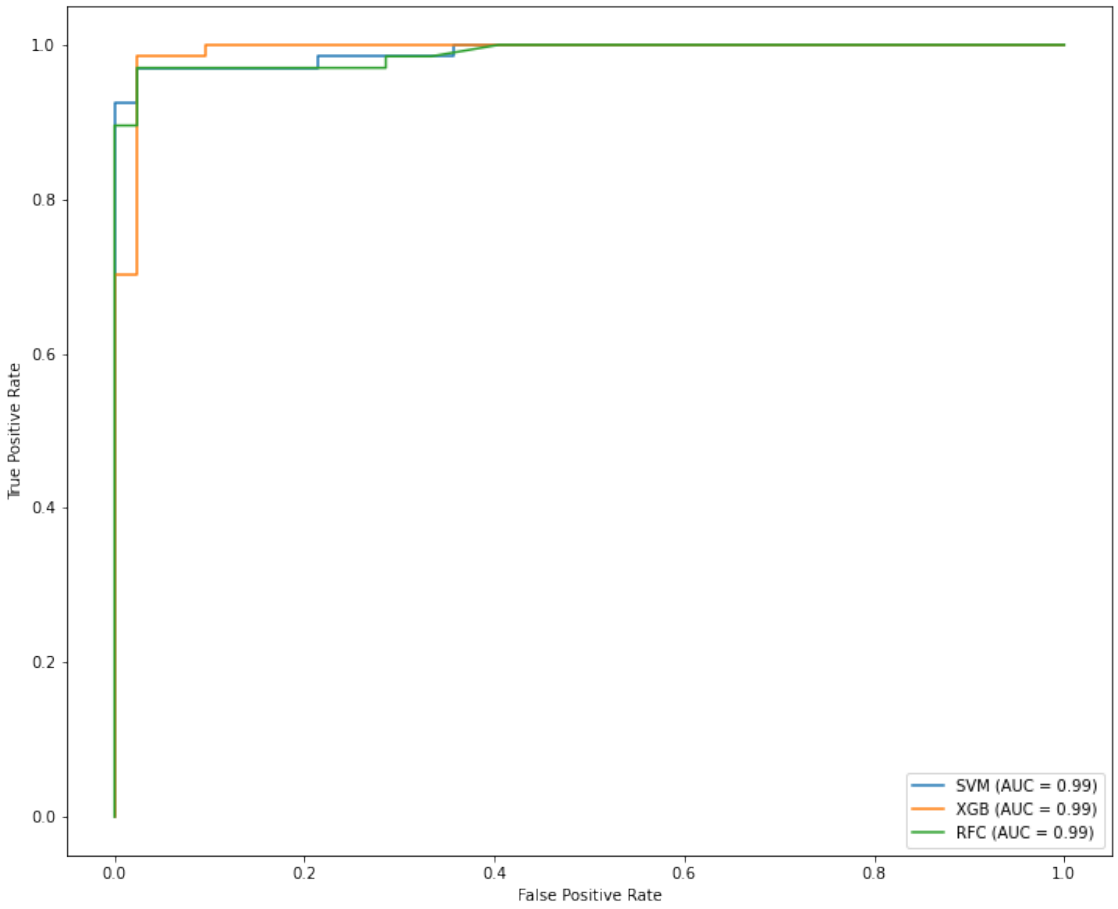
[60]:
```
tn, fp, fn, tp = confusion_matrix(y_test, xgb_base.predict(X_test)).ravel()
pd.DataFrame({"Actual positives": [tp, fp], "Actual negatives": [fn, tn]},␣
↪index = ["Positive predictions", "Negative predictions"])
```

[60]:

	Actual positives	Actual negatives
Positive predictions	65	2
Negative predictions	1	41

7.3 ROC AUC

[61]:
```
gs_svm
plt.figure(figsize=(12,10))
classifiers = [svm_base, xgb_base, rfc_base]
labels=["SVM", "XGB", "RFC"]
ax = plt.gca()
for i in range(3):
    metrics.plot_roc_curve(classifiers[i], X_test, y_test, ax=ax,␣
↪name=labels[i])
```



Na podstawie powyższych wyników zdecydowaliśmy się pozostać przy modelu **XGBoost** bez modyfikacji hiperparametrów czy wyboru zmiennych. W naszym przypadku nawet taki model osiagał bardzo wysokie accuracy, a był przy tym szybki.

8 Podsumowanie

8.1 EDA throwback - które predykcje się udały

[62]:
```
df_tmp=pd.read_csv("congressional_voting_dataset.csv")
df_tmp=df_tmp.replace("n", 0)
df_tmp=df_tmp.replace("y", 1)
df_tmp=df_tmp.replace("?",  0.5)
 ↪  #żeby uzyskać taki sam wykres jak w eda
df_tmp["political_party"]=df_tmp["political_party"].replace("republican", 0)
 ↪  #musimy użyć ramki danych z rzędem,
df_tmp["political_party"]=df_tmp["political_party"].replace("democrat", 1)
 ↪  # który wcześniej wyrzuciliśmy
```

```
y_tmp=df_tmp["political_party"]
X_tmp=df_tmp.drop(["political_party"], axis=1)


xgb_tmp=xgb.XGBClassifier(objective = "binary:logistic", seed = 1613,␣
↪use_label_encoder=False, verbosity=0)
df_tmp["predicted_correctly"]=False
for i in range(len(df_tmp)):
    X_train_tmp=X_tmp.drop(i, axis=0)
    y_train_tmp=y_tmp.drop(i, axis=0)
    X_test_tmp=X_tmp.iloc[[i]]
    y_test_tmp=y_tmp[i]
    xgb_tmp.fit(X_train_tmp, y_train_tmp)
    pred_tmp=xgb_tmp.predict(X_test_tmp)[0]
    if y_tmp[i]==pred_tmp:
        df_tmp["predicted_correctly"][i]=True
```

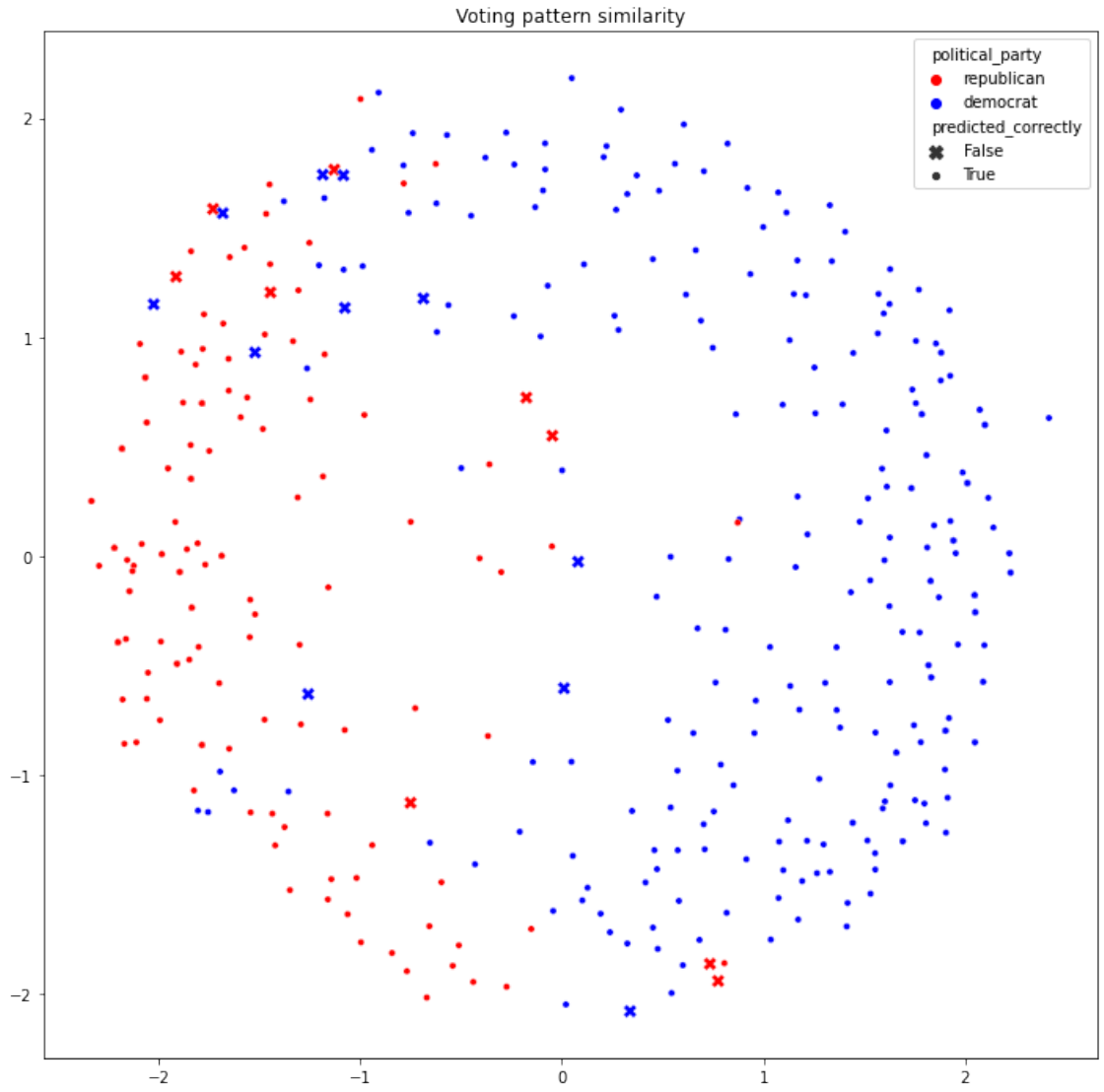
[63]:
```
adist=sklearn.metrics.pairwise_distances(df_tmp.drop(["political_party",␣
↪"predicted_correctly"], axis=1))
adist
```

[63]:
```
array([[0.        , 1.22474487, 2.17944947, …, 1.22474487, 1.5       ,
        1.22474487],
       [1.22474487, 0.        , 1.93649167, …, 1.22474487, 1.5       ,
        1.22474487],
       [2.17944947, 1.93649167, 0.        , …, 1.93649167, 2.54950976,
        2.17944947],
       …,
       [1.22474487, 1.22474487, 1.93649167, …, 0.        , 1.5       ,
        1.87082869],
       [1.5       , 1.5       , 2.54950976, …, 1.5       , 0.        ,
        1.80277564],
       [1.22474487, 1.22474487, 2.17944947, …, 1.87082869, 1.80277564,
        0.        ]])
```

[64]:
```
df_tmp["political_party"]=df_tmp["political_party"].replace(0, "republican")
df_tmp["political_party"]=df_tmp["political_party"].replace(1, "democrat")
adist=np.array(adist)
mds = manifold.MDS(n_components=2, dissimilarity="precomputed", random_state=6)
results = mds.fit(adist)
coords = results.embedding_
fig, ax = plt.subplots(figsize=(12,12))
sns.scatterplot(
    coords[:, 0], coords[:, 1], marker = 'o', hue=df_tmp["political_party"],␣
↪palette=["red", "blue"],
```

```
    style=df_tmp["predicted_correctly"], markers=["X", "o"],␣
↪size=df_tmp["predicted_correctly"]
    )
ax.set_title("Voting pattern similarity")
```

[64]: Text(0.5, 1.0, 'Voting pattern similarity')



Na koniec wróciliśmy do wykresu z naszego EDA żeby zobaczyć, których z reprezentantów udało przewidzieć się prawidłowo – wykonaliśmy w tym celu predykcje na podstawie reszty ramki danych dla każdego pojedynczego rzędu. Jak widać, błędne predykcje zazwyczaj znajdują się w zróżnicowanym pod względem partii otoczeniu. Jednak duża skuteczność nawet dla nieoczywistych polityków pokazuje świadczy o jakości XGBoosta, nawet bez specjalnego dopasowywania cech czy tuningu hiperparametrów.