Projekt PJN

May 23, 2022

1 Temat projektu:

1.1 Optymalizacja hiperparametryczna klasyfikatorów na przykładzie zagadnienia rozpoznawania spamu

Celem projektu jest przeprowadzenie uczenia modeli klasyfikacji przy pomocy kilku różnych metod. Metody uwzględnione w ramach projektu to: - Bag of words - wektory tf-idf - PCA - TSVD - Word2vec - LDiA

Dla wszystkich tych metod zdecydowano się na trening następujących klasyfikatorów: - Regresja logistyczna - Las losowy - Naiwny klasyfikator Bayesa - Drzewo decyzyjne - Maszyna wektorów nośnych - K najbliższych sąsiadów - Klasyfikator Ada Boost - LDA

Dodatkowo po treningu i ewaluacji wszystkich modeli wybrane zostaną 4 z najwyższym wynikiem dokładności klasyfikacji oraz zostanie dla nich przeprowadzona optymalizacja hiperparametryczna.

Po przeprowadzeniu wszystkich tych kroków wyznaczony będzie najlepszy model rozpoznawania spamu.

2 Klasyfikacja komentarzy z youtube jako spam

Dataset

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

[2]:
```
df = pd.read_csv('Dataset/data.csv')
```

[3]:
```
df = df.drop(['comment_id', 'author', 'date'], axis=1)
```

[4]:
```
df = df.dropna()
df = df[df['class'] != 'class']
```

[5]:
```
df
```

[5]:

	content	class
0	i love this so much. AND also I Generate Free …	true
1	http://www.billboard.com/articles/columns/pop-…	true
2	Hey guys! Please join me in my fight to help a…	true
3	http://psnboss.com/?ref=2tGgp3pV6L this is the…	true

```
4     Hey everyone. Watch this trailer!!!!!!!!  http…   true
…                                                  …     …
1165  I love this song because we sing it at Camp al…  false
1166  I love this song for two reasons: 1.it is abou…  false
1167                                               wow  false
1168                            Shakira u are so wiredo  false
1169                          Shakira is the best dancer  false

[1168 rows x 2 columns]
```

[6]:
```
np.unique(df['class'], return_counts=True)
```

[6]:
```
(array(['false', 'true'], dtype=object), array([574, 594], dtype=int64))
```

[7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1168 entries, 0 to 1169
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   content  1168 non-null   object
 1   class    1168 non-null   object
dtypes: object(2)
memory usage: 27.4+ KB
```

2.0.1 Przygotowanie struktur wejściowych

Bag of words

[8]:
```
from sklearn.feature_extraction.text import CountVectorizer
from nltk.tokenize.casual import casual_tokenize
from nltk.stem import PorterStemmer

ps = PorterStemmer()

def tokenizing_and_steamming(text):
    tokens = casual_tokenize(text)
    return [ps.stem(token) for token in tokens]

counter = CountVectorizer(tokenizer=tokenizing_and_steamming)
bag = pd.DataFrame(counter.fit_transform(raw_documents=df['content']).toarray())
column_nums, terms = zip(*sorted(zip(counter.vocabulary_.values(), counter.
 ↪vocabulary_.keys())))
bag.columns = terms
bag
```

[8]:
```
      !  "  #  #2015  #activ  #awesom  #eminem  #katycat  #king  \
0     1  0  0      0       0        0        0         0      0
1     0  0  0      0       0        0        0         0      0
2     3  0  0      0       0        0        0         0      0
3     0  0  0      0       0        0        0         0      0
4     3  0  0      0       0        0        0         0      0
…   .. .. ..    …       …        …        …         …      …
1163  2  0  0      0       0        0        0         0      0
1164  0  0  0      0       0        0        0         0      0
1165  0  0  0      0       0        0        0         0      0
1166  0  0  0      0       0        0        0         0      0
1167  0  0  0      0       0        0        0         0      0

      #lovethewayyouli  …
0                    0  …  0  0  0  0  0  0  0  0  0  0
1                    0  …  0  0  0  0  0  0  0  0  0  0
2                    0  …  0  0  0  0  0  0  0  0  0  0
3                    0  …  0  0  0  0  0  0  0  0  0  0
4                    0  …  0  0  0  0  0  0  0  0  0  0
…                  …  …  .. .. .. .. .. .. .. .. .. ..
1163                 0  …  0  0  0  0  0  0  0  0  0  0
1164                 0  …  0  0  0  0  0  0  0  0  0  0
1165                 0  …  0  0  0  0  0  0  0  0  0  0
1166                 0  …  0  0  0  0  0  0  0  0  0  0
1167                 0  …  0  0  0  0  0  0  0  0  0  0

[1168 rows x 2952 columns]
```

TFIDF, PCA, Truncated SVD

[9]:
```
#2. make TF-IDF vector
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(tokenizer=tokenizing_and_steamming)
tfidf_docs = tfidf.fit_transform(df['content']).toarray()
print("TFIDF docs\n", tfidf_docs)

#3. dimmensional reduction with PCA

from sklearn.decomposition import PCA
pca = PCA(n_components=16)
pca = pca.fit(tfidf_docs)
pca_topic_vectors = pca.transform(tfidf_docs)
columns = ['topic{}'.format(i) for i in range(pca.n_components)]
pca_topic_vectors = pd.DataFrame(pca_topic_vectors, columns=columns)
print("\nPCA topics\n",pca_topic_vectors.round(3).head(6))
print('explained variation', pca.explained_variance_.sum())
```

```python
#4. dimmensional reduction with truncated svd
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=16)
svd.fit(tfidf_docs)
svd_topic_vectors = svd.transform(tfidf_docs)
print("\nTruncated SVD topics\n", svd_topic_vectors)
print('explained variation', svd.explained_variance_ratio_.sum())
```

```
TFIDF docs
 [[0.10863708 0.         0.         … 0.         0.         0.        ]
 [0.         0.         0.         … 0.         0.         0.        ]
 [0.15390298 0.         0.         … 0.         0.         0.        ]
 …
 [0.         0.         0.         … 0.         0.         0.        ]
 [0.         0.         0.         … 0.         0.         0.        ]
 [0.         0.         0.         … 0.         0.         0.        ]]

PCA topics
    topic0  topic1  topic2  topic3  topic4  topic5  topic6  topic7  topic8  \
0  -0.055   0.014   0.054   0.167  -0.033   0.029  -0.098  -0.011   0.032
1  -0.015  -0.041  -0.116  -0.086   0.031   0.146   0.032   0.041   0.029
2  -0.074   0.165  -0.069  -0.002   0.026   0.075   0.044  -0.040   0.002
3   0.000  -0.184   0.087  -0.015  -0.202  -0.109   0.188  -0.003  -0.012
4  -0.130   0.286   0.247  -0.114  -0.053   0.021   0.021  -0.033   0.031
5   0.057   0.065  -0.114   0.059   0.149  -0.041   0.012   0.037   0.136

   topic9  topic10  topic11  topic12  topic13  topic14  topic15
0  -0.093    0.002   -0.057   -0.002   -0.038   -0.007   -0.020
1  -0.029   -0.007   -0.032   -0.005    0.089   -0.002   -0.011
2  -0.019   -0.023   -0.094   -0.003    0.015    0.000   -0.007
3   0.101    0.147    0.047    0.064   -0.082   -0.050   -0.090
4  -0.125   -0.002   -0.057   -0.059    0.051   -0.015   -0.018
5  -0.033   -0.034    0.155    0.007   -0.009    0.046    0.070
explained variation 0.21478910946720053

Truncated SVD topics
 [[ 0.25931405 -0.06066595  0.05581693 … -0.02381328 -0.01410658
   0.00087317]
 [ 0.12932393 -0.0126793  -0.0394179  …  0.05344674  0.00716585
   0.02599439]
 [ 0.25032626 -0.08006427 -0.1615759  …  0.00217587  0.00961804
   0.00987274]
 …
 [ 0.0055361  -0.00262596  0.0015347  … -0.03133198 -0.01189845
   0.01350617]
```

```
 [ 0.05869936 -0.02649554  0.01569798 … -0.10664426 -0.01380359
  -0.11917612]
 [ 0.1064023  -0.05053886  0.05306774 … -0.1738374  -0.01022039
  -0.01555486]]
explained variation 0.22169333740180816
```

Word2Vec

[10]:
```
from gensim.models.keyedvectors import KeyedVectors
word_vectors = KeyedVectors.load_word2vec_format('./
 ↪GoogleNews-vectors-negative300.bin.gz', binary=True, limit=200000)
```

[11]:
```
# converting comments to word2vec vectors average
w2vec_x = []
w2vec_y = []

for i, row in df.iterrows():
    tokens = row['content'].split(' ') # not stemming gives better results,␣
 ↪explained later
    vec = sum(word_vectors[word] for word in tokens if word in word_vectors)
    if type(vec) is np.ndarray and vec.size > 0:
        w2vec_x.append(vec/vec.size)
        w2vec_y.append(row['class'])
w2vec_x, w2vec_y = np.array(w2vec_x), np.array(w2vec_y)
w2vec_x.shape, w2vec_y.shape
```

[11]:
```
((1076, 300), (1076,))
```

LDiA

[12]:
```
from sklearn.decomposition import LatentDirichletAllocation

ldia = LatentDirichletAllocation(n_components=16)

ldia = ldia.fit(bag)
ldia_docs = ldia.transform(bag)
pd.DataFrame(ldia.components_.T, index=terms)
```

[12]:

	0	1	2	3	4	5	\
!	21.365132	1.062527	1.348453	514.818675	0.062501	0.062500	
"	0.062500	0.062500	4.419767	0.062500	0.101690	26.135634	
#	0.062500	0.062500	0.062500	0.062500	0.062500	0.062500	
#2015	0.062500	0.062500	0.062500	0.062500	0.062500	0.062500	
#activ	0.062500	0.062500	0.062500	0.062500	0.062500	0.062500	
…	…	…	…	…	…	…	
	1.062500	0.062500	0.062500	0.062500	1.062500	0.062500	
	1.062500	0.062500	0.062500	0.062500	0.062500	0.062500	
	0.062500	0.062500	0.062500	0.062500	0.062500	1.062500	

```
          0.062500  0.062500  0.062500    2.062500  0.062500   0.062500
          0.062500  0.062500  0.062500    0.062500  0.062500   0.062500

                6         7          8         9           10        11  \
!        0.062562  0.062500   0.649120  0.062500  100.656752  0.062500
"        0.062500 13.937071 134.279824  4.284401    0.062500  0.062500
#        0.062501  0.062500   0.062500  0.062500    0.062500  0.062501
#2015    0.062500  0.062500   1.062500  0.062500    0.062500  0.062500
#activ   0.062500  1.062500   0.062500  0.062500    0.062500  0.062500
…              …         …          …         …           …         …
         0.062500  0.062500   0.062500  0.062500    0.062500  0.062500
         0.062500  0.062500   0.062500  0.062500    0.062500  0.062500
         0.062500  0.062500   0.062500  0.062500    0.062500  0.062500
         0.062500  0.062500   0.062500  0.062500    0.062500  0.062500
         0.062500  0.062500   0.062500  1.062500    0.062500  0.062500

                 12        13       14      15
!        144.189397  0.062500  0.40988  0.0625
"         25.279112  0.062500  0.06250  0.0625
#          0.062500  1.062498  0.06250  0.0625
#2015      0.062500  0.062500  0.06250  0.0625
#activ     0.062500  0.062500  0.06250  0.0625
…                 …         …        …       …
           0.062500  0.062500  0.06250  0.0625
           0.062500  0.062500  0.06250  0.0625
           0.062500  0.062500  0.06250  0.0625
           0.062500  0.062500  0.06250  0.0625
           0.062500  0.062500  0.06250  0.0625

[2952 rows x 16 columns]
```

2.0.2 Klasyfikatory

[13]:
```
def evaluate_classifier(clf, X_train, Y_train, cv=10):
    scores = cross_val_score(clf, X_train, Y_train, cv=cv)
    print(clf.__class__.__name__, " = ","Accuracy: {:.3f} (+/-{:.3f})".
 ↪format(scores.mean(), scores.std() * 2))
    return scores.mean(), scores.std()
```

[14]:
```
from sklearn.model_selection import cross_val_score
import copy

from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

cv = 5
scores = {}

clfs = [
LogisticRegression(),
RandomForestClassifier(),
GaussianNB(),
DecisionTreeClassifier(),
SVC(),
KNeighborsClassifier(),
AdaBoostClassifier(),
LDA()
]

clfs_tfidf = copy.deepcopy(clfs)
clfs_pca = copy.deepcopy(clfs)
clfs_tsvd = copy.deepcopy(clfs)
clfs_ldia = copy.deepcopy(clfs)
clfs_w2vec = copy.deepcopy(clfs)

print("Bag of words")
for clf in clfs:
    name = clf.__class__.__name__
    accuracy, std = evaluate_classifier(clf, bag, df['class'], cv)
    scores[name] = {'bow_accuracy': accuracy, 'bow_std': std}

print("\n\nTFIDF")
for clf in clfs_tfidf:
    name = clf.__class__.__name__
    accuracy, std = evaluate_classifier(clf, tfidf_docs, df['class'], cv)
    scores[name]['tfidf_accuracy'] = accuracy
    scores[name]['tfidf_std'] = std

print("\n\nPCA")
for clf in clfs_pca:
    name = clf.__class__.__name__
    accuracy, std = evaluate_classifier(clf, pca_topic_vectors, df['class'], cv)
    scores[name]['pca_accuracy'] = accuracy
    scores[name]['pca_std'] = std

print("\n\nTSVD")
for clf in clfs_pca:
    name = clf.__class__.__name__
```

```
    accuracy, std = evaluate_classifier(clf, svd_topic_vectors, df['class'], cv)
    scores[name]['tsvd_accuracy'] = accuracy
    scores[name]['tsvd_std'] = std


print("\n\nLDiA")
for clf in clfs_ldia:
    name = clf.__class__.__name__
    accuracy, std = evaluate_classifier(clf, ldia_docs, df['class'], cv)
    scores[name]['w2v_accuracy'] = accuracy
    scores[name]['w2v_std'] = std

print("\n\nWord2vec")
for clf in clfs_w2vec:
    name = clf.__class__.__name__
    accuracy, std = evaluate_classifier(clf, w2vec_x, w2vec_y, cv)
    scores[name]['w2v_accuracy'] = accuracy
    scores[name]['w2v_std'] = std
```

```
Bag of words
LogisticRegression  =  Accuracy: 0.861 (+/-0.203)
RandomForestClassifier  =  Accuracy: 0.899 (+/-0.141)
GaussianNB  =  Accuracy: 0.754 (+/-0.094)
DecisionTreeClassifier  =  Accuracy: 0.863 (+/-0.208)
SVC  =  Accuracy: 0.846 (+/-0.211)
KNeighborsClassifier  =  Accuracy: 0.783 (+/-0.147)
AdaBoostClassifier  =  Accuracy: 0.844 (+/-0.209)
LinearDiscriminantAnalysis  =  Accuracy: 0.791 (+/-0.179)


TFIDF
LogisticRegression  =  Accuracy: 0.892 (+/-0.118)
RandomForestClassifier  =  Accuracy: 0.882 (+/-0.145)
GaussianNB  =  Accuracy: 0.756 (+/-0.099)
DecisionTreeClassifier  =  Accuracy: 0.864 (+/-0.179)
SVC  =  Accuracy: 0.902 (+/-0.124)
KNeighborsClassifier  =  Accuracy: 0.855 (+/-0.124)
AdaBoostClassifier  =  Accuracy: 0.871 (+/-0.141)
LinearDiscriminantAnalysis  =  Accuracy: 0.789 (+/-0.122)


PCA
LogisticRegression  =  Accuracy: 0.880 (+/-0.172)
RandomForestClassifier  =  Accuracy: 0.871 (+/-0.141)
GaussianNB  =  Accuracy: 0.765 (+/-0.115)
DecisionTreeClassifier  =  Accuracy: 0.814 (+/-0.091)
SVC  =  Accuracy: 0.869 (+/-0.155)
```

```
KNeighborsClassifier  =  Accuracy: 0.846 (+/-0.092)
AdaBoostClassifier  =  Accuracy: 0.860 (+/-0.119)
LinearDiscriminantAnalysis  =  Accuracy: 0.879 (+/-0.188)


TSVD
LogisticRegression  =  Accuracy: 0.885 (+/-0.158)
RandomForestClassifier  =  Accuracy: 0.885 (+/-0.094)
GaussianNB  =  Accuracy: 0.806 (+/-0.100)
DecisionTreeClassifier  =  Accuracy: 0.829 (+/-0.100)
SVC  =  Accuracy: 0.897 (+/-0.104)
KNeighborsClassifier  =  Accuracy: 0.852 (+/-0.096)
AdaBoostClassifier  =  Accuracy: 0.866 (+/-0.100)
LinearDiscriminantAnalysis  =  Accuracy: 0.879 (+/-0.158)


LDiA
LogisticRegression  =  Accuracy: 0.775 (+/-0.121)
RandomForestClassifier  =  Accuracy: 0.773 (+/-0.104)
GaussianNB  =  Accuracy: 0.773 (+/-0.132)
DecisionTreeClassifier  =  Accuracy: 0.748 (+/-0.109)
SVC  =  Accuracy: 0.781 (+/-0.125)
KNeighborsClassifier  =  Accuracy: 0.745 (+/-0.099)
AdaBoostClassifier  =  Accuracy: 0.764 (+/-0.134)
LinearDiscriminantAnalysis  =  Accuracy: 0.775 (+/-0.130)


Word2vec
LogisticRegression  =  Accuracy: 0.686 (+/-0.149)
RandomForestClassifier  =  Accuracy: 0.898 (+/-0.105)
GaussianNB  =  Accuracy: 0.672 (+/-0.118)
DecisionTreeClassifier  =  Accuracy: 0.812 (+/-0.059)
SVC  =  Accuracy: 0.909 (+/-0.075)
KNeighborsClassifier  =  Accuracy: 0.876 (+/-0.120)
AdaBoostClassifier  =  Accuracy: 0.881 (+/-0.083)
LinearDiscriminantAnalysis  =  Accuracy: 0.813 (+/-0.116)
```

[13]:
```
import pandas as pd
scores
pd.DataFrame(scores).T
# important note -> word2vec without stemming gives better results, probably␣
  ↪because w2vec has whole words(not stemmed)
```

[13]:

	bow_accuracy	bow_std	tfidf_accuracy	tfidf_std	\
LogisticRegression	0.861271	0.101632	0.892168	0.059142	
RandomForestClassifier	0.886145	0.073337	0.891281	0.071715	
GaussianNB	0.754371	0.047195	0.756087	0.049635	

```
DecisionTreeClassifier          0.862987  0.104332        0.862111   0.089089
SVC                             0.845890  0.105454        0.901592   0.062121
KNeighborsClassifier            0.782546  0.073716        0.855398   0.062176
AdaBoostClassifier              0.844151  0.104582        0.870705   0.070293
LinearDiscriminantAnalysis      0.791185  0.089298        0.789443   0.061070

                          pca_accuracy   pca_std  tsvd_accuracy  tsvd_std  \
LogisticRegression            0.878530  0.086825       0.882792  0.078294
RandomForestClassifier        0.869062  0.074911       0.894725  0.044403
GaussianNB                    0.761186  0.054407       0.790305  0.052754
DecisionTreeClassifier        0.803096  0.039706       0.842526  0.047544
SVC                           0.870757  0.081122       0.894699  0.053230
KNeighborsClassifier          0.845926  0.046633       0.850211  0.048780
AdaBoostClassifier            0.851928  0.056862       0.878464  0.042963
LinearDiscriminantAnalysis    0.878522  0.090133       0.875925  0.073519

                          w2v_accuracy   w2v_std
LogisticRegression            0.800558  0.043296
RandomForestClassifier        0.804835  0.070020
GaussianNB                    0.767177  0.069859
DecisionTreeClassifier        0.777492  0.070474
SVC                           0.800572  0.062810
KNeighborsClassifier          0.755152  0.055198
AdaBoostClassifier            0.803973  0.075020
LinearDiscriminantAnalysis    0.791130  0.047888
```

Z powyższego zestawienia wyników klasyfikacji można wyróżnić 3 najlepsze modele:

1. w2v SVC, accuracy: 0.908958
2. tf-idf SVC, accuracy: 0.901592
3. tsvd SVC, sccuracy: 0.894696

Modele te zostaną poddane optymalizacji hiperparametrów algorytmem grid search aby zmaksymalizować dokładność modelu. Dodatkowo z racji że wszystkie 3 najlepsze uzyskane modele opierają się o maszynę wektorów nośnych dodatkowo przeprowadzimy również optymalizację dla modelu z klasyfikatorem regresji logistycznej na wektorach tf-idf (accuracy: 0.892168)

[14]:
```
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
```

2.1 Optymalizacja hiperparametrów klasyfikatorów algorytmem grid search

2.1.1 Model 1 - SVC, w2v

[17]:
```
parameters = {
    'kernel':['linear', 'sigmoid', 'poly', 'rbf'],
              'C':[C for C in np.arange(0.01, 11.01, 1.0)],
              'gamma': ['scale', 'auto'],
              'decision_function_shape': ['ovo', 'ovr'],
              'shrinking': [True, False]
             }

svcGridSearch = SVC()

clfSVC = GridSearchCV(svcGridSearch, parameters, n_jobs=-1, cv=5, verbose=10)
clfSVC.fit(w2vec_x, w2vec_y)
```

```
Fitting 5 folds for each of 352 candidates, totalling 1760 fits
```

[17]:
```
GridSearchCV(cv=5, estimator=SVC(), n_jobs=-1,
             param_grid={'C': [0.01, 1.01, 2.01, 3.01, 4.01, 5.01, 6.01, 7.01,
                               8.01, 9.01, 10.01],
                         'decision_function_shape': ['ovo', 'ovr'],
                         'gamma': ['scale', 'auto'],
                         'kernel': ['linear', 'sigmoid', 'poly', 'rbf'],
                         'shrinking': [True, False]},
             verbose=10)
```

[18]:
```
clfSVC.best_estimator_
```

[18]:
```
SVC(C=9.01, decision_function_shape='ovo')
```

[19]:
```
clfSVC.best_score_
```

[19]:
```
0.9126873385012919
```

2.1.2 Model 2 - SVC, wektory tf-idf

[15]:
```
parameters = {
    'kernel':['linear', 'sigmoid', 'poly', 'rbf'],
              'C':[C for C in np.arange(0.01, 11.01, 1.0)],
              'gamma': ['scale', 'auto'],
              'decision_function_shape': ['ovo', 'ovr'],
              'shrinking': [True, False]
             }

svcGridSearch = SVC()
```

```
clfSVC = GridSearchCV(svcGridSearch, parameters, n_jobs=20, cv=5, verbose=10)
clfSVC.fit(tfidf_docs, df['class'])
```

```
Fitting 5 folds for each of 352 candidates, totalling 1760 fits
```

[15]:
```
GridSearchCV(cv=5, estimator=SVC(), n_jobs=20,
             param_grid={'C': [0.01, 1.01, 2.01, 3.01, 4.01, 5.01, 6.01, 7.01,
                               8.01, 9.01, 10.01],
                         'decision_function_shape': ['ovo', 'ovr'],
                         'gamma': ['scale', 'auto'],
                         'kernel': ['linear', 'sigmoid', 'poly', 'rbf'],
                         'shrinking': [True, False]},
             verbose=10)
```

[16]:
```
clfSVC.best_estimator_
```

[16]:
```
SVC(C=5.01, decision_function_shape='ovo')
```

[17]:
```
clfSVC.best_score_
```

[17]:
```
0.9075785921279482
```

2.1.3 Model 3 - SVC, tsvd

[19]:
```
parameters = {
    'kernel':['linear', 'sigmoid', 'poly', 'rbf'],
              'C':[C for C in np.arange(0.01, 11.01, 1.0)],
              'gamma': ['scale', 'auto'],
              'decision_function_shape': ['ovo', 'ovr'],
              'shrinking': [True, False]
             }

svcGridSearch = SVC()

clfSVC = GridSearchCV(svcGridSearch, parameters, n_jobs=20, cv=5, verbose=10)
clfSVC.fit(svd_topic_vectors, df['class'])
```

```
Fitting 5 folds for each of 352 candidates, totalling 1760 fits
```

[19]:
```
GridSearchCV(cv=5, estimator=SVC(), n_jobs=20,
             param_grid={'C': [0.01, 1.01, 2.01, 3.01, 4.01, 5.01, 6.01, 7.01,
                               8.01, 9.01, 10.01],
                         'decision_function_shape': ['ovo', 'ovr'],
                         'gamma': ['scale', 'auto'],
                         'kernel': ['linear', 'sigmoid', 'poly', 'rbf'],
                         'shrinking': [True, False]},
```

```
                verbose=10)
```

[20]:
```
clfSVC.best_estimator_
```

[20]:
```
SVC(C=4.01, decision_function_shape='ovo')
```

[21]:
```
clfSVC.best_score_
```

[21]:
```
0.8981145225780418
```

2.1.4 Model 4 - Logistic regression, wektory tf-idf

[26]:
```
parameters = {'C':[C for C in np.arange(0.01, 26.01, 1.0)],
              'solver': ['newton-cg', 'lbfgs', 'liblinear', 'saga'],
              'fit_intercept': [True, False],
              'class_weight': ['balanced', None]
             }

lrGridSearch = LogisticRegression()

clfLR = GridSearchCV(lrGridSearch, parameters, cv=5, n_jobs=20, verbose=10)
clfLR.fit(tfidf_docs, df['class'])
```

```
Fitting 5 folds for each of 416 candidates, totalling 2080 fits
```

[26]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(), n_jobs=20,
             param_grid={'C': [0.01, 1.01, 2.01, 3.01, 4.01, 5.01, 6.01, 7.01,
                               8.01, 9.01, 10.01, 11.01, 12.01, 13.01, 14.01,
                               15.01, 16.01, 17.01, 18.01, 19.01, 20.01, 21.01,
                               22.01, 23.01, 24.01, 25.01],
                         'class_weight': ['balanced', None],
                         'fit_intercept': [True, False],
                         'solver': ['newton-cg', 'lbfgs', 'liblinear', 'saga']},
             verbose=10)
```

[27]:
```
clfLR.best_estimator_
```

[27]:
```
LogisticRegression(C=4.01, fit_intercept=False, solver='newton-cg')
```

[28]:
```
clfLR.best_score_
```

[28]:
```
0.9007299805583067
```

3 Wyniki optymalizacji hiperparametrów

4 Optymalizacji podlegały 4 modele:

4.1 1. w2v SVC

Accuracy prze optymalizacją: 0.908958 (90,9%)

Accuracy po optymalizacji: 0.9126873385012919 (91,1%)

4.2 2. tf-idf SVC

Accuracy prze optymalizacją: 0.901592 (90,1%)

Accuracy po optymalizacji: 0.9075785921279482 (90,7%)

4.3 3. tsvd SVC

Accuracy prze optymalizacją: 0.894696 (89,4%)

Accuracy po optymalizacji: 0.8981145225780418 (89,8%)

4.4 4. tf-idf Logistic regression

Accuracy prze optymalizacją: 0.892168 (89%)

Accuracy po optymalizacji: 0.9007299805583067 (90%) Z powyższego zestawienia wyników widać, że w wyniku optymalizacji hiperparametrów udało się nieznacznie poprawić dokładność wszystkich klasyfikatorów. Modele klasyfikacji po optymalizacji mają wyższą dokładność o maksymalnie 1 punkt procentowy (regresja logistyczne na wektorach tf-idf).

4.4.1 Ostatecznie najlepszym modelem jaki udało się uzyskać jest model wykorzystujący klasyfikator maszyny wektorów nośnych przy użyciu w2v z wynikiem dokładności 91,1%. Jest to wysoka dokładność i dość dobry model.

Parametry najlepszego modelu będącego wynikiem analizy przeprowadzonej w ramach projektu:

Maszyna wektorów nośnych: - C = 9.01 - decision_function_shape=‘ovo’ - gamma = ‘scale’ - kernel =‘rbf’ - shrinking=True