Eksploracja danych internetowych

Laboratorium 5

Prowadzący: pracownik UR
Wykonał: Piotr Rojek, pr125159

Zadanie 1
Przy użyciu techniki GridSearchCV wykonaj porównanie modeli klasyfikacyjnych trzech reprezentacji wektorowych:

- **Count Vectorizer**
- **N-Gram**
- **TF-IDF**

Przeprowadź dyskusję wyników.

```
import re
import nltk
import pandas as pd
import string
import warnings

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

# nltk.download('stopwords')
stopwords = nltk.corpus.stopwords.words("english")
ps = nltk.PorterStemmer()

warnings.filterwarnings( action: "ignore", category=DeprecationWarning)

data = pd.read_csv( filepath_or_buffer: "SMSSpamCollection.tsv", sep='\t', header=None)
data.columns = ['label', 'body_text']

pd.set_option('display.width', 225)
pd.set_option('display.max_columns', 10)
pd.set_option('display.max_colwidth', 50)

def count_punctuation(text):  1 usage
    count = sum([1 for char in text if char in string.punctuation])
    return round(count / (len(text) - text.count(" ")), 3) * 100
```

```
def clean_text(text):  1 usage
    if not isinstance(text, int) and not isinstance(text, float):
        no_punctuation = "".join([char for char in text if char not in string.punctuation])
        lower = no_punctuation.lower()
        tokens = re.split( pattern: r'\W+', lower)
        text = tokens
    text = [word for word in text if word not in stopwords]
    return text
```

```
def steaming(tokenized_text):  1 usage
    text = [ps.stem(word) for word in tokenized_text]
    return text

data['body_text_stemmed'] = data['body_text'].apply(clean_text).apply(steaming)
data['body_text_stemmed'] = (data['body_text_stemmed']
                              .apply(lambda text: ' '.join(text) if isinstance(text, list) else ''))
data['body_len'] = data['body_text'].apply(lambda x: len(x) - x.count(" "))
data['punctuation_%'] = data['body_text'].apply(lambda x: count_punctuation(x))
```

```
# Count Vectorizer
count_vect = CountVectorizer()
x_count = count_vect.fit_transform(data['body_text_stemmed'])
X_count_feat = pd.concat( objs: [data['body_len'], data['punctuation_%'],
                              pd.DataFrame(x_count.toarray(), columns=count_vect.get_feature_names_out())], axis=1)
# print("Count Vectorizer:\n", X_count_feat.head(), "\n")

# N-Gram
ngram_vect = CountVectorizer(ngram_range=(2,2))
x_ngram = ngram_vect.fit_transform(data['body_text_stemmed'])
X_ngram_feat = pd.concat( objs: [data['body_len'], data['punctuation_%'],
                              pd.DataFrame(x_ngram.toarray(), columns=ngram_vect.get_feature_names_out())], axis=1)
# print("N-Gram:\n", X_ngram_feat.head(), "\n")

# TF-IDF
tfidf_vect = TfidfVectorizer()
x_tfidf = tfidf_vect.fit_transform(data['body_text_stemmed'])
X_tfidf_feat = pd.concat( objs: [data['body_len'], data['punctuation_%'],
                              pd.DataFrame(x_tfidf.toarray(), columns=tfidf_vect.get_feature_names_out())], axis=1)
# print("TF-IDF:\n", X_tfidf_feat.head(), "\n")
```

```
# GridSearchCV

rf = RandomForestClassifier()
param = {'n_estimators': [10, 150, 300], 'max_depth': [30, 60, 90, None]}
gs = GridSearchCV(rf, param, cv=5, n_jobs=-1)

gs_fit_count = gs.fit(X_count_feat, data['label'])
print("Count:\n", pd.DataFrame(gs_fit_count.cv_results_).sort_values( by: 'mean_test_score', ascending=False)[0:5])

gs_fit_ngram = gs.fit(X_ngram_feat, data['label'])
print("N-Gram:\n", pd.DataFrame(gs_fit_ngram.cv_results_).sort_values( by: 'mean_test_score', ascending=False)[0:5])

gs_fit_tfidf = gs.fit(X_tfidf_feat, data['label'])
print("TF-IDF:\n", pd.DataFrame(gs_fit_tfidf.cv_results_).sort_values( by: 'mean_test_score', ascending=False)[0:5])

print("Najlepszy mean_test_score:")

gs.fit(X_count_feat, data['label'])
print("Count:", round(gs.best_score_, 5))

gs.fit(X_ngram_feat, data['label'])
print("N-Gram:", round(gs.best_score_, 5))

gs.fit(X_tfidf_feat, data['label'])
print("TF-IDF:", round(gs.best_score_, 5))
```

Testy:

```
Count:
    mean_fit_time  std_fit_time  mean_score_time  std_score_time param_max_depth  ...  split3_test_score  split4_test_score  mean_test_score  std_test_score  rank_test_score
10      17.937267      0.343770         0.154968        0.026877            None  ...           0.968582           0.970377         0.973617        0.003799                1
7       17.345489      0.471931         0.200279        0.012649              90  ...           0.968582           0.973070         0.973258        0.003189                2
11      25.724992      0.571347         0.151303        0.015432            None  ...           0.970377           0.970377         0.973079        0.002280                3
8       32.863902      0.466363         0.253013        0.034214              90  ...           0.966786           0.971275         0.972540        0.003801                4
6        1.692553      0.161996         0.112161        0.012713              90  ...           0.964093           0.969479         0.970745        0.003757                5

[5 rows x 15 columns]
```

```
N-Gram:
    mean_fit_time  std_fit_time  mean_score_time  std_score_time param_max_depth  ...  split3_test_score  split4_test_score  mean_test_score  std_test_score  rank_test_score
11      94.849783      1.846754         0.380625        0.027227            None  ...           0.944345           0.950628         0.949030        0.002512                1
10      63.092635      1.635151         0.397301        0.033374            None  ...           0.946140           0.947935         0.949030        0.001846                1
9        7.595357      0.336770         0.496689        0.159212            None  ...           0.934470           0.936266         0.938980        0.003221                3
6        6.061857      1.331968         0.587655        0.204747              90  ...           0.925494           0.935368         0.928751        0.004737                4
7       36.943602      0.880433         0.826666        0.213086              90  ...           0.921005           0.928187         0.928211        0.004306                5

[5 rows x 15 columns]
```

```
TF-IDF:
    mean_fit_time  std_fit_time  mean_score_time  std_score_time param_max_depth  ...  split3_test_score  split4_test_score  mean_test_score  std_test_score  rank_test_score
11      25.131840      0.457054         0.140176        0.018235            None  ...           0.969479           0.973070         0.974514        0.003044                1
7       15.952247      0.238358         0.196436        0.031546              90  ...           0.968582           0.971275         0.973976        0.003461                2
8       31.410926      0.411874         0.261430        0.017647              90  ...           0.967684           0.971275         0.973258        0.003480                3
10      17.234838      0.452019         0.147486        0.026116            None  ...           0.968582           0.971275         0.973079        0.002959                4
5       25.438713      0.638413         0.248299        0.013025              60  ...           0.966786           0.972172         0.971822        0.003098                5

[5 rows x 15 columns]
```

```
Najlepszy mean_test_score:
Count: 0.97326
N-Gram: 0.94921
TF-IDF: 0.97523
```

Dyskusja wyników:

Na podstawie przeprowadzonych testów z użyciem GridSearchCV, najlepsze rezultaty klasyfikacji osiągnięto przy wykorzystaniu TF-IDF, gdzie mean_test_score był najwyższy. Niewiele gorzej wypadł CountVectorizer, co oznacza, że sama obecność słów w tekście również niesie dużą wartość informacyjną w kontekście klasyfikacji wiadomości jako spam lub ham. Najsłabszy wynik uzyskano dla reprezentacji N-Gram, co może wynikać z większego wymiaru cech i wprowadzenia rzadkich kombinacji wyrazów, które nie zawsze poprawiają jakość modelu. Podsumowywując, TF-IDF okazał się najbardziej efektywnym podejściem, łącząc wysoką skuteczność z odpornością na szum w danych.