project1

December 27, 2023

1 Team Members

1.1 1. Mahmoud Salah Ahmed 20180254

1.2 2. Alaa Eldin Ebrahim 20200330

1.3 3. Hana Hany Ayman 20201213

1.4 4. Donia Ahmed Abo Zeid 20201060

1.5 5. AbdAllah Shouker 20200301

2 Required imports

[44]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import spacy
from spacy.lang.en.stop_words import STOP_WORDS
from sklearn.svm import LinearSVC
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix
import string
import warnings
```

[45]:
```
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning, module="scikeras")
warnings.filterwarnings("ignore", message=".*'token_pattern'.*")
```

3 Data Preprocessing

[46]:
```
nlp = spacy.load('en_core_web_sm')
stopwords = list(STOP_WORDS)
stopwords.remove('not')
```

[59]:
```
data = pd.read_csv('sentimentdataset (Project 1).csv')
print(data.head(10))
```

```
  Source  ID                                            Message  Target
0   Yelp   0                                Crust is not good.       0
1   Yelp   1         Not tasty and the texture was just nasty.       0
2   Yelp   2  Stopped by during the late May bank holiday of…       1
3   Yelp   3  The selection on the menu was great and so wer…       1
4   Yelp   4     Now I am getting angry and I want my damn pho.       0
5   Yelp   5             Honeslty it didn't taste THAT fresh.)       0
6   Yelp   6  The potatoes were like rubber and you could te…       0
7   Yelp   7                          The fries were great too.       1
8   Yelp   8                                  A great touch.       1
9   Yelp   9                           Service was very prompt.       1
```

[60]:
```
data = data.drop(columns=['ID', 'Source'])
```

[62]:
```
print(data['Target'].value_counts())
```

```
Target
1    1385
0    1360
Name: count, dtype: int64
```

[48]:
```
def text_data_cleaning(sentence):
    doc = nlp(sentence)

    tokens = []  # list of tokens
    for token in doc:
        if token.lemma_ != "-PRON-":
            temp = token.lemma_.lower().strip()
        else:
            temp = token.lower_
        tokens.append(temp)

    cleaned_tokens = []
    for token in tokens:
        if token not in stopwords and token not in string.punctuation:
            cleaned_tokens.append(token)
    return cleaned_tokens
```

[49]:
```
X = data['Message']
y = data['Target']
```

[50]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

4 Linear SVC

[51]:
```
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(tokenizer=text_data_cleaning)),
    ('clf', LinearSVC()),
])
```

[52]:
```
param_grid = {
    'tfidf__max_df': [0.5, 0.75, 1.0],
    'tfidf__ngram_range': [(1, 1), (1, 2)],
    'clf__C': [0.1, 1, 10],
}
```

[53]:
```
grid_search = GridSearchCV(pipeline, param_grid, cv=5)

grid_search.fit(X_train, y_train)
```

[53]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('tfidf',
                                        TfidfVectorizer(tokenizer=<function
text_data_cleaning at 0x7f2b887be680>)),
                                       ('clf', LinearSVC())]),
             param_grid={'clf__C': [0.1, 1, 10],
                         'tfidf__max_df': [0.5, 0.75, 1.0],
                         'tfidf__ngram_range': [(1, 1), (1, 2)]})
```

[54]:
```
best_params = grid_search.best_params_
print("Best Parameters:", best_params)
```

```
Best Parameters: {'clf__C': 1, 'tfidf__max_df': 0.5, 'tfidf__ngram_range': (1,
2)}
```

[55]:
```
y_pred = grid_search.predict(X_test)
```

[56]:
```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
[[220  45]
 [ 42 242]]
              precision    recall  f1-score   support

           0       0.84      0.83      0.83       265
           1       0.84      0.85      0.85       284

    accuracy                           0.84       549
   macro avg       0.84      0.84      0.84       549
weighted avg       0.84      0.84      0.84       549
```

```
Accuracy: 0.8415300546448088
```

[57]:
```
import joblib
best_model = grid_search.best_estimator_
joblib.dump(best_model, 'linear_svm_best_model.joblib')
```

[57]:
```
['linear_svm_best_model.joblib']
```

[58]:
```
loaded_model = joblib.load('linear_svm_best_model.joblib')

new_data = ["it is very good", 'it is bad', 'awesome', 'I am not comfortable␣
 ↪with that']
predictions = loaded_model.predict(new_data)

print(predictions)
```

```
[1 0 1 0]
```

5 ANN

[8]:
```
from sklearn.neural_network import MLPClassifier
```

[10]:
```
data['processed_text'] = data['Message'].apply(text_data_cleaning)
```

[12]:
```
data['processed_text'] = [' '.join(sentence) for sentence in␣
 ↪data['processed_text']]
```

[13]:
```
data
```

[13]:

	Message	Target	processed_text
0	Crust is not good.	0	crust not good
1	Not tasty and the texture was just nasty.	0	not tasty texture nasty
2	Stopped by during the late May bank holiday of…	1	stop late bank holiday rick steve recommendati…
3	The selection on the menu was great and so wer…	1	
4	Now I am getting angry and I want my damn pho.	0	
…	…	…	
2740	The screen does get smudged easily because it …	0	
2741	What a piece of junk.. I lose more calls on th…	0	
2742	Item Does Not Match Picture.	0	
2743	The only thing that disappoint me is the infra…	0	
2744	You can not answer calls with the unit, never …	0	

```
3                              selection menu great price
4                                    angry want damn pho
…                                                    …
2740                 screen smudge easily touch ear face
2741                           piece junk .. lose phone
2742                             item not match picture
2743                thing disappoint infra red port irda
2744                                  not answer unit work

[2745 rows x 3 columns]
```

[14]:
```
data = data.drop(['Message'], axis=1)
```

[15]:
```
X = data['processed_text']
y = data['Target']
```

[17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

[34]:
```
parameters = {
    'hidden_layer_sizes': [(50,), (100,), (128,), (256,), (512,)],
    'learning_rate_init': [0.001, 0.01, 0.1],
    'batch_size': [32, 64, 128],
}
```

[35]:
```
tfidf_vectorizer = TfidfVectorizer()

X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

[36]:
```
ann_model = MLPClassifier(max_iter=500)
```

[37]:
```
grid_search = GridSearchCV(estimator=ann_model, param_grid=parameters, cv=3,␣
↪scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train_tfidf, y_train)
```

[37]:
```
GridSearchCV(cv=3, estimator=MLPClassifier(max_iter=500), n_jobs=-1,
             param_grid={'batch_size': [32, 64, 128],
                         'hidden_layer_sizes': [(50,), (100,), (128,), (256,),
                                                (512,)],
                         'learning_rate_init': [0.001, 0.01, 0.1]},
             scoring='accuracy')
```

[38]:
```
best_params = grid_search.best_params_
```

[39]:
```
print(best_params)
```

```
{'batch_size': 64, 'hidden_layer_sizes': (256,), 'learning_rate_init': 0.1}
```

[40]:
```
best_ann_model = MLPClassifier(max_iter=500, **best_params)
best_ann_model.fit(X_train_tfidf, y_train)
```

[40]:
```
MLPClassifier(batch_size=64, hidden_layer_sizes=(256,), learning_rate_init=0.1,
              max_iter=500)
```

[41]:
```
y_pred = best_ann_model.predict(X_test_tfidf)
print("Classification Report:")
print(classification_report(y_test, y_pred))
print(f'Accuray: {accuracy_score(y_pred, y_test)}')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.80      0.80       265
           1       0.82      0.82      0.82       284

    accuracy                           0.81       549
   macro avg       0.81      0.81      0.81       549
weighted avg       0.81      0.81      0.81       549

Accuray: 0.8105646630236795
```

[42]:
```
import joblib
joblib.dump(best_ann_model, 'best_ann_model.joblib')
```

[42]:
```
['best_ann_model.joblib']
```

[43]:
```
loaded_model = joblib.load('best_ann_model.joblib')
new_data = ["This is a positive sentence.", "This is a negative sentence."]
processed_new_data = [text_data_cleaning(sentence) for sentence in new_data]
processed_new_data = [' '.join(sentence) for sentence in processed_new_data]
new_data_tfidf = tfidf_vectorizer.transform(processed_new_data)
predictions = loaded_model.predict(new_data_tfidf)
print("Predictions on new data:")
print(predictions)
```

```
Predictions on new data:
[1 0]
```

[ ]:
```

```