Narzędzie do identyfikacji podobnych tekstów (znaczenie, a nie tekst).

Projekt z przedmiotu

Przetwarzanie Języka Naturalnego

Maciej Włosek

Spis treści

1. Wprowadzenie

Analiza danych tekstowych jest jednym z ważnych obszarów w dziedzinie analizy danych i uczenia maszynowego. W dzisiejszym świecie, w którym generujemy ogromne ilości tekstu w postaci pytań, recenzji, opinii czy tweetów, istnieje potrzeba efektywnego przetwarzania i rozumienia tych tekstów. W kontekście przetwarzania języka naturalnego (NLP), zadanie rozpoznawania podobieństwa między dwoma tekstami ma szczególne znaczenie.

W tym kontekście przedstawiony tutaj kod ma na celu zbudowanie modelu klasyfikacji, który jest w stanie ocenić, czy dwa pytania w zbiorze danych są powiązane (is_duplicate), czy nie. Taki model może znaleźć zastosowanie w wielu dziedzinach, takich jak wyszukiwarki, rekomendacje treści, automatyczne generowanie odpowiedzi lub filtrowanie duplikatów.

2. Opis technologiczny

Aplikacja została napisana w języku Python i korzysta z różnych bibliotek i narzędzi do przetwarzania danych tekstowych, trenowania modelu i oceny wyników. Oto najważniejsze technologie i biblioteki użyte w tym projekcie:

Python: Język programowania, który pozwala na elastyczne i wydajne tworzenie kodu.

pandas: Biblioteka do manipulacji i analizy danych, która umożliwia łatwe wczytywanie danych z plików CSV i manipulację nimi w postaci DataFrame'ów.

scikit-learn: Biblioteka do uczenia maszynowego, która dostarcza narzędzia do przetwarzania wstępnego danych, budowy modeli i oceny wyników.

XGBoost: Biblioteka do gradient boosting, która oferuje wydajne i skuteczne modele klasyfikacji i regresji.

numpy: Biblioteka do obliczeń naukowych, używana do operacji na macierzach i wektorach.

matplotlib i seaborn: Biblioteki do wizualizacji danych, które pozwalają na tworzenie wykresów i grafik dla lepszej interpretacji wyników.

contractions: Biblioteka, która umożliwia rozwinięcie skrótów językowych w pytaniach, co może poprawić jakość przetwarzania wstępnego.

re i string: Moduły Pythona służące do manipulacji tekstami, takie jak usuwanie znaków specjalnych, filtrowanie treści itp.

mlxtend: Biblioteka do rozszerzonych wizualizacji danych, która oferuje możliwość tworzenia rozbudowanych macierzy pomyłek.

3. Schemat i działanie aplikacji

Aplikacja składa się z kilku etapów, które obejmują wczytanie danych, przetwarzanie wstępne, trenowanie modelu i ocenę wyników. Poniżej przedstawiono główne kroki w działaniu aplikacji:

1. **Rozpakowanie plików danych**: Na początku aplikacja rozpakowuje pliki "train.csv.zip" i "test.csv.zip" do plików CSV "train.csv" i "test.csv" odpowiednio. Wykorzystywana jest funkcja shutil.unpack_archive().

```
# Rozpakowanie plików dancyh
shutil.unpack_archive("train.csv.zip", "train.csv")
shutil.unpack_archive("test.csv.zip", "test.csv")
```

2. **Wczytywanie danych**: Dane treningowe są wczytywane z pliku "train.csv" za pomocą funkcji read_data(). Wczytane dane są przechowywane w postaci DataFrame'u df.

```
# Wczytywanie danych
def read_data(filename, encoding="utf8"):
    df = pd.read_csv(filename)
    # Drop null values
    df.dropna(inplace=True)
    df.reset_index(drop=True, inplace=True)
    print("\nNumber of records:", len(df))
    return df

df = read_data("train.csv/train.csv")

# Wybór kolumn
df = df[["question1","question2","is_duplicate"]]
df.head()
```

3. **Przetwarzanie wstępne danych**: Tekst w kolumnach "question1" i "question2" DataFrame'u df jest poddawany różnym operacjom przetwarzania wstępnego za pomocą funkcji preprocess_data(). Celem jest standaryzacja i czyszczenie danych tekstowych, takie jak zamiana na małe litery, usuwanie znaków specjalnych, linków, skrótów językowych itp. Wynik przetwarzania wstępnego jest zapisywany w nowym DataFrame'u df_copy.

```
#Przetwarzanie wstępne dancych
def preprocess_data(text):
    # conver to string
    text = str(text)
    # lowercase
    text = text.lower()
    # remove contractions
    text = contractions.fix(text)
    # remove hashtags
    text = re.sub(r'#(\w+)','',text)
    # remove special characters
    text = re.sub(r'[^\w ]+','',text)
    # remove links if any
    text = re.sub(r'https?://\S+|www\.\S+','',text)
    # remove non-ascii
    text = ''.join(word for word in text if ord(word) < 128)
    # remove punctuation
    text = text.translate(str.maketrans('','',string.punctuation))
    # remove digits
    text = re.sub(r'[\d]+','',text)
    # remove single letters
    text = ' '.join(word for word in text.split() if len(word) > 1)
    # remove multiple spaces
    text = ' '.join(text.split())

    return text

#Przetwarzanie wstępne na kopii
df_copy = df.copy()
df_copy.loc[:,"question1"] = df_copy["question1"].apply(preprocess_data)
df_copy.loc[:,"question2"] = df_copy["question2"].apply(preprocess_data)
df_copy.head()

#Generowanie liczby wystąpień dla unikalnej wartości
df_copy['is_duplicate'].value_counts()
```

4. **Podział danych**: Dane w DataFrame'ie df_copy są podzielone na zbiór treningowy i deweloperski (testowy) za pomocą funkcji train_test_split() z pakietu sklearn.model_selection. Ustalono, że 70% danych zostanie wykorzystane do treningu, a 30% do oceny wyników.

```
# Podział danych
df_train,df_dev = train_test_split(df_copy,
                                   test_size=0.3,
                                   stratify=df_copy['is_duplicate'],
                                   random_state=42)

print("Training data shape:",df_train.shape)
print("Dev data shape:",df_dev.shape)
```

5. **Wektoryzacja TF-IDF**: Wykorzystując klasę TfidfVectorizer z pakietu sklearn.feature_extraction.text, tekstowe pytania z danych treningowych i deweloperskich są zamieniane na wektory cech TF-IDF. Wektory cech reprezentują znaczenie słów w pytaniach i stanowią wejście dla modelu klasyfikacji.

```
#Wektoryzacja TF-IDF
tfidf = TfidfVectorizer()
train_tfidf = tfidf.fit_transform(df_train['question1']+'
'+df_train['question2'])
dev_tfidf = tfidf.transform(df_dev['question1']+' '+df_dev['question2'])

train_tfidf.shape,dev_tfidf.shape
```

6. **Trenowanie modelu**: Wykorzystując klasyfikator XGBoost (xgb.XGBClassifier()), model jest trenowany na danych treningowych. Model stara się nauczyć zależności między pytaniem a etykietą "is_duplicate".

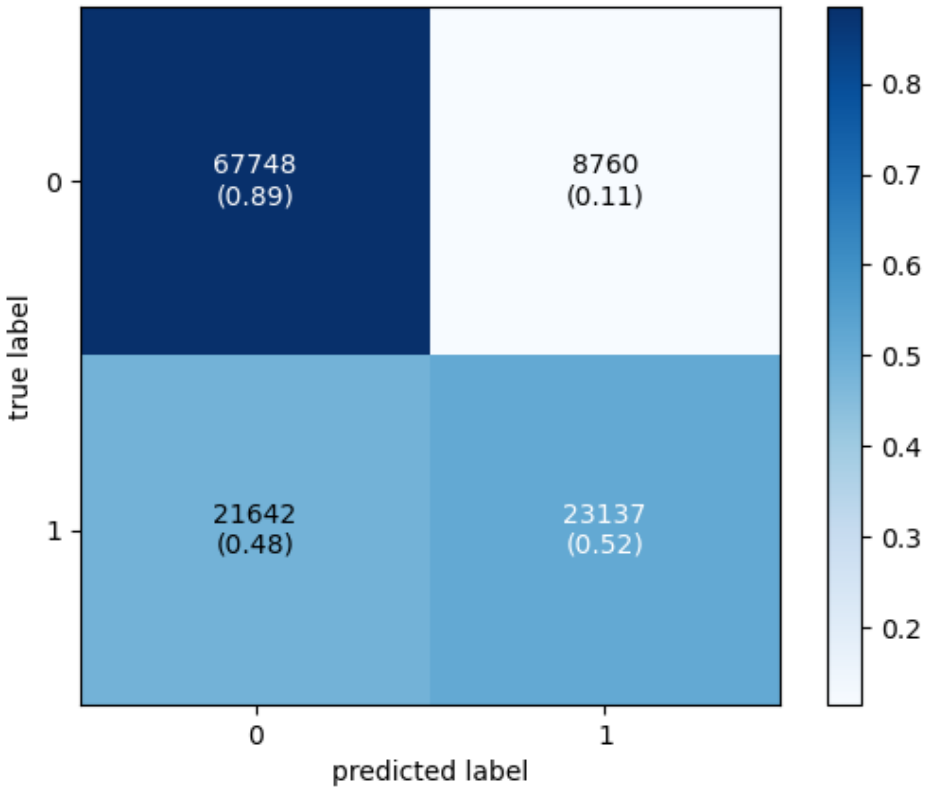
```
#Trenowanie modelu
labels = df_train['is_duplicate']
model_xgb = xgb.XGBClassifier()
model_xgb.fit(train_tfidf,labels)
```

7. **Predykcja i ocena wyników**: Na podstawie danych deweloperskich, model dokonuje predykcji i przewiduje etykiety "is_duplicate". Dokładność modelu jest oceniana na podstawie porównania przewidywanych etykiet z rzeczywistymi etykietami za pomocą funkcji accuracy_score() z pakietu sklearn.metrics.

```
#Predykcja i ocena wyników
predictions = model_xgb.predict(dev_tfidf)
predictions = list(predictions)

print('Accuracy score:',accuracy_score(df_dev['is_duplicate'],predictions))
```

8. **Wizualizacja wyników**: Wykorzystując biblioteki matplotlib, seaborn i mlxtend.plotting, aplikacja generuje różne wykresy i grafiki, takie jak macierz pomyłek, wykresy dokładności, które pozwalają na lepszą interpretację wyników i ocenę działania modelu.



4. Wnioski

Liczba rekordów: Zestaw danych składa się z 404 287 rekordów.

Podział danych: Dane zostały podzielone na zbiór treningowy i deweloperski (testowy). Zbiór treningowy zawiera 283 000 rekordów, natomiast zbiór deweloperski zawiera 121 287 rekordów.

Dokładność (Accuracy score): Model XGBoost osiągnął dokładność na poziomie 0.749, co oznacza, że około 74,9% przewidywań było zgodnych z rzeczywistymi etykietami w zbiorze deweloperskim.

Model XGBoost osiągnął przyzwoitą dokładność, jednak warto zwrócić uwagę na specyfikę problemu i kontekst aplikacji. W niektórych przypadkach może być konieczne zastosowanie innych miar oceny modelu, takich jak czułość, swoistość czy F1-score, szczególnie jeśli istnieje nierównowaga między klasami lub pewne błędy są bardziej kosztowne niż inne.

W przypadku dalszej analizy i ulepszania modelu, warto przeprowadzić dodatkowe etapy optymalizacji, takie jak strojenie hiperparametrów, zastosowanie technik regularyzacji czy uwzględnienie innych algorytmów klasyfikacji. Może to pomóc w poprawie dokładności i ogólnej wydajności modelu.

Ważne jest również zrozumienie kontekstu danych i potencjalnych ograniczeń zbioru treningowego. W przypadku zastosowania tego modelu w rzeczywistych zastosowaniach, warto dokładnie zbadać charakterystykę zbioru danych, jak również przeprowadzić walidację krzyżową i analizę błędów w celu oceny wydajności i dalszej optymalizacji.