

# Analiza i przewidywanie podobieństwa pytań Quora

Autor: WS

2 czerwca 2024

## Streszczenie

Projekt obejmuje analizę i przewidywanie podobieństwa pytań w zestawie danych Quora Question Pairs. Przetwarzane są dane tekstowe, przeprowadzana jest ekstrakcja cech za pomocą TF-IDF oraz różnych miar podobieństwa, a następnie budowany jest model uczenia maszynowego wykorzystując eXtreme Gradient Boosting (XGBoost). Wyniki są oceniane pod kątem metryki accuracy i F1-score.

## 1 Wstęp

### 1.1 Cel

Celem projektu jest stworzenie modelu, który będzie w stanie przewidzieć, czy dwie pytania z Quora są duplikatami. Analiza taka może pomóc w lepszym zarządzaniu treścią na platformie oraz poprawie doświadczeń użytkowników.

### 1.2 Zakres

Projekt obejmuje pełen proces przetwarzania danych, od czyszczenia i przygotowania danych tekstowych, przez ekstrakcję cech, aż po budowę i ocenę modelu predykcyjnego. Wykorzystałam metody takie jak wektory TF-IDF, podobieństwo kosinusowe, miara Jaccarda oraz odległość Levenshteina.

### 1.3 Metodyka

W projekcie użyłam następujących narzędzi i metod:

- Python i biblioteki: numpy, pandas, nltk, scikit-learn, scipy, xgboost
- Przetwarzanie języka naturalnego: stemming, usuwanie stop-words, usuwanie kontrakcji
- Ekstrakcja cech: wektory TF-IDF, podobieństwo kosinusowe, miara Jaccarda, odległość Levenshteina
- Modelowanie: XGBoost, GridSearchCV do optymalizacji hiperparametrów

## 2 Część Teoretyczna

### 2.1 Stemming

Stemming jest techniką redukcji wyrazów do ich podstawowych form, czyli rdzeni. Przykładowo, słowa takie jak “running” i “runner” zostaną zredukowane do “run”. W projekcie wykorzystywany jest Porter Stemmer, który jest jednym z najpopularniejszych algorytmów stemmingu.

### 2.2 Usuwanie Stop-Words

Usuwanie stop-words to kolejny ważny krok w przetwarzaniu tekstu. Stop-words to najczęściej używane słowa w języku, takie jak “and”, “the” czy “is”, które nie niosą ze sobą istotnej wartości semantycznej. Usunięcie tych słów pomaga w skupieniu się na bardziej znaczących wyrazach w tekście.

## 2.3 Konwersja Tekstu na Małe Litery

Konwersja tekstu na małe litery ma na celu zredukowanie różnorodności wynikającej z używania różnych form zapisu wyrazów. Jest to prosty, ale skuteczny sposób na zwiększenie jednorodności danych tekstowych.

## 2.4 Metoda TF-IDF

Metoda TF-IDF (ang. Term Frequency-Inverse Document Frequency) jest jednym z najpopularniejszych sposobów reprezentacji tekstu w postaci wektorowej. TF-IDF mierzy ważność słowa w dokumencie, biorąc pod uwagę jego częstość występowania w dokumencie (TF) oraz rzadkość w całym korpusie (IDF).

### 2.4.1 Wartość TF-IDF

Wartość TF-IDF dla słowa  $t$  w dokumencie  $d$  jest iloczynem dwóch wskaźników: częstości terminu (TF) i odwrotnej częstości dokumentów (IDF):

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

### 2.4.2 Term Frequency (TF)

Częstość terminu (TF) jest zdefiniowana jako liczba wystąpień słowa  $t$  w dokumencie  $d$  podzielona przez całkowitą liczbę słów w dokumencie  $d$ :

$$\text{TF}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

### 2.4.3 Inverse Document Frequency (IDF)

Odwrotna częstość dokumentów (IDF) jest zdefiniowana jako logarytm dziesiętny ilorazu całkowitej liczby dokumentów w korpusie przez liczbę dokumentów zawierających słowo  $t$ :

$$\text{IDF}(t) = \log \left( \frac{N}{|\{d \in D : t \in d\}|} \right)$$

gdzie  $N$  to całkowita liczba dokumentów w korpusie, a  $|\{d \in D : t \in d\}|$  to liczba dokumentów zawierających słowo  $t$ .

TF-IDF pozwala na lepsze uwzględnienie istotnych słów, które mogą być kluczowe dla zrozumienia tekstu, w przeciwieństwie do bardziej podstawowych metod, takich jak bag-of-words. Jest to kluczowe dla ekstrakcji cech w projekcie.

## 2.5 Miary podobieństwa

### 2.5.1 Podobieństwo cosinusowe

Miara oparta na kącie między wektorami reprezentującymi pytania. Jest to jedna z najczęściej stosowanych metod do oceny podobieństwa tekstu. Podobieństwo cosinusowe jest obliczane jako iloczyn skalarny dwóch wektorów podzielony przez iloczyn ich norm. Formalnie, podobieństwo cosinusowe między dwoma wektorami  $\mathbf{A}$  i  $\mathbf{B}$  można wyrazić wzorem:

$$\text{cosine\_similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} \quad (1)$$

gdzie  $\mathbf{A} \cdot \mathbf{B}$  to iloczyn skalarny wektorów, a  $\|\mathbf{A}\|$  i  $\|\mathbf{B}\|$  to ich normy (długości).

### 2.5.2 Miara Jaccarda

Stosunek wspólnych elementów do całkowitej liczby unikalnych elementów w dwóch zbiorach. W kontekście tekstu, miara ta pozwala na ocenę podobieństwa między dwoma zestawami słów. Formalnie, miarę Jaccarda można zdefiniować jako:

$$\text{Jaccard}(\mathbf{A}, \mathbf{B}) = \frac{|\mathbf{A} \cap \mathbf{B}|}{|\mathbf{A} \cup \mathbf{B}|} \quad (2)$$

gdzie  $\mathbf{A} \cap \mathbf{B}$  to liczba wspólnych elementów (przecięcie zbiorów), a  $\mathbf{A} \cup \mathbf{B}$  to liczba unikalnych elementów (suma zbiorów).

### 2.5.3 Odległość Levenshteina

Liczba operacji wymaganych do przekształcenia jednego ciągu znaków w drugi. Operacje te mogą obejmować wstawienia, usunięcia lub zamiany pojedynczych znaków. Odległość Levenshteina, znana również jako odległość edycyjna, jest szczególnie przydatna w analizie tekstu, gdzie mogą występować literówki i drobne różnice. Formalnie, odległość Levenshteina dla dwóch ciągów znaków  $A$  i  $B$  jest zdefiniowana jako:

$$\text{Lev}(A, B) = \begin{cases} 0 & \text{if } A = B \\ \min \begin{cases} \text{Lev}(A_{1..m-1}, B) + 1 \\ \text{Lev}(A, B_{1..n-1}) + 1 \\ \text{Lev}(A_{1..m-1}, B_{1..n-1}) + \mathbf{1}_{A_m \neq B_n} \end{cases} & \text{otherwise} \end{cases} \quad (3)$$

gdzie  $A_{1..m-1}$  i  $B_{1..n-1}$  oznaczają odpowiednio ciągi znaków  $A$  i  $B$  bez ostatniego znaku, a  $\mathbf{1}_{A_m \neq B_n}$  jest funkcją charakterystyczną, która wynosi 1, gdy ostatnie znaki  $A$  i  $B$  są różne, a 0, gdy są takie same.

## 3 Część Praktyczna

### 3.1 Przygotowanie danych

Przygotowanie danych jest kluczowym etapem w każdym projekcie analizy danych, a szczególnie w przetwarzaniu języka naturalnego. Proces ten rozpoczynamy od usunięcia brakujących wartości, co jest niezbędne, aby uniknąć błędów w dalszej analizie. W naszym projekcie, brakujące wartości w kolumnach zawierających pytania są usuwane, aby zapewnić spójność danych. Następnie przystępujemy do usuwania znaków specjalnych, takich jak HTML tags, które mogą zakłócać proces analizy tekstu. Kolejnym krokiem jest normalizacja tekstu poprzez konwersję wszystkich znaków na małe litery oraz usunięcie znaków interpunkcyjnych, co pomaga w ujednoliceniu formy tekstu i redukuje jego złożoność. Ostatecznie, przetworzony tekst jest gotowy do dalszej analizy i ekstrakcji cech.

```
stop_words = set(stopwords.words('english'))
ps = PorterStemmer()

def preprocess(text):
    text = re.sub(r'<.*?>', ' ', text)
    text = text.lower()
    text = contractions.fix(text)
    text = text.translate(str.maketrans('', '', string.punctuation))
    words = text.split()
    words = [ps.stem(word) for word in words if word.isalpha() and word not in stop_words]
    return ' '.join(words)

train_df['question1_processed'] = train_df['question1'].astype(str).progress_apply(
    preprocess)
train_df['question2_processed'] = train_df['question2'].astype(str).progress_apply(
    preprocess)
```

### 3.2 Ekstrakcja cech

Ekstrakcja cech jest kluczowym procesem, który pozwala na przekształcenie tekstu w reprezentacje wektorowe, niezbędne do modelowania. W naszym projekcie wykorzystujemy metodę TF-IDF (Term Frequency-Inverse Document Frequency) do reprezentacji pytań. TF-IDF pozwala na uwzględnienie zarówno częstości występowania słów w pytaniu, jak i ich rzadkości w całym zbiorze danych, co pomaga w identyfikacji najbardziej istotnych słów. Po stworzeniu wektorów TF-IDF dla każdego pytania, przystępuję do obliczania różnych miar podobieństwa, takich jak podobieństwo kosinusowe i miara Jaccarda. Miary te pozwalają na ocenę stopnia podobieństwa między dwoma pytaniami, co jest kluczowe dla naszego celu identyfikacji duplikatów. Dodatkowo, obliczamy różnice długości pytań oraz odległość Levenshteina, aby uwzględnić różnice w długości i strukturze tekstu. Ekstrakcja tych cech stanowi podstawę dla budowy modelu predykcyjnego.

```
combined_text = X.question1_processed.values.tolist() + X.question2_processed.values.tolist()

tfidf = TfidfVectorizer(lowercase=True)
tfidf.fit(combined_text)

q1_tfidf = tfidf.transform(X.question1_processed.values)
q2_tfidf = tfidf.transform(X.question2_processed.values)

def jaccard_similarity(row):
    q1_words = set(row['question1_processed'].split())
    q2_words = set(row['question2_processed'].split())
    intersection = len(q1_words & q2_words)
    union = len(q1_words | q2_words)
    if union == 0:
        return 0.0
    return intersection / union

cosine_similarities = []
for q1, q2 in tqdm(zip(q1_tfidf, q2_tfidf), total=len(train_df), desc='Calculating CS'):
    cosine_similarities.append(cosine_similarity(q1, q2)[0][0])

train_df['cosine_similarity'] = cosine_similarities

train_df['jaccard_similarity'] = train_df.progress_apply(jaccard_similarity, axis=1)
train_df['length_diff'] = abs(train_df['question1_processed'].str.len() - train_df['question2_processed'].str.len())
train_df['levenshtein_dist'] = train_df.progress_apply(lambda x: fuzz.ratio(x['question1_processed'], x['question2_processed']), axis=1)
```

### 3.3 Modelowanie i ocena

Modelowanie w naszym projekcie opiera się na algorytmie XGBoost, który jest jednym z najbardziej wydajnych algorytmów uczenia maszynowego. XGBoost, czyli eXtreme Gradient Boosting, jest szczególnie skuteczny w zadaniach klasyfikacji i regresji dzięki swojej zdolności do modelowania złożonych wzorców w danych. Algorytm ten optymalizuje funkcję straty, która w przypadku klasyfikacji binarnej jest zazwyczaj log loss:

$$\text{log\_loss} = -\frac{1}{n} \sum_{i=1}^n [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] \quad (4)$$

gdzie  $y_i$  to prawdziwa etykieta, a  $p_i$  to przewidywane prawdopodobieństwo klasy pozytywnej.

Aby znaleźć optymalne hiperparametry dla modelu, użyliśmy GridSearchCV, techniki przeszukiwania siatki, która pozwala na systematyczne testowanie różnych kombinacji parametrów. Proces ten obejmował testowanie różnych głębokości drzew, liczby estymatorów oraz szybkości uczenia, co pozwoliło na znalezienie najlepszej konfiguracji dla naszego modelu. Po wytrenowaniu modelu, przystąpiliśmy do oceny jego wydajności przy użyciu metryk takich jak dokładność (accuracy) oraz F1-score, które są kluczowe dla oceny jakości modelu w kontekście klasyfikacji binarnej:

$$TP = \text{Liczba prawdziwie pozytywnych przypadków} \quad (5)$$

$$TN = \text{Liczba prawdziwie negatywnych przypadków} \quad (6)$$

$$FP = \text{Liczba fałszywie pozytywnych przypadków} \quad (7)$$

$$FN = \text{Liczba fałszywie negatywnych przypadków} \quad (8)$$

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (9)$$

$$F1\text{-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (10)$$

$$\text{recall} = \frac{TP}{TP + FN} \quad (11)$$

$$\text{precision} = \frac{TP}{TP + FP} \quad (12)$$

Dokładność mierzy procent poprawnych przewidywań, podczas gdy F1-score uwzględnia zarówno precyzję, jak i czułość, dając bardziej zrównoważoną ocenę wydajności modelu.

```
features = ['cosine_similarity', 'jaccard_similarity', 'length_diff', 'levenshtein_dist']
X_additional_features = train_df[features]
y = train_df['is_duplicate']
X_combined = hstack([q1_tfidf, q2_tfidf, X_additional_features])

X_train, X_test, y_train, y_test = train_test_split(X_combined, y, test_size=0.25,
                                                    random_state=42)

params = {
    'max_depth': [3, 4, 5, 6],
    'n_estimators': [100, 200, 300, 400],
    'learning_rate': [0.01, 0.1, 0.2, 0.3]
}

xgb = XGBClassifier()
clf = GridSearchCV(xgb, params, scoring='accuracy', cv=3, verbose=10)
clf.fit(X_train, y_train)
best_model = clf.best_estimator_
y_pred = best_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(f'F1-score: {f1}')
```

### 3.4 Wyniki

Wyniki modelu XGBoost na zestawie testowym wykazały wysoką dokładność, co wskazuje na skuteczność modelu w przewidywaniu podobieństwa pytań. Dokładność modelu wynosi 0.8001, co oznacza, że około 80.01% wszystkich przewidywań było poprawnych. Model osiągnął również F1-score na poziomie 0.7174, co potwierdza jego zdolność do równoważenia precyzji i czułości. F1-score wskazuje na dobre wyważenie między liczbą prawdziwie pozytywnych i fałszywie pozytywnych wyników. Wyniki te są zgodne z oczekiwaniami, biorąc pod uwagę złożoność zastosowanych metod ekstrakcji cech oraz optymalizacji modelu. Wyniki te sugerują, że model może być skutecznie zastosowany do automatycznego wykrywania duplikatów pytań na platformach takich jak Quora, co może znacząco poprawić jakość danych i doświadczenie użytkowników.

Metryka	Wartość
Accuracy	80.01%
F1-score	71.74%

Tabela 1: Metryki modelu

## 4 Podsumowanie

Celem projektu było stworzenie modelu predykcyjnego do identyfikacji duplikatów pytań na platformie Quora. Model osiągnął wysoką dokładność i F1-score, co wskazuje na jego skuteczność w rozwiązaniu problemu. Dalsze prace mogą obejmować poprawę metod przetwarzania tekstu oraz eksplorację innych algorytmów modelowania.

## 5 Bibliografia

### Literatura

- [1] Radosław Kycia (2024). *Przetwarzanie Języka Naturalnego - wykłady i laboratoria*
- [2] <https://www.kaggle.com/competitions/quora-question-pairs/code>
- [3] Towards Data Science - TF-IDF
- [4] ChatGPT - wsparcie przy pisaniu dokumentacji