

Politechnika Krakowska
im. Tadeusza Kościuszki
Wydział Informatyki i Telekomunikacji

Projekt końcowy z przedmiotu
Przetwarzanie Języka Naturalnego
Quora Question Pairs

Lenart Piotr
Mizera Damian

Kraków, 2023

Spis treści

Spis treści	2
1. Cel i zakres pracy.....	3
2. Wstępne przetworzenie danych	4
3. Modele biblioteczne	6
4. Własny model głębokiej sieci neuronowej	8
5. Użycie modelu word2vec.....	11
6. Wykorzystanie modelu w praktyce	11
7. Podsumowanie	13
Bibliografia	13

1. Cel i zakres pracy

Celem projektu było przygotowanie programu, który będzie w stanie identyfikować podobne teksty na bazie danych dostępnych w problemie **Quora Question Pairs** na platformie Kaggle [1].

Portal Kaggle jest popularnym serwisem internetowym, na którym zarejestrowani użytkownicy mogą zmierzyć się z zadaniami konkursowymi opartymi na statystyce i przetwarzaniu danych. Jednym z tych problemów jest zadanie Quora Question Pairs, gdzie celem jest stworzenie modelu, który będzie w stanie ocenić, czy dane dwa pytania są o tym samym (nie chodzi o słowa, lecz znaczenie pytań - ich sens). Do tego zadania jest dostarczona baza pytań do nauki, gdzie podane są pytania oraz prawidłowe odpowiedzi w formie etykiet.

W tym raporcie przedstawimy nasz sposób rozwiązania postawionego zadania, programy oraz modele i ich skuteczność. Do realizacji tematu użyliśmy języka Python z bibliotekami służącymi przetwarzaniu języka naturalnego oraz tworzeniem modeli uczenia maszynowego, takich jak: NLTK [2], scikit-learn [3], Scipy [4] i Keras [5].

W ramach projektu przetestowano możliwość wykorzystania czterech bibliotecznych modeli SI, dwie własne architektury zbudowanych w Keras oraz przetestowano możliwość wykorzystania biblioteki word2vec.

2. Wstępne przetworzenie danych

Dane w formacie csv po wczytaniu do programu prezentują się w następujący sposób:

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when 23^{24} is divided by 1000	0
4	4	9	10	Which one dissolve in water quickly sugar, salt...	Which fish would survive in salt water?	0
...
404285	404285	433578	379845	How many keywords are there in the Racket prog...	How many keywords are there in PERL Programmin...	0
404286	404286	18840	155606	Do you believe there is life after death?	Is it true that there is life after death?	1
404287	404287	537928	537929	What is one coin?	What's this coin?	0
404288	404288	537930	537931	What is the approx annual cost of living while...	I am having little hairfall problem but I want...	0
404289	404289	537932	537933	What is like to have sex with cousin?	What is it like to have sex with your cousin?	0

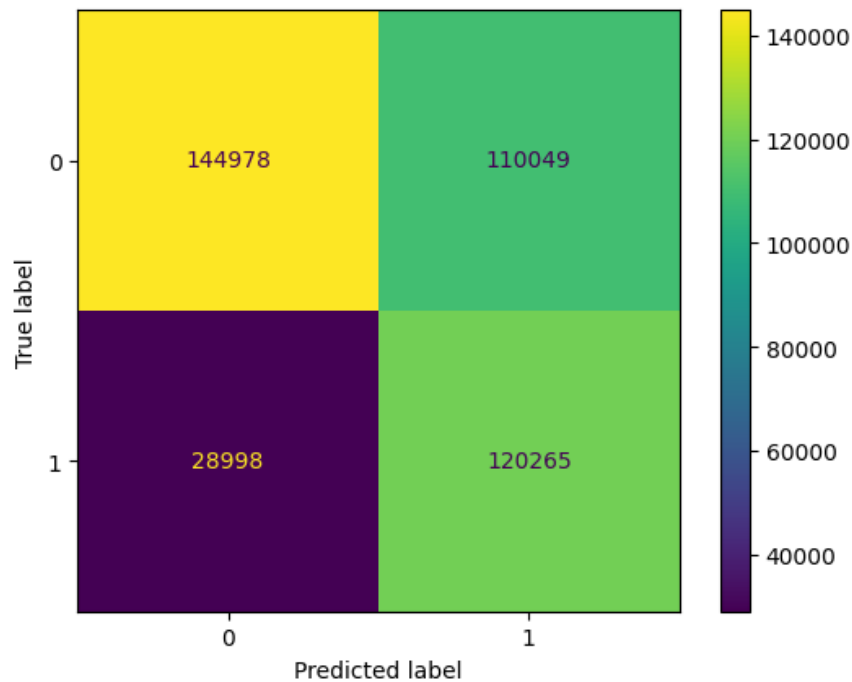
Rys. 1 – ramka danych pandas utworzona z pobranego pliku

Dane trenujące posiadają kolumny: id porządkowe, id pytań, treści pytań oraz informacje czy pytania dotyczą tego samego. Można tutaj zauważyć, że id ostatniego pytania to 537 933, a wierszy w bazie jest 404 289, więc znaczna część pytań się powtarza.

Pierwszym etapem w projekcie była zamiana tekstu pytań na formę liczbową – możliwą do interpretacji przez komputer. Utworzyliśmy korpus zawierający wszystkie pytania z bazy bez duplikatów. Następnie, utworzyliśmy wektory TFIDF. Ze względów wydajnościowych ich długość została ograniczona do 5 000 elementów.

Pytanie, które sobie zadaliśmy to, czy w procesie tokenizacji zastosować słowa stopu (ang. *stop words*). Przeglądając bazę zobaczyliśmy wiele krótkich pytań, które nie zawsze znaczą to samo (na przykład “What is one coin?” i “What’s this coin?”). W ich przypadku użycie wyrazów stopu spowodowałoby redukcję większości słów do jednego czy dwóch, które byłyby te same dla obu pytań. W rezultacie model, który by się uczył na tych danych mógłby założyć, że zawsze te same wektory oznaczają to samo (i byłoby to dobre założenie, lecz wyniki byłyby błędne), lub wektory, które są te same zawsze oznaczają inny sens (byłoby to błędne założenie, lecz mogłoby się sprawdzić dla tych konkretnych danych). Niezależnie od rezultatu takie przygotowanie danych byłoby nieodpowiednie, stąd decyzja o zrezygnowaniu ze stopu. Cały skrypt, który przygotowuje wektory TFIDF umieściliśmy w pliku `prepareCorpus.py`.

Pierwszym krokiem było znalezienie podobieństwa cosinusowego, aby sprawdzić jak ich miara podobieństwa ma się do informacji z danych trenujących (is_duplicate). Dokładność miary to około 66%, co nie jest zbyt dobrym wynikiem. Oto macierz pomyłek dla miary podobieństwa cosinusowego:



Rys. 2 – macierz pomyłek dla podobieństwa cosinusowego

Na wykresie możemy zobaczyć, że najwięcej pomyłek jest w sytuacjach, gdzie różne pytania są złożone z podobnych słów, a podobieństwo cosinusowe zalicza to jako to samo znaczenie (prawa górna część rysunku).

3. Modele biblioteczne

Przetestowaliśmy różne modele do klasyfikacji tekstu z różnymi parametrami, aby sprawdzić, który z nich najlepiej sobie poradzi z danymi. Trenowane modele to:

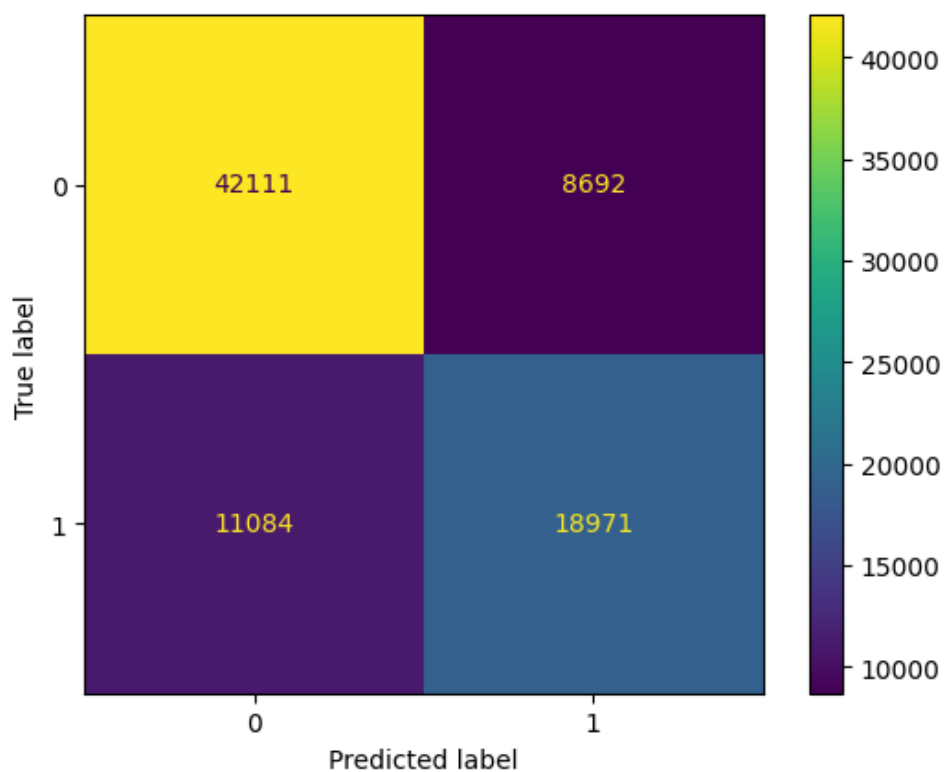
- MultinomialNB [6] – klasyfikator oparty na algorytmie Bayesa. Nadaje się do klasyfikacji teksów, dobrze sobie radzi z wektorami TFIDF.
- SGDClassifier [7] – klasyfikator oparty na algorytmie stochastycznego spadku gradientu, szeroko stosowany do klasyfikacji tekstu. Stosuje on algorytm spadku gradientu do optymalizacji funkcji straty.
- LinearSVC [8] – klasyfikator, który próbuje znaleźć taką hiperpłaszczyznę, która separuje dane. Jest to inna wersja bazowego klasyfikatora SVC z liniową funkcją straty (przedrostek linear). Próbowaliśmy również użyć zwykłego klasyfikatora SVC, lecz nie jest on przeznaczony do pracy na dużych danych. Czas jego nauki był nieakceptowalnie długi więc zastąpiliśmy go jego wersją liniową.
- LGBMClassifier [9] – (Light Gradient Boosting Machine Classifier) - klasyfikator oparty na technologii gradient boosting. Buduje on silne modele predykcyjne, dodając do siebie słabe modele w procesie boostingu.

Podzieliliśmy dane wejściowe na dane trenujące oraz walidacyjne. Przetrenowaliśmy te modele również na różnych wersjach wektorów TFIDF (z zastosowaniem lematyzacji, stemmingu i obu tych technik). Wyniki dokładności (na danych walidacyjnych) prezentuje poniższa tabela:

Model name	TfIdf - Lemmatization	TfIdf - Stemming	TfIdf - L + S
MultinomialNB	74,13%	73,76%	73,75%
SGDClassifier	74,32%	74,19%	74,26%
LinearSVC	75,54%	75,38%	75,43%
LGBMClassifier	75,31%	75,38%	75,44%

Rys. 3 – porównanie dokładności predykcji dla modeli bibliotecznych

Z tabeli możemy zobaczyć, że modele uczą się na podobnym poziomie. Najlepiej poradził sobie model LinearSVC. Poniżej macierz pomyłek dla tego modelu:



Rys. 4 – macierz pomyłek dla modelu LinearSVC

Na rysunku można zaobserwować, że model dosyć dobrze poradził sobie z pytaniami, które używają podobnych słów a mają inne znaczenie.

4. Własny model głębokiej sieci neuronowej

Osiągnięta dokładność 75,6% przez model LinearSVC to akceptowalny wynik, lecz chcieliśmy spróbować stworzyć własny model za pomocą pakietu **Keras** aby spróbować osiągnąć wyższą dokładność. Zastosowano model, w którym wektory TFIDF oby pytań są łączone w jeden i podowane na wejście sieci neuronowej o kilku warstwach ukrytych.

Dobieraliśmy różne warstwy oraz różną liczbę neuronów w warstwach, aby przetestować, jak zachowują się różne modele oraz jakie warstwy powinny być wstawione, aby uzyskać jak największą dokładność. Ostatecznie struktura modelu prezentuje się w następujący sposób:

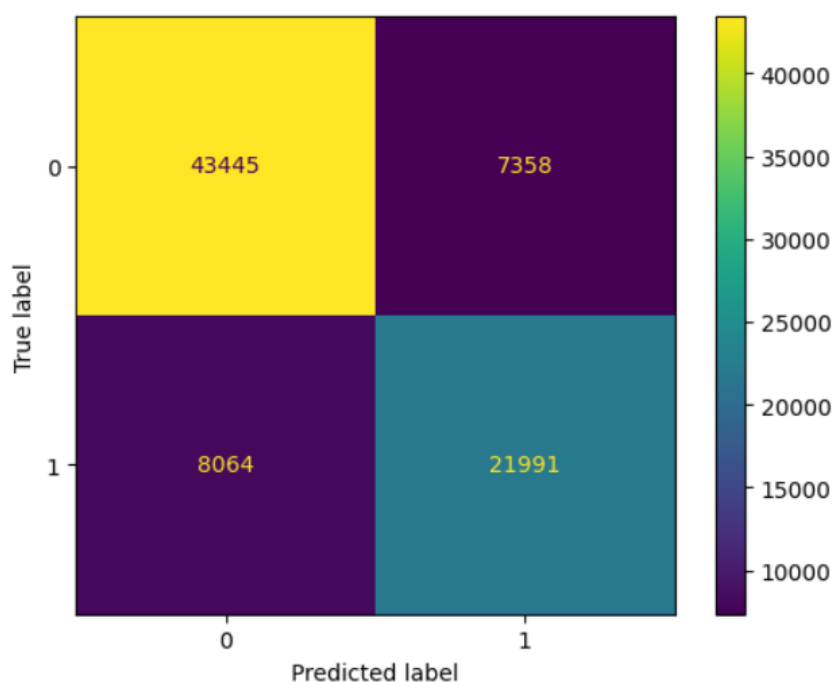
```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 1024)	10241024
dense_1 (Dense)	(None, 32)	32800
dropout (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 32)	1056
dropout_1 (Dropout)	(None, 32)	0
dense_3 (Dense)	(None, 32)	1056
dropout_2 (Dropout)	(None, 32)	0
dense_4 (Dense)	(None, 32)	1056
dense_5 (Dense)	(None, 1)	33

```
Total params: 10277025 (39.20 MB)
Trainable params: 10277025 (39.20 MB)
Non-trainable params: 0 (0.00 Byte)
```

Rys. 5 – architektura własnego modelu w Keras

Gotowy model osiągnął dokładność 80% dla danych walidacyjnych:



Rys. 6 – macierz pomyłek dla własnego modelu

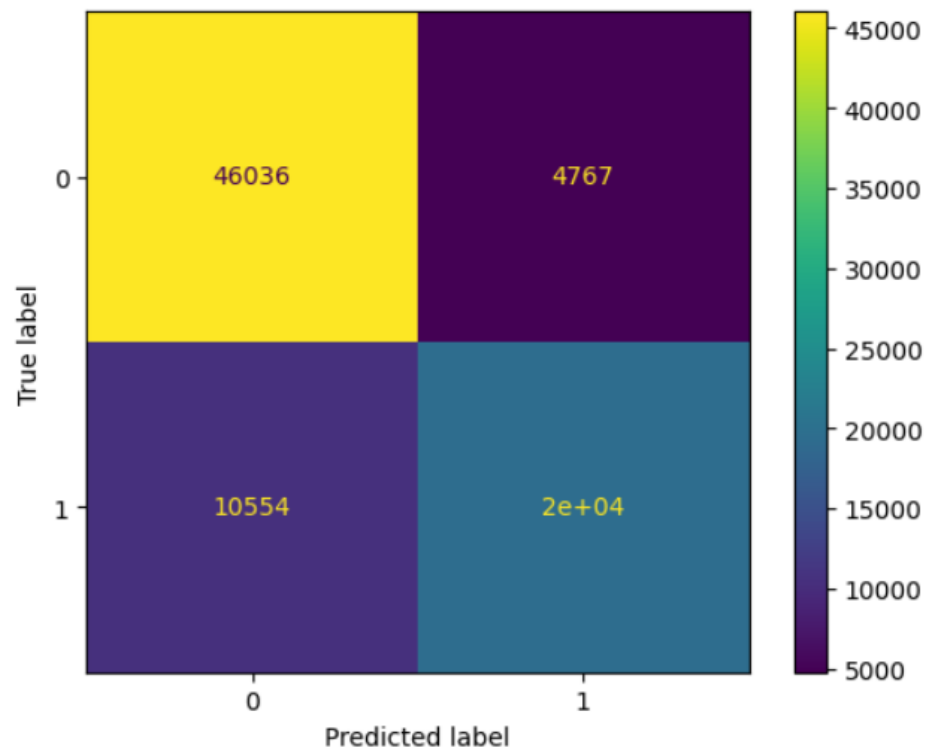
Jak możemy zobaczyć model osiągnął nieco lepsze wyniki niż klasyfikatory dostępne w pakiecie scikit-learn.

Wadą powyższego rozwiązania jest bez wątpienia fakt, że model musiał sam znaleźć granicę między wektorami dwóch pytań. Alternatywnym podejściem może być zastosowanie architektury sieci składającej się z dwóch gałęzi, z których każda przetwarza osobno swoje pytanie i wyszukuje istotnych cech, a następnie z ich połączenia wnioskowane jest podobieństwo.



Rys. 7 – schemat architektury modelu z dwiema gałęziami

Przetestowany model osiągnął nieznacznie lepsze wyniki (dokładność dla danych walidacyjnych na poziomie 81%).



Rys. 8 – macierz pomyłek dla modelu z dwiema gałęziami

5. Użycie modelu word2vec

Ostatnim etapem projektu było sprawdzenie działania własnego modelu uczenia maszynowego na innej reprezentacji danych niż wektory TFIDF, mianowicie po przetworzeniu zdań z korpusu za pomocą sieci word2vec:

```
tokenized_questions = [custom_tokenize(str(question).lower()) for question in df['question1'] + df['question2']]
vector_size = 100
window_size = 5

model = Word2Vec(sentences=tokenized_questions, vector_size=vector_size, window=window_size, min_count=1, workers=4)

def pad_vector(vector, target_length):
    if not vector:
        return [[0] * vector_size] * target_length
    elif len(vector) < target_length:
        return vector + [[0] * len(vector[0])] * (target_length - len(vector))
    elif len(vector) > target_length:
        return vector[:target_length]
    else:
        return vector

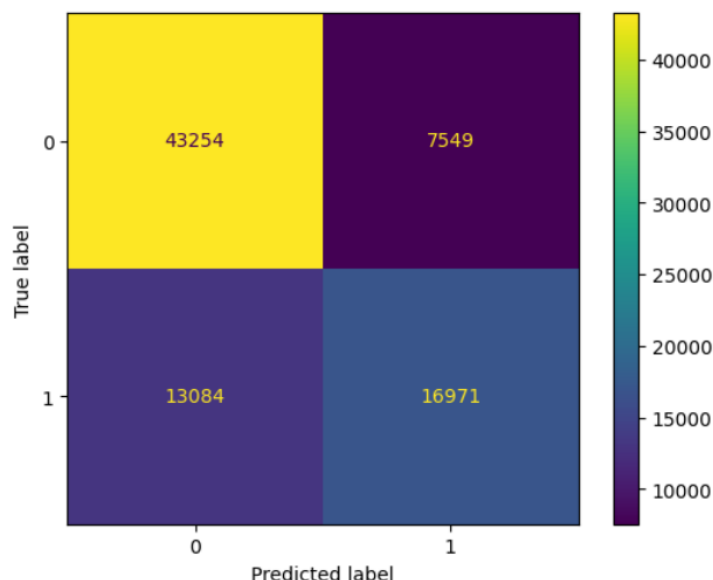
def question_to_vector(question, model, max_length):
    tokens = word_tokenize(str(question).lower())
    vector = [model.wv[word] for word in tokens if word in model.wv]
    padded_vector = pad_vector(vector, target_length=max_length)
    return padded_vector

MAX_LENGTH = 15
df['question1'] = df['question1'].apply(lambda x: question_to_vector(x, model, MAX_LENGTH))
df['question2'] = df['question2'].apply(lambda x: question_to_vector(x, model, MAX_LENGTH))
df = df[['question1', 'question2', 'is_duplicate']]
```

Rys. 9 – wykorzystanie modelu word2vec do otrzymania wektorów słów

Ze względów wydajnościowych zdecydowano się na ograniczenie do stu elementowych wektorów dla piętnastu słów w pytaniu (w praktyce rzadko pytania z korpusu były dłuższe).

Model uzyskał nieco gorszy wynik niż dla wektorów TFIDF (skuteczność dla danych walidacyjnych osiągnęła poziom 74 %):



Rys. 10 – macierz pomyłek dla modelu wykorzystującego wektory słów uzyskane z word2vec

6. Wykorzystanie modelu w praktyce

Utworzyliśmy skrypt w pythonie, aby użytkownik mógł wprowadzić własne dwa pytania oraz zobaczyć informację zwrotną od modelu czy pytania dotyczą tego samego sensu.

```
while(True):
    q1 = input()
    q2 = input()
    if len(q1) == 0 or len(q2) == 0:
        break
    print(q1)
    print(q2)
    inp = hstack([vectorizer.transform([q1]), vectorizer.transform([q2])])
    ans = model.predict(vstack([inp]))[0]
    print('> Yes' if ans else '> No')
```

Oto krótkie pytania, które zadawaliśmy skryptowi oraz odpowiedzi na nie:

How are you feel?	How are you?	Yes
What's this coin?	What is color of this coin?	No
Are you good with math?	Are you a mathematician?	No
Have you a computer mouse?	Have you a pet mouse at home?	No
Have you computer mouse?	Do you have a mouse to your computer?	Yes

7. Podsumowanie

W ramach projektu z powodzeniem przetestowano modele uczenia maszynowego w problemie semantycznego porównywania podobieństw pytań wyrażonych w języku naturalnym. Biblioteczne modele oparte SI oparte o wektory TFIDF uzyskały zadawalającą dokładność na poziomie 75%. Własne modele wykorzystujące uczenie głębokie poprawiły wynik do ok. 80%. W czasie prac kluczową kwestią okazał się etap wstępnego przetwarzania danych. Na podstawie tych doświadczeń, a także analizy rozwiązań opublikowanych w serwisie Kaggle uzasadnionym wydaje się twierdzenie, że dalsza poprawa wydajności powinna skupiać się przede wszystkim na etapie tokenizacji i stemmingu.

Wektory słów uzyskane modelem word2vec nie spełniły oczekiwań, a testowane modele uczenia maszynowego okazały się dla nich mniej skuteczne. Należy jednak mieć na uwadze, że tworzenie wektorów z wykorzystaniem modelu word2vec wymagało większej mocy obliczeniowej, a ich przechowywanie większej przestrzeni w pamięci w porównaniu do, z natury rzadkich, wektorów TFIDF, dlatego konieczne było bardziej rygorystyczne ograniczenie ich rozmiarów.

Projekt pozwolił na dogłębne zrozumienie podstawowych zagadnień związanych z tematyką przetwarzania języka naturalnego: lematyzacji, stemmingu czy wektorów TFIDF.

Bibliografia

- [1] <https://www.kaggle.com/c/quora-question-pairs/overview>
- [2] <https://www.nltk.org/>
- [3] <https://scikit-learn.org/>
- [4] <https://scipy.org/>
- [5] <https://keras.io/>
- [6] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
- [7] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
- [8] <https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html>
- [9] <https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html>