Quora Question Pairs

Lenart Piotr

Mizera Damian

Spis treści

Wstęp

Celem projektu jest przygotowanie programu, który będzie identyfikować podobne teksty na bazie danych opisanych w problemie na portalu Kaggle: https://www.kaggle.com/c/quora-question-pairs/overview.

Portal Kaggle jest popularnym serwisem internetowym, na którym użytkownicy mogą zmierzyć się z zadaniami konkursowymi opartych na statystyce i przetwarzaniu danych. Jednym z tych problemów jest zadanie Quora Question Pairs, gdzie celem jest stworzenie modelu, który będzie w stanie stwierdzić czy dwa pytania pytają o to samo (nie chodzi o słowa, lecz znaczenie pytań - ich sens). Do tego zadania jest dostarczona baza pytań do nauki, gdzie podane są pytania oraz stwierdzone jest czy ich sens jest taki sam czy inny.

W tym raporcie przedstawimy nasze podejście do tematu, nasze programy oraz modele i ich skuteczność. Do realizacji tematu użyliśmy języka python z bibliotekami do analizy tekstu oraz tworzeniem modeli AI, takich jak: *nltk*, *sklearn*, *scipy*, *keras.*

Przygotowanie danych

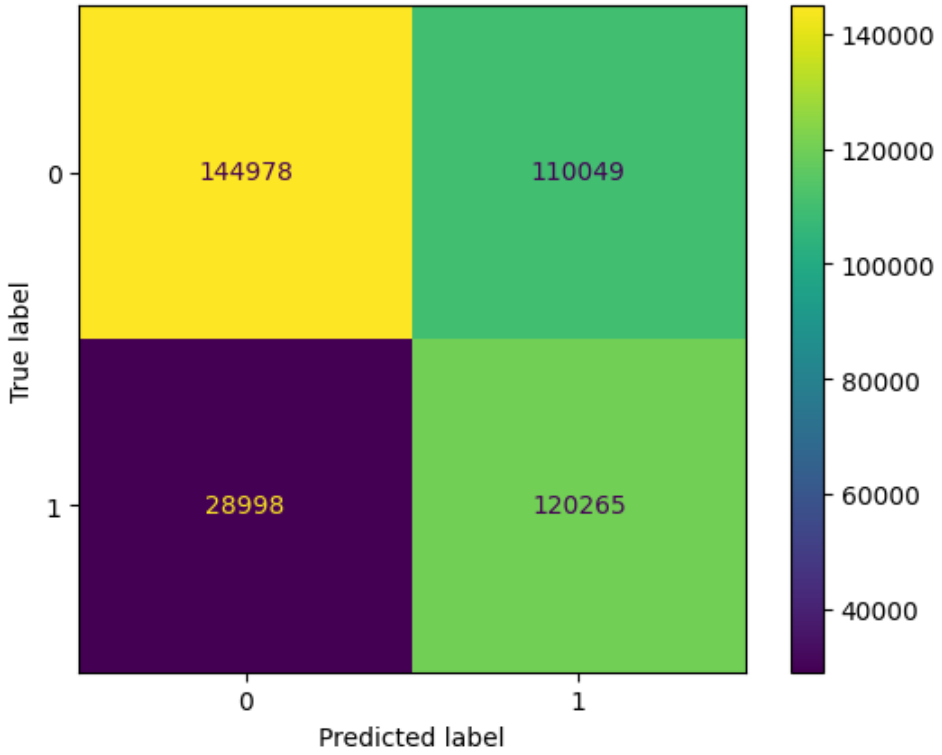
Dane csv po wczytaniu do programu prezentują się w następujący sposób:

	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 [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly 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

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 tym projekcie była zamiana pytań na liczby. Utworzyliśmy korpus zawierający wszystkie pytania z bazy bez duplikatów następnie utworzyliśmy wektory TFIDF. Pytanie, które sobie zadaliśmy to, czy w tokenizacji używać stemingu. Przeglądając bazę zobaczyliśmy wiele krótkich pytań, na przykład “What is one coin?” i “What’s this coin?”, które nie zawsze znaczą to samo. W ich przypadku użycie steamingu spowodowałoby ucięcie większości słów do jednego czy dwóch, które są 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 podejścia te dane byłby źle przygotowane, więc zrezygnowaliśmy ze stemmingu. Cały skrypt, który przygotowuje wektory umieściliśmy w pliku prepareCorpus.py.

Mając wektory pierwsze co zrobiliśmy to podobieństwo cosinusowe, 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 prawd i fałszu dla miary 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).

Testowanie modeli

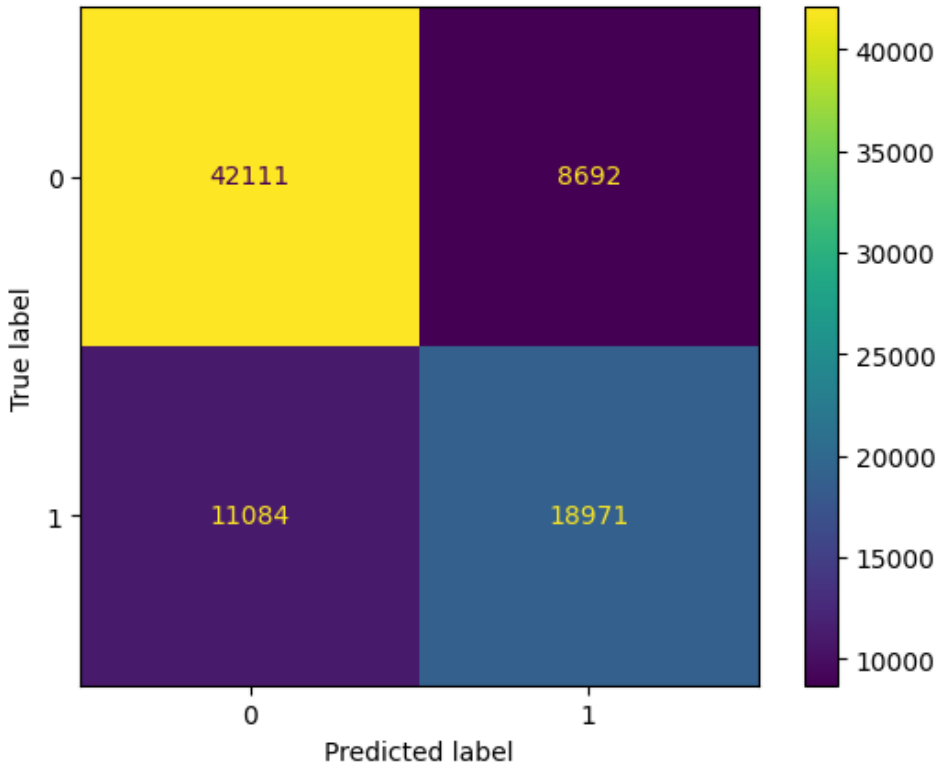
Przetestowaliśmy różne modele do klasyfikacji tekstu z różnymi parametrami, aby sprawdzić, który z nich najlepiej sobie poradzi z danymi. Trenowane modele to:

- MultinomialNB – jest to klasyfikator oparty na algorytmie Bayesa. Nadaje się do klasyfikacji teksów, dobrze sobie radzi z wektorami TFIDF.
- SGDClassifier – klasyfikator oparty na algorytmie stochastycznego spadku gradientu, szeroko stosowany do klasyfikacji tekstu. Stosuje on algorytm spadku gradientu do optymalizacji funkcji straty.
- LinearSVC – jest to 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. Jego nauka szła bardzo wolno więc zastąpiliśmy go jego wersją liniową.
- LGBMClassifier - (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 lematyzacją, z stemmingiem i z oboma). 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%

Z tabeli możemy zobaczyć, że modele uczą się na podobnym poziomie. Użycie stemmingu pogarsza wyniki, lecz w stopniu mniejszym niż przypuszczaliśmy na początku. Najlepiej poradził sobie model LinearSVC – oto tabela prawdy i fałszu dla tego modelu:



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

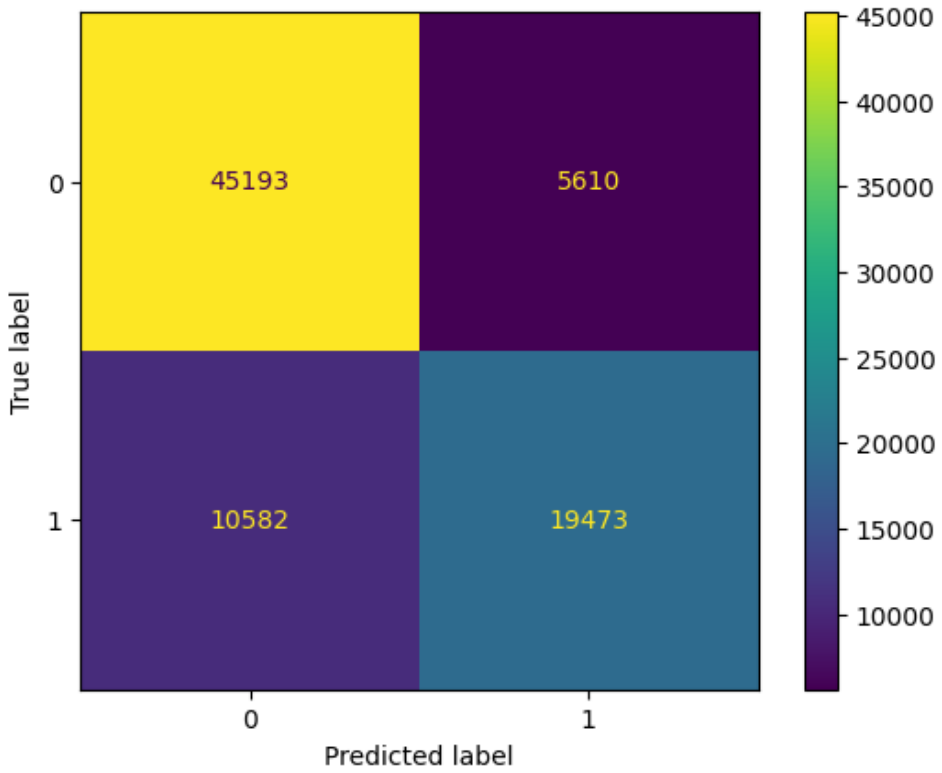
Własny model AI

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.

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

```
model = Sequential()
model.add(Dense(units=1024, input_dim=10000, activation='relu'))
model.add(Dense(units=1024, activation='relu'))
model.add(Dense(units=1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=1024, activation='relu'))
model.add(Dense(units=1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=1024, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))
```

Gotowy model osiągnął dokładność 80%. Oto macierz prawdy i fałszu:



Jak możemy zobaczyć model osiągnął nieco lepsze wyniki niż klasyfikatory dostępne w pakietach.

Modele 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

Źródła

https://www.kaggle.com/c/quora-question-pairs/overview

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html