Predykcja zainteresowania postami w social media z użyciem metod NLP

Projekt zaliczeniowy

Jakub Uchman

2 czerwca 2024

Wstęp

Celem tego projektu była predykcja zainteresowania postami w mediach społecznościowych z użyciem metod NLP (Natural Language Processing). Dokładniej, wykorzystałem zestaw danych „Trump Tweets" udostępniony w serwisie Kaggle. Jako miarę zainteresowania wykorzystałem liczbę retweet-ów. Jednym z dodatkowych założeń było oparcie modelu jedynie na tekście samego tweet'a, bez meta-informacji takich jak data tweet'a czy liczba polubień tak aby model był jak najlepiej dostosowany do przewidywania kolejnych wpisów.

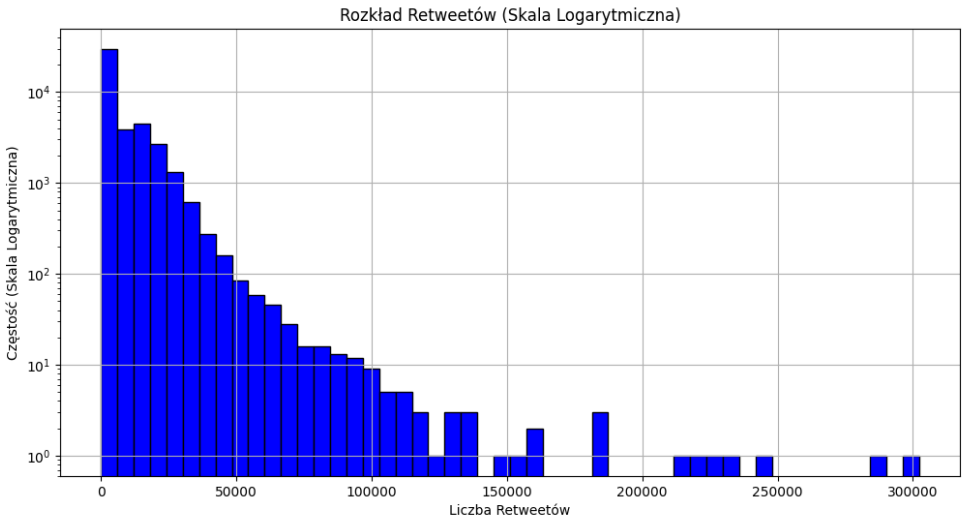
Przed przejściem dalej zerknijmy trochę na to jak wygląda nasz dataset

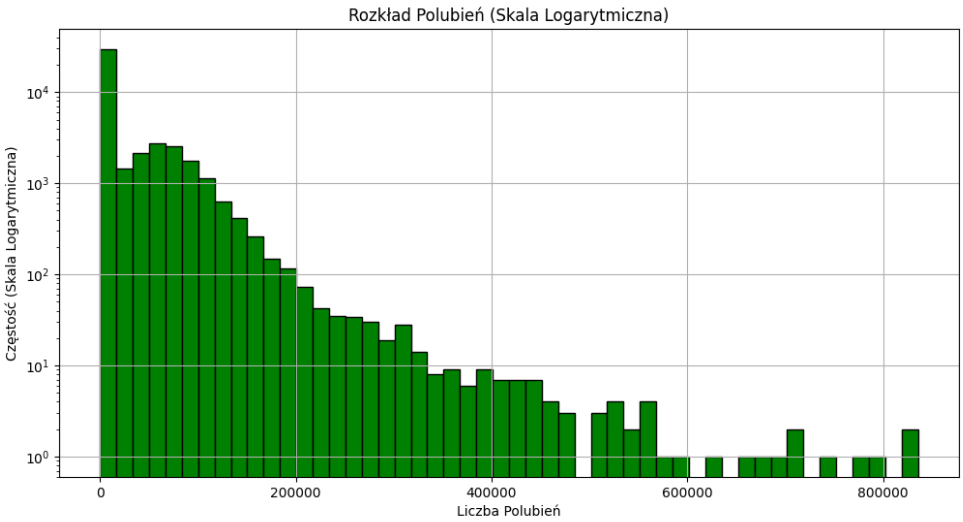
Zbiór danych

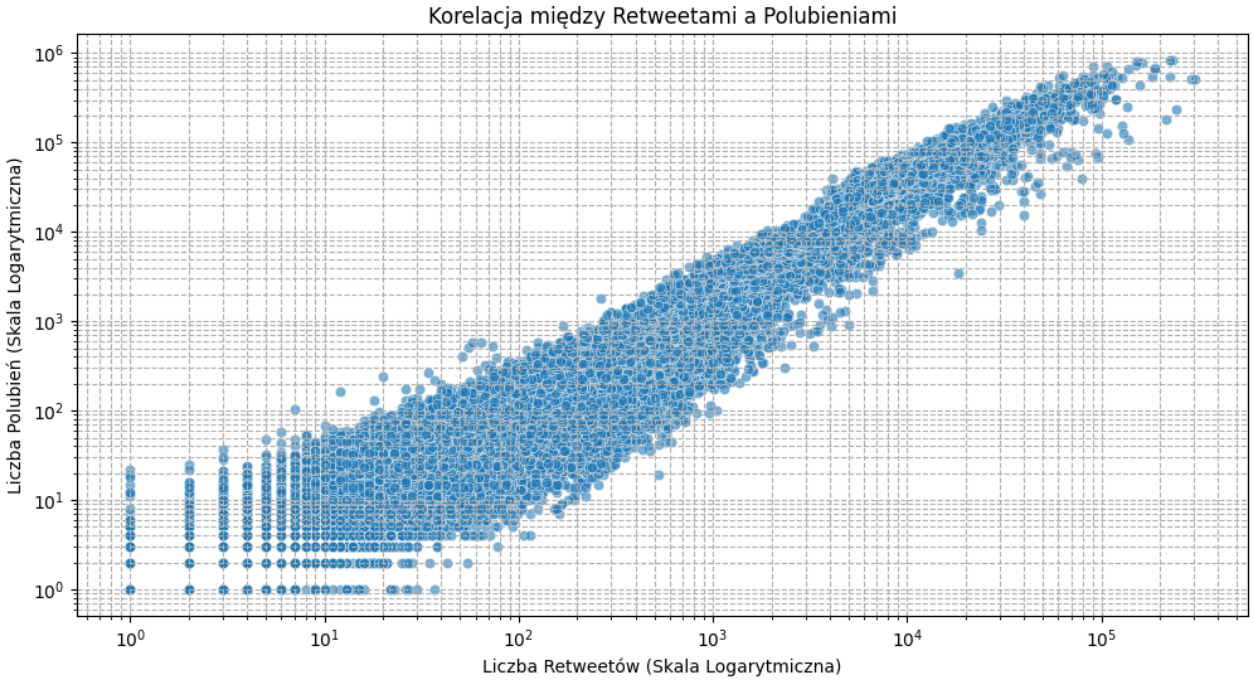
Moje źródło danych to plik .csv zawierający 43352 rekordów gdzie każdy rekord to wpis na Twitterze Donalda Trump'a, format pliku jest następujący:

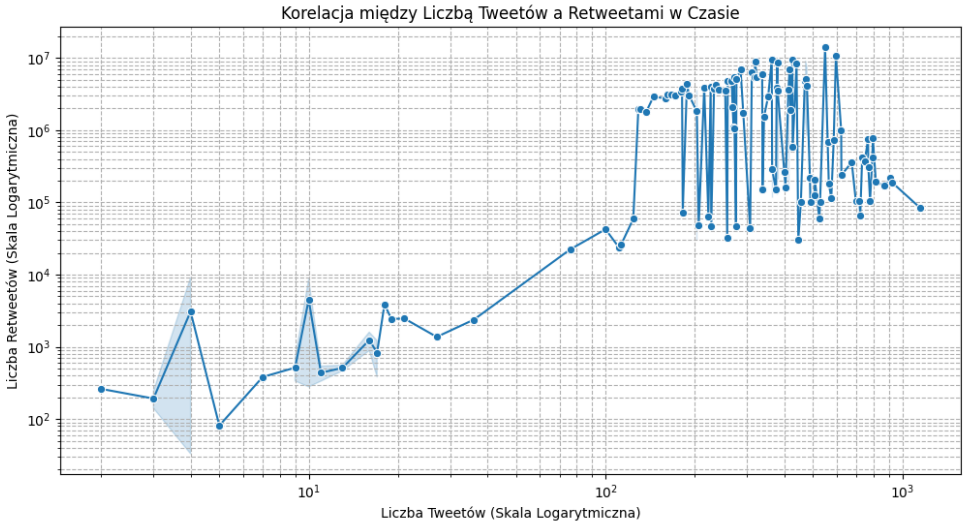
id,link,content,date,retweets,favorites,mentions,hashtags

Zobaczymy kilka wizualizacji które pomogą nam zrozumień lepiej ten zbiór.




Rozkład Polubień (Skala Logarytmiczna)
Częstość (Skala Logarytmiczna)
Liczba Polubień
0
200000
400000
600000
800000


Korelacja między Retweetami a Polubieniami
Liczba Polubień (Skala Logarytmiczna)
Liczba Retweetów (Skala Logarytmiczna)



Dwa najmniej popularne tweety:

```
ID: 247769829384269824
Link: https://twitter.com/realDonaldTrump/status/247769829384269824
Treść: @ bodysouls @ digitaljournal Great statement.
Data: 2012-09-17 13:51:45
Retweety: 0
Polubienia: 0


ID: 247774447333949440
Link: https://twitter.com/realDonaldTrump/status/247774447333949440
Treść: @ Travis_George Thanks, but I have a lot of businesses to run which includes creating jobs.
Data: 2012-09-17 14:10:06
Retweety: 0
Polubienia: 2
```

Dwa najbardziej popularne tweety:

```
ID: 881503147168071680
Link: https://twitter.com/realDonaldTrump/status/881503147168071680
Treść: # FraudNewsCNN # FNNpic.twitter.com/WYUnHjjUjg
Data: 2017-07-02 08:21:42
Retweety: 302269
Polubienia: 515729


ID: 795954831718498305
Link: https://twitter.com/realDonaldTrump/status/795954831718498305
Treść: TODAY WE MAKE AMERICA GREAT AGAIN!
Data: 2016-11-08 05:43:14
Retweety: 289872
Polubienia: 508758
```

Aby zacząć dokładniej opisywać moją drogę do realizacji tego projektu, warto zapoznać się z tym, czym w ogóle jest NLP i w jaki sposób możemy przygotować dane wejściowe dla takiego modelu. Wiemy przecież że modele uczenia maszynowego bazują na danych liczbowych, a tekst jest od nich bardzo różny, musimy więc opracować jakiś sposób konwersji całych zdań do reprezentacji liczbowej akceptowalnej przez model.

Wstępne przetwarzanie tekstu

Temat przetwarzania wstępnego tekstu dla modeli NLP jest bardzo dobrze znany i studiowany. Nie wystarczy zamienić tekstu na dowolną reprezentację liczbową, aby model mógł osiągnąć najlepsze możliwe wyniki. Transformacja ta powinna być przemyślana i dostosowana do specyficznych cech charakterystycznych uczenia maszynowego. Podstawą pre processiangu jest oczyszczenie tekstu z informacji mało znaczących które możemy zaklasyfikować jako swego rodzaju szum, staramy się go wyeliminować, a następnie oczyszczony tekst odpowiednio ztokenizować i zamienić na reprezentację liczbową.

Poniżej przedstawiam na jakie kroki pre processiangu zdecydowałem się przygotowując swój model.

Wstępne przetwarzanie tekstu będzie miało na celu tokenizacje treści tweeta, czyli podzielenie jednego ciągu znakowego na wiele tokenów, które będą przekazane do modelu.

Oto jak to wygląda to w moim modelu:

1. Tweet wejściowy:

„@ RickSantorum is 100% losing his home state of Pennsylvania to @ MittRomney http://bit.ly/GTlJyP @ RickSantorum is only (cont) http://tl.gd/gmbpae"

2. Usuwanie linków:

„@ RickSantorum is 100% losing his home state of Pennsylvania to @ MittRomney @ RickSantorum is only (cont)"

3. Usuwanie znaków niebędących częścią słów ani liczbami:

„RickSantorum is 100 losing his home state of Pennsylvania to MittRomney RickSantorum is only cont"

4. Usuwanie liczb:

„RickSantorum is losing his home state of Pennsylvania to MittRomney RickSantorum is only cont"

5. Tokenizacja:

['ricksantorum', 'is', 'losing', 'his', 'home', 'state', 'of', 'pennsylvania', 'to', 'mittromney', 'ricksantorum', 'is', 'only', 'cont']

6. Lematyzacja i usuwanie stop-words

Teraz gdy podzieliliśmy nasz oczyszczony tweet na tokeny, które w naszym przypadku są słowami (choć nie muszą nimi być), nadszedł czas na etap, który technicznie nie jest wymagany, ale który może znacząco zwiększyć zdolności modelu do generalizacji przy jednoczesnym zmniejszeniu wymiarowości danych wejściowych. Mowa tutaj o lematyzaacji i usuwaniu tzw. Stop-words. W moim kodzie oba te procesy realizowane są w jednym kroku, ale można robić to oddzielnie)

Zacznijmy od Lematyzacji:

Lematyzacja polega na sprowadzaniu słowa do jego podstawowej formy którą nazywamy właśnie „lemem" (inaczej hasłem). Dzięki temu słowa takie jak „running", „ran" czy „better", „best" zostają sprowadzone do formy podstawowej („running", „ran") -> „run" („better", „best") -> „good" (wykorzystując słownik) dzięki temu zmniejsza nam się różnorodność słów, co za tym idzie - liczba unikalnych tokenów. Dzięki temu model uczy się szybciej i lepiej „rozumie" kontekst gramatyczny.

Innym podobnym podejściem zastępującym lematyzację jest stemming czyli jej bardziej prymitywny odpowiednik. Usuwamy w nim końcówki fleksyjne słów pozostawiając sam rdzeń np. „running" -> „run" ale w przeciwieństwie do lematyzaacji „better" -> „better" oraz „ran" -> „ran", zalety to oczywiście łatwiejsza i szybsza implementacja, ale niesie to ze sobą także dosyć spore wady. Jest to metoda mniej precyzyjna (jak widzimy na przykładzie „better") oraz mogąca prowadzić do powstawania nieistniejących słów.

A co gdyby nie stosować ani lematyzaacji ani stemmingu?

Model musiałby radzić sobie z wieloma dodatkowymi tokenami, co zmniejszyłby jego zdolność do generalizacji - każda forma tego samego słowa byłaby dla niego kompletnie nowym tokenem. W rezultacie model dostosowywałby się do bardzo specyficznych danych i nie potrafiłby dobrze generalizować bo nawet inna forma czasownika mogłaby go bardzo łatwo „zdezorientować".

Napisałem wyżej że oprócz lematyzaacji, mój model dokonuje dodatkowo usuwania tzw. „Stop-words" co to takiego?

Stop-words to często występujące słowa takie jak „is”, „of”, „to” itd. które nie niosą istotnej informacji dla analizy tekstu przez model a przy tym niepotrzebnie zwiększają one złożoność danych wejściowych. Usunięcie stop-words pomaga w redukcji tokenów a co za tym idzie zwiększa efektywność modelu, pozwalając mu się skupić na prawdziwie ważnych tokenach.

Tweet po lematyzaacji i usuwaniu stop-words:
['ricksantorum', 'lose', 'home', 'state', 'pennsylvania', 'mittromney', 'ricksantorum', ,cont']

Tak oto dostajemy w pełni oczyszczony i ztokenizowany tekst. Teraz każdy token może zostać przekształcony na reprezentacje liczbową i przekazany do modelu. Ale o tym w kolejnej części.

Dane wejściowe dla modelu

Teraz kiedy nasz tweet jest już wstępnie przetworzony możemy przejść do przygotowania ostatecznych danych które trafią już bezpośrednio do modelu. W trakcie pracy nad tym projektem testowałem różne architektury modelu, i wiele różnych konfiguracji danych wejściowych. Głównym założeniem było to że model powinien być przydatny w realnych warunkach, do przewidywania zainteresowania przyszłymi tweet’ami, dlatego nie powinniśmy stosować w danych treningów meta informacji które dostępna są w naszym pliku .csv ale których nie moglibyśmy użyć w realnej sytuacji, mówię tutaj w szczególności o liczbie polubień (Chociaż w jednym z moich podejść w celach testowych zastosowałem te dane). Pierwotnie starałam polegać jedynie na odpowiednio przetworzonym tekście (tym uzyskanym w poprzedniej sekcji) jako wejściem do

modelu, jednak uznałem uzyskiwane wyniki za niewstarczające, niezależnie od architektury modelu model wydawał się uderzać w ścianę nie do przekroczenia.Wiedziałem więc że problemem nie jest sama architektura modelu ale same dane nie pozwalały lepiej przewidywać zainteresowania. Wtedy postanowiłem dopuścić nazwijmy to meta-danymi ale pod warunkiem że da się je uzyskać jedynie z treści tweet'a i tak dodałem następujące dane:

1. ***Liczba słów napisana caps-lockiem*** - W danych źródłowych takie słowa zdarzały się dosyć często i niosą one specjalny ładunek emocjonalny który w mojej opinii może wpływać na liczbę retweetów poprzez oddziaływanie na czytelników, gdybym nie dodał tych danych ta potencjalnie cenna informacja zostałaby utracona ponieważ we wstępnym przetwarzaniu danych na pewnym etapie zamieniamy wszystkie litery na małe. Samą liczbę słów zliczamy na tekście nieoczyszczonym.
2. ***Liczba hashtagów*** - csv oferuje kolumnę w której trzymane są hashtag które zostały wstawione w treści tweeta, a nawet gdybyśmy nie mieli takiego pola wyodrębnienie liczby hashtagów z treści jest bardzo łatwe. Postanowiłem dodać tą liczbę jako dane dodatkowe ponieważ łatwo można ją wyodrębnić z treści samego tweeta, a także z własnych obserwacji zestawu danych w dużej części przypadków tweety z dużą liczbą hashtagów osiągają słabsze wyniki bo nie są one zazwyczaj emocjonujące.
3. ***Liczba wzmianek*** - To także wynika po części z mojej intuicji, tweety z dużą liczbą wzmianek często są odpowiedziami do kogoś, gdzie kogoś się oznacza żeby przekazać mu jakieś informacje tweety takie są bardziej komentarzami do konkretnej osoby, lub są informacyjne. Tweety bez wzmianek są częściej oświadczeniami, albo przemyśleniami i one częściej

bardziej angażują użytkowników, z tego powodu postanowiłem wyodrębnić to jako osobny feature.

4. ***Liczba słów i liczba znaków*** - te dane dodałem bo są łatwe do ekstrakcji z tekstu tweet i mogą korelować z liczbą retweet-ów

Kiedy wypisane przeze mnie powyżej dodatkowe cechy zostały już stworzone, przyszedł czas aby nasze tablice z tokenami zamienić na wektory. Jest wiele różnych metod które są w stanie to zrobić, ja dla swojego modelu wybrałem tzw TF-IDF (Term Frequency-Inverse Document Frequency) Jest to dosyć prosta ale porównam ją do jeszcze prostszej BoW (Bag of Words)

TF-IDF:

Metoda ta uwzględnia częstość występowania słów w całym zbiorze danych, oraz ich rzadkość, co pozwala lepiej ocenić ich ważność w tweet'ach.

Jak to działa?

TF(Term Frequency): Jest miarą określającą jak często dane słowo pojawia się w konkretnym Tweecie. Obliczamy ją poprzez podzielenie liczby wystąpień danego słowa w Tweecie przez całkowitą liczbę słów w tym Tweecie. Służy to zrozumieniu wagi danego słowa w kontekście konkretnego tweeta.

IDF(Inverse Document Frequency) Z kolei pomaga ocenić jak ważne jest dane słowo w kontekście całego zbioru danych, obliczamy ją jako logarytm(zazwyczaj naturalny) stosunku całkowitej liczby tweetów w naszym zbiorze danych do liczby tweetów w których dane słowo się pojawia. Co ciekawe ze względu na charakterystykę tego algorytmu „Stop-words" które wcześniej usuwałem otrzymałyby bardzo niskie wartości IDF i co za tym idzie zmniejszyłby się ich wpływ na reprezentację wektorową.

$$TFIDF(t, d, D) = TF(t, d) \times IDF(t, D)$$

t *- słowo*

d *- tweet*

D *- Zbiór tweetów*

TFIDF to ostateczny wektor który jest wejściem dla naszego modelu, chociaż w naszej rozbudowanej wersji jako wejście dodajemy jeszcze dodatkowy wektor z cechami dodatkowymi które wcześniej wyodrębniliśmy.

Zalety:

- Redukcja wpływu słów powszechnych.
- Podkreślenie słów unikatowych

Wady:

- Nie uwzględnia semantyki
- Mniej skuteczne dla krótkich dokumentów (Dlatego dosyć poważnie wahałem czy ją zastosowaćj)

Innym prostszym podejściem byłoby zastosowanie ***BoW***, które tworzy wektor cech na podstawie częstości występowania słów w dokumencie podobnie jak w przypadku TF-IDF wymiar wektora zależy od liczby unikalnych słów użytych we wszystkich tweetach.

Załóżmy że mamy trzy tweety:

„The cat in the hat”,

„The cat is in the hat”,

„The hat”

Po segmentacji mamy:

["the", "cat", "in", "hat", „is"] itp.

A po zastosowaniu BoW

"the cat in the hat" -> [1, 1, 1, 1, 0]

"the cat is in the hat" -> [1, 1, 1, 1, 1]

"the hat" -> [1, 0, 0, 1,]

Zauważmy że długość wektora zależy od całkowitej liczby słów w całym zestawie danych. Na pozycjach gdzie w danym tekście nie ma danego słowa pojawia się zero. Gdzie słowo pojawia się więcej razy jest liczba większa niż 1. w naszym rozwiązaniu TF-IDF działa bardzo podobnie, z tą różnicą że nasze wartości są liczbami zmiennoprzecinkowymi, ale długości wektorów są takie same. Ze względu na to że długość wektorów może się znacząco rozrastać zastosowałem ograniczanie że w moim modelu maksymalna długość wektora, czyli maksymalna liczba unikalnych słów, to 1000.

Architektura Modelu i wyniki

Moim głównym celem było uzyskanie jak najlepszego wyniku i generalizacji. Nie starałem się za wszelką cenę własnoręcznie napisać architektury modelu, ponieważ wiedziałem że ze względu na brak mojego doświadczenia taka architektura na pewno byłaby suboptymalna. Dlatego postanowiłem wykorzystać gotowe architektury modelów do regresji (oczywiście przy każdym takim modelu zapoznawałem się dokładnie jak on działa).

Zacząłem od prostego modelu regresji liniowej dostępnego w scikit-learn chciałem zobaczyć, jak radzi sobie podstawowy model, dodatkowo zastosowałem GridSearchCV który trenuje model dla różnych zestawów parametrów które zdefiniowałem, i wybiera ten z najlepszymi wynikami. Okazało się że model dla wielu różnych parametrów osiąga ten sam wynik: MAE ok. 5300 i R2 ok. 0.3 gdzie MAE to średnia wartość bezwzględnych błędów między przewidywanymi wartościami a rzeczywistymi wartościami(Liczby retweetów) a R2 to współczynnik determinacji, który mierzy, jak dobrze przewidywane wartości modelu dopasowują się do rzeczywistych danych. Wartość R2 wynosi od 0 do 1, gdzie 1 oznacza idealne dopasowanie, a 0 oznacza, że model nie wyjaśnia żadnej zmienności danych.

Uznałem te wyniki za suboptymalne, Zaniepokoiło mnie to, że testując różne parametry, wyniki praktycznie się nie zmieniały co mogłoby sugerować że architektura modelu nie jest odpowiednia do tego problemu, lub dane (tylko oczyszczony tekst) nie są wystarczające aby przewidzieć liczbę retweetów. Następnym krokiem było zastosowanie odpowiednio Lasso regression i Ridge regression czyli modeli regresji liniowej z odpowiednio regularyzacją L1 i L2. Wyniki były odrobinę lepsze, osiągając MAE ~5300 R2 ~0.3 Spróbowałem wtedy

zastosować ElasticNet czyli połączenie Ridge regression i Lasso regression gdzie możemy ustawić nawet współczynnik mieszania pomiędzy regularyzacją L1 i L2, przygotowałem dużo potencjalnych parametrów tak że model był trenowany aż 75 razy dla różnych parametrów i wyniki zmieniły się na MAE ~4700 R2 ~0.4. Uznałem, że regresja liniowa może być zbyt mało wydolną architekturą dla skomplikowanych problemów, jak NLP, lub dane nie są wystarczające do przewidywania liczby retweetów., wtedy jeszcze nie wpadłem na pomysł wyciąganie dodatkowych cech. Dlatego zdecydowałem się wyjść ze sfery regresji liniowej i wypróbować bardziej zaawansowane techniki.

Wybór padł na tzw. „Ensemble learning" czyli podejście w którym łączymy kilka podstawowych modeli w celu stworzenia jednego optymalnego modelu predykcyjnego. Konkretnie zastosowałem RandomForestRegressor dostępny w scikit-learn, czyli model opary na drzewach decyzyjnych które są znacznie bardziej bardziej zaawansowane niż regresja liniowa z której wcześniej korzystałem. Działa to mniej więcej w ten sposób:

Tworzenie Drzew:

- Losowo wybierane są podzbiory danych treningowych z zamianą.
- Dla każdego podzbioru trenuje się drzewo decyzyjne.
- Podczas tworzenia każdego drzewa, na każdym etapie podziału, losowo wybierana jest określona liczba cech do rozważenia przy podziale.

Predykcja:

- Każde drzewo w lesie generuje swoją własną predykcję dla nowych danych.
- Wynik końcowy jest średnią predykcji ze wszystkich drzew.

Metoda ta dała najlepsze wyniki, udało jej się przełamać wcześniejsze bariery i uzyskać (bez cech dodatkowych) wyniki MAE ~4000 i R2 ~0.41. Jednak ogromnym minusem była długość trenowania danych korzystając z GPU na google Colab trening trwał znacząco powyżej godziny, co uniemożliwiło zastosowania GridSearchCV oraz analizy SHAP.

W tym momencie wpadłem na pomysł zastosowania danych dodatkowych. Na początku postanowiłem dodać wszystkie możliwe dane, aby znaleźć jakiś punkt odniesienia który będzie mówił jaki jest najlepszy (ale nieosiągalny wynik) zastosowałem dodatkowo liczbę polubień co spowodowało że model całkowicie przestał zwracać uwagę na tekst tweeta bo liczba retweetów silniej korelowała z liczbą polubień (co widać na wykresie z początku raportu). Uzyskałem wynik MAE ~1500 R2 ~0.9 Czyli wyniki były bardzo dobre, w szczególności R2 mówiło że model świetnie opisuje zmienność ale to oczywiście zasługa liczby polubień. Bardziej zależało mi na znalezieniu jakiejś sensownej wartości odniesienia MAE czyli o ile retweetów średnio myli się model, Z tego wyniku dowiedziałem się że realistycznie model nigdy nie osiągnie wyniku lepszego niż 1500.

Po tym wszystkim zacząłem stosować cechy dodatkowe takie jak opisałem je w poprzedniej części, z nimi RandomForestRegressor osiągnął świetny wynik MAE ~3400 i R2 ~0.42 Jednak ze względu na ekstremalnie długie trenowanie musiałem go porzucić bo nie było możliwości żeby przeprowadzić na nim analizę SHAP. Wtedy zdecydowałem się na kompromis, i zastosowałem zaawansowany framework LightGBM który stosuje metodę gradient boosting z dodatkowymi optymalizacjami. Zastosowałem go aby osiągnąć wynik porównywalny do RandomForestRegressor ale w realistycznym czasie który pozwoliłby mi na dostosowanie parametrów, szybki trening i nawet

zastosowania analizy SHAP która zajmuje wielokrotnie więcej czau niż sam trening.

Czym jest LightGBM?

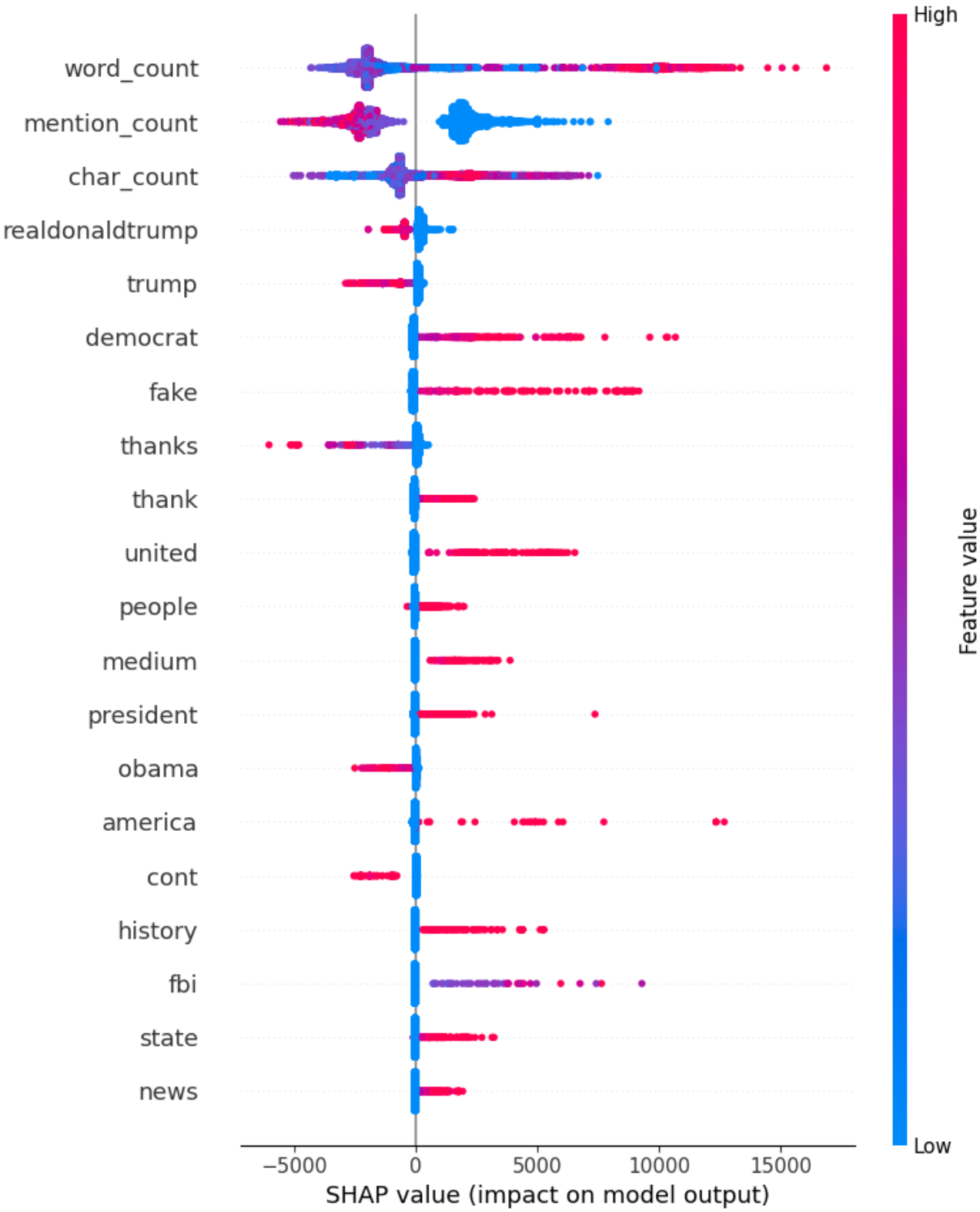
Jak mówi oficjalny opis: LightGBM (Light Gradient Boosting Machine) to zaawansowany algorytm uczenia maszynowego zaprojektowany z myślą o wysokiej wydajności i niskim zużyciu zasobów obliczeniowych. Jest to efektywny algorytm gradient boosting, który nadaje się do dużych zbiorów danych i problemów z wieloma cechami. LightGBM został opracowany przez Microsoft.

Co ciekawe mimo jego zaawansowania LightGBM korzystający z cech dodatkowych osiągnął gorszy wynik niż RandomForestRegressor(nawet bez cech dodatkowych) a dokładnie MAE ~3900 i R2 ~ 0.43 (Dla RFR, było to odpowiednio ~3400 i ~0.42). LightGBM okazał się jednak niezwykle wydajny i to jego wybrałem jak model dla którego przeprowadzę analizę SHAP.

Analiza SHAP

Zanim przedstawię wyniki analizy SHAP, warto zapoznać się z tym czym w ogóle jest SHAP.

SHAP (SHapley Additive exPlanations) to metoda interpretacji modeli uczenia maszynowego, która przypisuje każdej cesze wartość ważności w kontekście jej wkładu do predykcji modelu. SHAP opiera się na koncepcji wartości Shapleya z teorii gier, która służy do rozdzielania zysków między graczy w sposób sprawiedliwy i spójny.



Wykres dla ostatecznego modelu LightGBM + Cechy dodatkowe.

Wyjaśnijmy co tutaj w ogóle widzimy

Oś pozioma (SHAP value):

Wartości SHAP (SHAP value) na osi poziomej pokazują, o ile każda cecha przesuwa przewidywaną wartość modelu.

Wartości dodatnie przesuwają przewidywaną wartość w górę, a wartości ujemne przesuwają ją w dół.

Oś pionowa (Feature names):

Na osi pionowej znajdują się nazwy cech (features) używanych w modelu.

Kolory (Feature value):

Kolory punktów reprezentują wartości cech. Kolor czerwony oznacza wysokie wartości cechy, a kolor niebieski oznacza niskie wartości cechy.

Na przykład, jeśli cecha "word_count" jest czerwona, oznacza to, że dla tych obserwacji liczba słów w tweecie jest wysoka.

Interpretacja:

word_count:

Większość punktów po prawej stronie jest, co oznacza że większa liczba słów (wysokie wartości) zwiększa przewidywaną wartość retweetów.

mention_count:

Co ciekawe widzimy że mniejsza ilość oznaczeń innych użytkowników koreluje z większą liczbą retweetów, a większa liczba wspomnień z mniejszą liczbą retweetów, jest to zgodne z moimi przemyślaniami z początku raportu.

char_count:

Większość punktów jest niebieska po lewej stronie, co sugeruje, że mniejsza liczba znaków (niskie wartości) zmniejsza przewidywaną wartość.

realDonaldTrump, trump, democrat, fake, itp.:

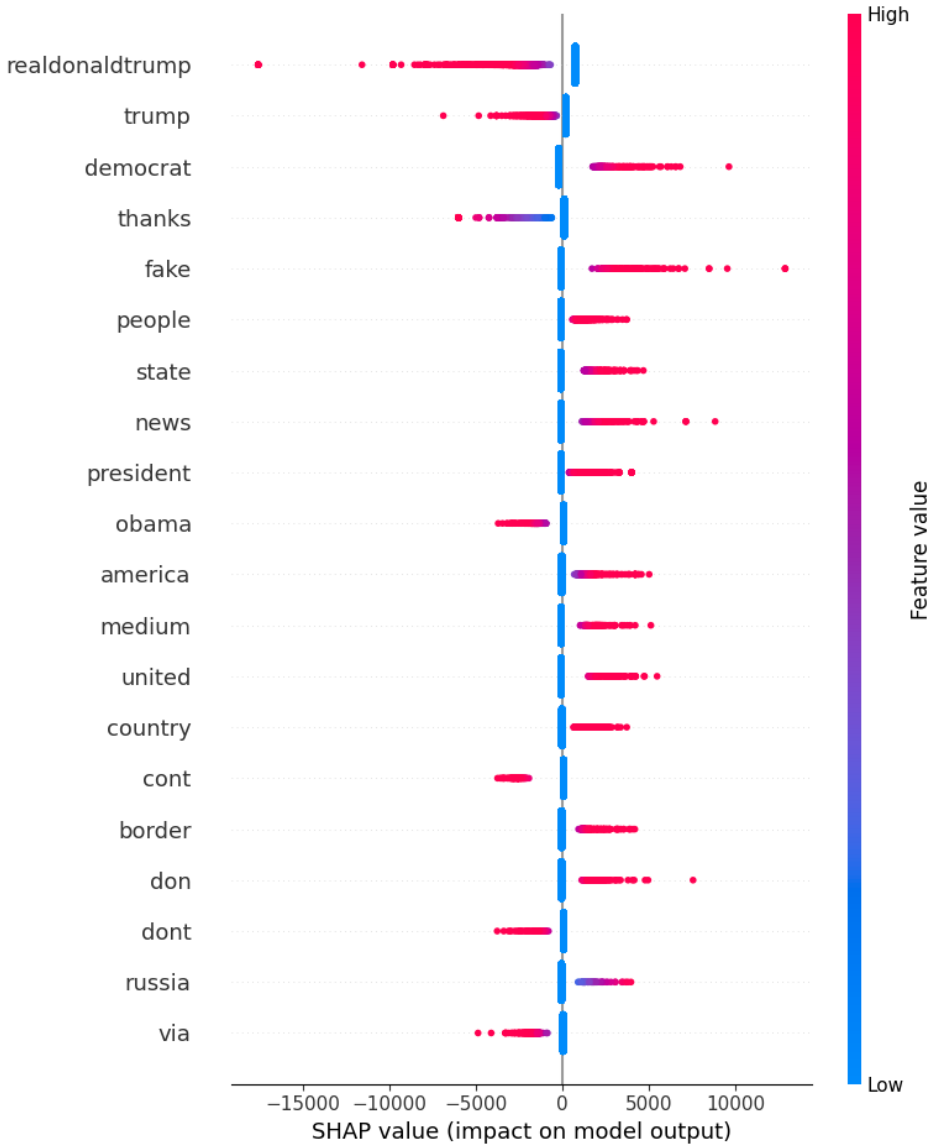
Słowa kluczowe jak „trump", „democrat", „fake”, „united”, „America” mają duży wpływ na przewidywaną wartość, co wskazuje na to że angażują użytkowników platformy prowokując ich do retweetowania.

Warto zauważyć że na wykresie pokazane są własności które mają największy wpływ na liczbę retweetów.

Z tego widać że model uznał nasze meta-informacje które wydobyliśmy z tekstu za cenne i dobrze je wykorzystuje. Jednak nie dominują one do tego stopnia że model całkowicie zaniedbuje sam tekst.

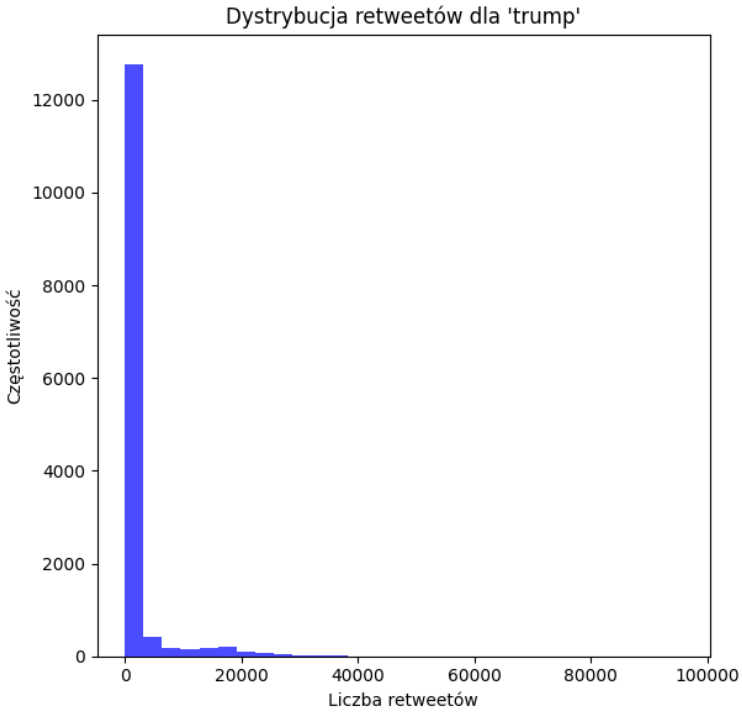
Dla porównania pokażę jeszcze wyniki dla modelu w którym nie mamy żadnych cech dodatkowych

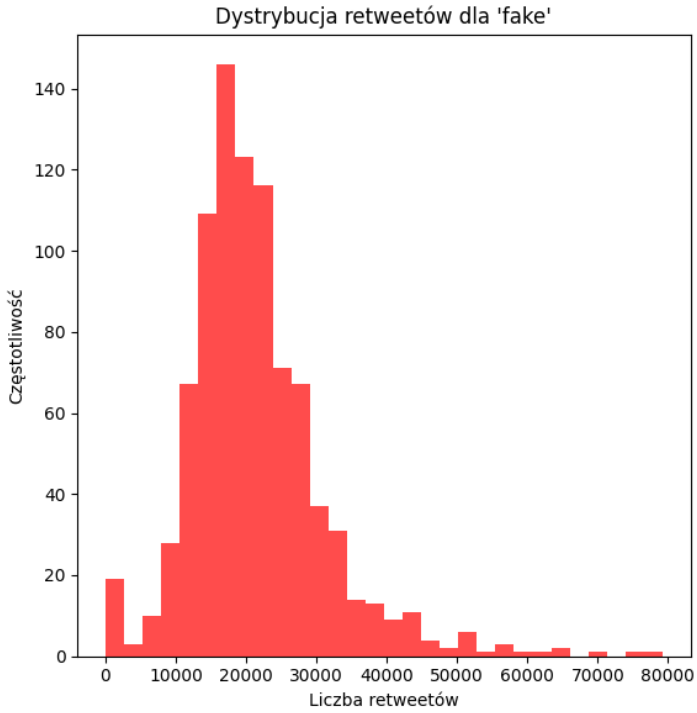
MAE: 53524, R2: 0.30



Widzimy tutaj że słowa które uznane są za najważniejsze pod względem wpływu na retweety, są takie same jak w modelu który ma dodatkowo cechy dodatkowe.

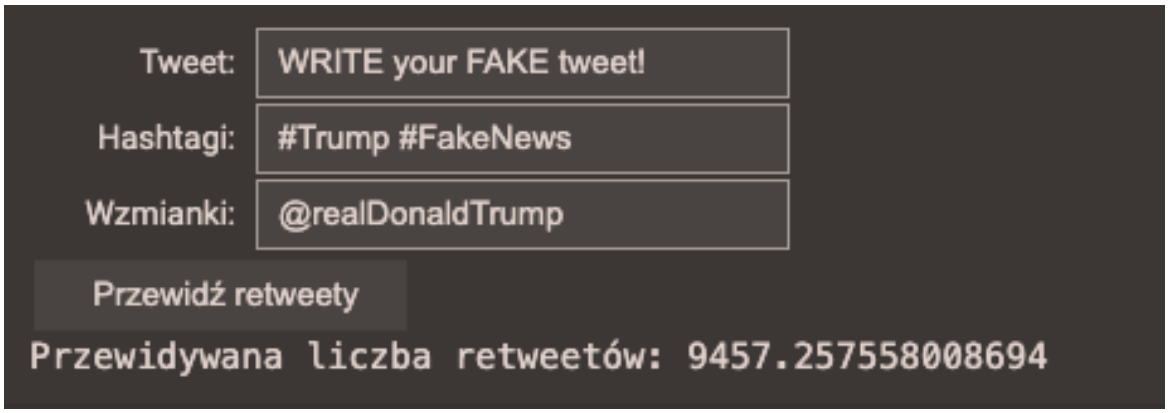
Co ciekawe patrząc na rozkład retweetów dla tweetów zawierających określone słowa możemy zobaczyć dlaczego model tak nie lubi słowa „trump" a dlaczego lubi słowo „fake"





Cały kod źródłowy wraz z szczegółowymi komentarzami znajduje się w repozytorium GitHub, razem z zapisanym modelem lightGBM i wektoryzatorem TF-IDF tak aby nie trzeba było cokolwiek trenować.

Przygotowałem jeszcze mini-program działający w środowisku Jupyter notebook który po załadowaniu modelu, może przewidzieć nasz własny tweet


Tweet:
WRITE your FAKE tweet!
Hashtagi:
#Trump #FakeNews
Wzmianki:
@realDonaldTrump
Przewidź retweety
Przewidywana liczba retweetów: 9457.257558008694