Wprowadzenie do eksploracji danych tekstowych w sieci WWW

Projekt 2018 Z

Prognozowanie istotnych zmian kursow walut na podstawie informacji z portali informacyjnych

Igor Markiewicz
Paweł Walczak

Prowadzący – dr inż. Piotr Andruszkiewicz

Spis treści

1 Opis tematu

Głównym celem projektu było prognozowanie istotnych zmian kursu ustalonej waluty na podstawie informacji z portali aukcyjnych. Zadanie zostało potraktowane jako problem klasyfikacji, polegający na predykcji zmian (spadek, wzrost, bez znaczenia) kursu na podstawie informacji tekstowych. Wybraną walutą został Bitcoin.

2 Moduły zbierania danych

2.1 Moduł zbierania cen

2.1.1 Opis

Pierwszym elementem implementacji systemu na potrzeby projektu było stworzenie modułu przeznaczego do pobierania cen BTC. Do tego celu użyliśmy języka Python oraz pewnych dodatkowych bibliotek służących do tworzenia 'HTTP-requests'. Przed implementacją kluczowym zadaniem było znalezienie w sieci WWW, API dzięki któremu moglibyśmy pobierać ceny BTC, zarówno dla chwili terazniejszej jak i dla pewnego zakresu czasowego. Nasze wymagania spełniało REST API udostępnione pod adresem: *api.coindesk.com* W naszym programie udostępniliśmy dwie możliwości pobierania cen:

- dla chwili terazniejszej (moment uruchomienia skryptu)
- dla danego zakresu czasowego, określonego przez datę startu oraz datę zakończenia (z jednodniowym interwałem czasowym)

Pliki z cenami są zapisywane w formacie CSV, w folderze dostępnym pod nazwą ./data. Kolumny pliku CSV to data oraz cena (wyrażona w jednostce USD/BTC).

	date	price
2	2013-06-18	107.35
3	2013-06-19	108.251
4	2013-06-20	111.29
5	2013-06-21	109.5
6	2013-06-22	108.2
7	2013-06-23	107.9
8	2013-06-24	102.09
9	2013-06-25	103.3293
10	2013-06-26	104
11	2013-06-27	101.7368

Rys. 1: Przykładowy plik csv

2.1.2 Sposób uruchomienia

Aby uruchomić skrypt, należy udać się do folderu, w którym znajduje się plik ze skryptem (*Modules/BitcoinPriceCrawler*). Skrypt uruchamiamy z konsoli w następujący sposób: *python3 scrapePrice.py trybPobieraniaCen dataStartu dataKonca* gdzie:

- *trybPobieraniaCen* – argument odpowiadający za to, czy skrypt ma pobierać cenę aktualną czy historyczną, 'historic', 'current'
- *dataStartu, dataKonca* - arugmenty wymagane w przypadku trybu pracy 'historic', określające zakres dat z jakich chcemy pobrać ceny BTC

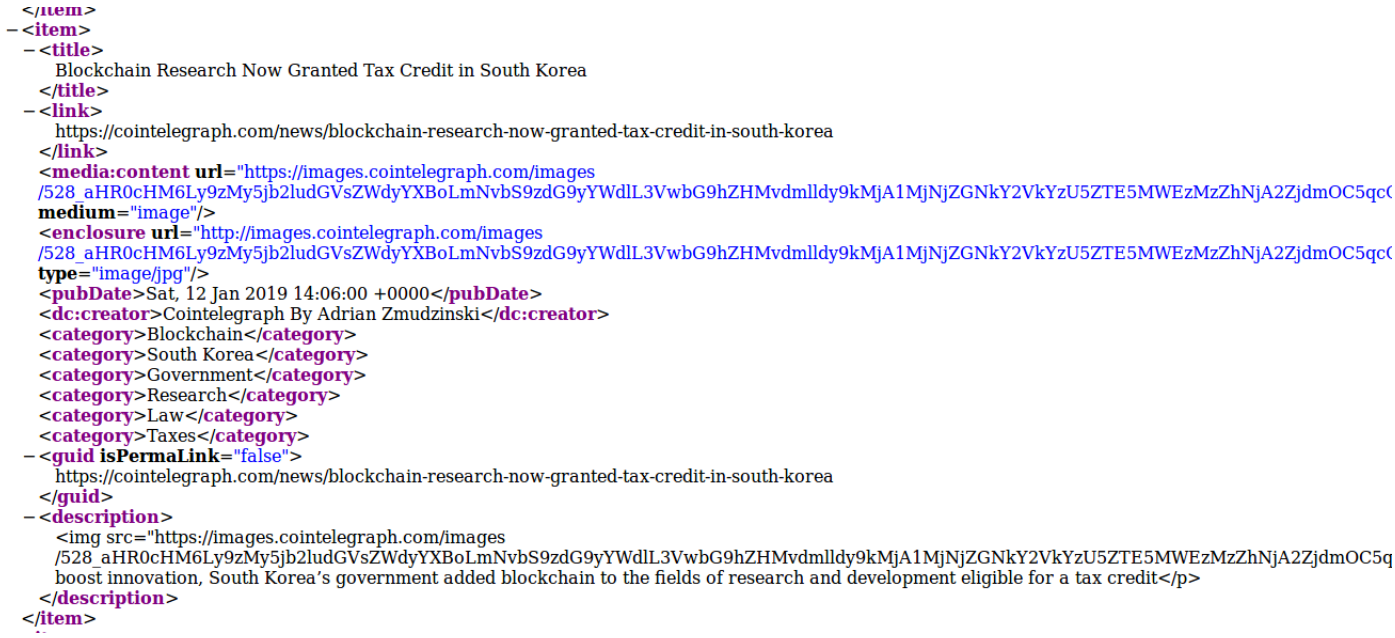
2.2 Moduł zbierania informacji tekstowych

2.2.1 Opis

Kolejnym elementem implementacji systemu, było dostarczenie danych w celu stworzenia klasyfikatora. Stworzone zostały dwa podmoduły:

- pierwszy uruchamiany wyłącznie raz w celu zebrania historycznych danych na potrzeby procesu uczenia modelu
- drugi uruchamiany w celu zebrania 'świeżych artykułów'

Istotnym wyzwaniem w trakcie realizacji projektu, było znalezienie w sieci WWW dobrych zródeł danych. Początkowo skupiliśmy się na podejściu, które wykorzystywałoby standardowe strony internetowe i blogi, oraz stworzenie 'crawlera', który przeszukiwałby takie miejsca i odpowiednio zapisywał zebrane dane. Zrezygnowaliśmy z tego podejścia, ze względu na potrzebę tworzenia dla każdej strony internetowej osobnego, specjalnie dostowanego crawlera. Ostatecznym podejściem, było przeszukanie sieci i wykorzystanie tzw. RSS FEEDS. Jest to ustandaryzowany sposób tworzenia zródeł danych (w postaci plików XML).



Rys. 2: Przykładowy plik XML

Dla pierwszego modułu (zbieranie świeżych artykułów), wykorzystaliśmy RSS FEED udostępniony przez stronę `https://cointelegraph.com/`. Nasz wybór motywujemy wysokim wskaznikiem authority tejże strony. Jest to populatna strona o tematyce kryptowalut. Artykuły udostępniane w tym serwisie cieszą się wysoką populatnością wśród internautów (dziesiątki tysięcy wyświetleń każdego z artykułów). Dane zebrane z tego portalu, posłużyły nam do bieżącej weryfikacji wyników naszego modelu.
Dla drugiego modułu wykorzystaliśmy RSS FEED udostępniony przez serwis letstalkbitcoin.com. Na tym etapie szukaliśmy w sieci miejsca gdzie udostępniony feed, byłby z dużego zakresu czasowego. Potrzebne była również duża ilość danych w celu wyuczenia i wytrenowania klasyfikatora. Jak się okazało, zdecydowana większość udostępnianych feed-ów, była jedynie dla najświeższych artykułów (zazwyczaj około kilka dni wstecz) oraz o bardzo ograniczonej ilości (około 40). Niedopuszczalne było dla nas zebranie artykułów z tak małego okna czasowego. Serwis `https://letstalkbitcoin.com/` wyróżniał się możliwością stworzenia 'customowego' zestawu danych. Sumarycznie udało nam się zebrać około 2200 artykułów z około 4 lat. Niestety nie byliśmy w stanie stwierdzić istotności każdego z arytkułów, ze względu na ich dużą ilość. W obu przypadkach dane zapisujemy w postaci pliku csv, a kolumnami *date, title i description*. Jako, że w kolumnie *desciption*, pojawiały się tagi html-owe, dane musieliśmy poddać delikatnej obróbce, poprzez usunięcie wszystkich tagów.

2.2.2 Sposób uruchomienia

Aby uruchomić skrypt, należy udać się do folderu, w którym znajduje się plik ze skryptem (*Modules/RssFeedModule*). Skrypt uruchamiamy z konsoli w następujący sposób: *python3 feedCrawler.py trybPobierania* gdzie:

- trybPobierania, argument odpowiadający za to, czy skrypt ma pobierać feed aktualny czy historyczny, 'historic', 'current'

Zapisane pliki csv znajdują sie w podfolderze data.

3 Analiza danych

3.1 Przepływ danych

Szczegółowy przepływ danych możemy wyrazić w najstępujący sposób:

1. plik *parseData.py* – na jego wejście podajemy następujące argumenty:

   - --path_to_prices – ścieżka do pliku z historią cen
   - --path_to_messages – ścieżka do pliku z zebranymi wiadomościami tekstowymi
   - --save_to – ścieżka do folderu, w którym chcemy zapisać plik wynikowy (z jego nazwą)
   - --threshold – moduł wartości procentowej zmiany, powyżej której uznajemy istotny spadek lub wzrost. Zmianę procentową definiujemy jako:

   $$diff\left(y_i, y_{i+1}\right) = 100 \cdot \frac{y_{i+1} - y_i}{y_i}$$

   $$\begin{cases} diff\left(y_i, y_{i+1}\right) \leq -threshold \implies \text{istotny spadek} \\ diff\left(y_i, y_{i+1}\right) \geq threshold \implies \text{istotny wzrost} \\ w.p.p \implies \text{brak istotnej zmiany} \end{cases}$$

   W tym pliku wykonujemy następujące czynności:

   (a) obliczamy zmiany procentowe kursu pomiędzy dwoma kolejnymi notowaniami i przypisujemy odpowiadające klasy

   (b) dołączamy tematy artykułów do ich treści

   (c) grupujemy powstałe informacje względem daty

   (d) łączymy kategorie z pogrupowanymi informacjami względem daty

   (e) zapisujemy całość do pliku wynikowego

2. plik *NLPModel.py* – zestaw funkcji odczytujących i przetwarzających dane, oraz budujących model. Dwie główne funkcje:

   - *build_model* – funkcja budująca model z argumentami:
     - *path_to_data* – ścieżka do pliku z połączonymi wiadomościami oraz odpowiadajacymi im klasami
     - *is_model_saved* – czy zapisujemy wyuczony model

- *path_to_model_dir* – folder do którego zapisujemy model (wraz z nazwą modelu) jeśli *is_model_saved = True*
- *test_size* – ułamek przykładów jaki idzie na testowanie modelu (podział na zbiór trenujący i testowy następuje metodą losowania Monte Carlo)
- *split_data_random_state* – stan maszyny generującej liczby pseudolosowe (przydatny jeśli jest potrzeba deterministycznego podziału zbioru)
- *additional_stop_words* – zbiór dodatkowych słów, które chcielibyśmy usunąć z tekstu
- ***params* – słownik zawierający parametry modelu

- *read_and_preprocess* – funkcja do odczytywania i przetwarzania danych z argumentami:
  - *path* – ścieżka do pliku z połączonymi wiadomościami oraz odpowiadajacymi im klasami
  - *is_training* – jeśli *True* to następuje losowy podział na podzbiory trenujący i testowy (parametr *test_size*), przetworzenie danych oraz połączenie w *TaggedDocument* (używane przy trenowaniu modelu). Jeśli *False*, to dane są przetwarzane i zwracane w postaci listy list z tokenami (w przypadku wnioskowania)
  - *test_size* – ułamek przykładów jaki idzie na testowanie modelu
  - *additional_stop_words* – zbiór dodatkowych słów, które chcielibyśmy usunąć z tekstu
  - *split_data_random_state* – stan maszyny generującej liczby pseudolosowe (przydatny jeśli jest potrzeba deterministycznego podziału zbioru)

  Przetwarzanie danych w tej funkcji polega na:

  (a) tokenizacja, ususwanie części znaków, zamiana wielkich liter na małe (funkcja *simple_preprocess* z pakietu *Gensim* [3])

  (b) stemizacja przy użyciu *SnowballStemmer* [5] dla języka angielskiego z pakietu *NLTK*

  (c) filtrowanie wyrazów ze stop listy (z *NLTK* dla język angielskiego + *additional_stop_words*)

3. plik *modelTests.py* – przykładowe wywołanie funkcji z pliku *NLPModel.py*

3.2 Algorytm Doc2Vec

Modele są wyuczane metodą *Paragraph Vector – Distributed bag of words(PV–DBOW)* [1, 2, 4]. W [1] (*3.4. Some further observations*) istnieje sugestia że konkurencyjna metoda *Paragraph Vector – Distributed Memory(PV–DM)* z racji brania pod uwagę kontekstu wyrazów może sprawować się lepiej, jednak w trakice wstępnych eksperymentów nie udało się potwierdzić jej wyższości nad *PV–DBOW*. Z tego powodu oraz z tego że metoda *PV–DBOW* wymaga

mnejszych nakładów obliczeniowych postanowiono ostatecznie przyjąć *PV–DBOW* do rozwiązania problemu.

4 Wyniki

4.1 Opis danych

Zebrane dane to:

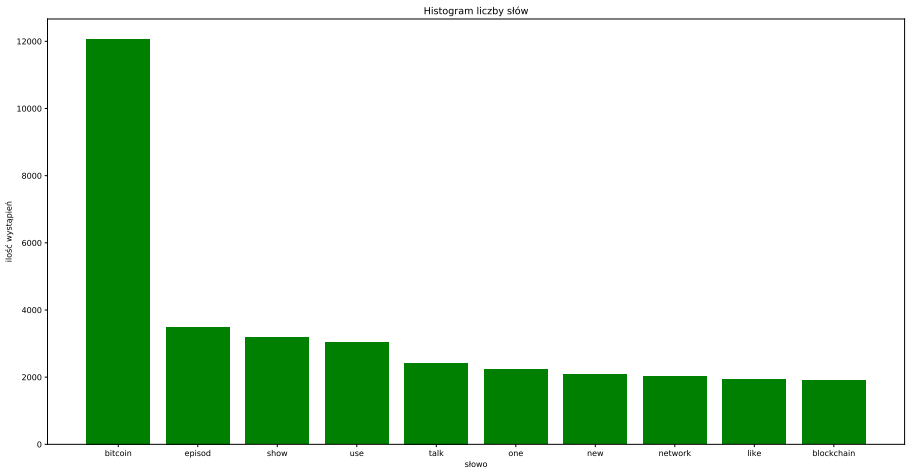
- dzienne ceny Bitcoina między 18. czerwca 2013 a 2. stycznia 2019, plik *BitcoinPriceCrawler/data/BitcoinPrice_from_2013-06-18_to_2019-01-02.csv* – 2025 rekordów
- wiadomości tekstowe między 18. czerwca 2013 a 2. stycznia 2019, plik *RssFeedModule/data/historicNews.csv* (przy czym może być wiele wiadomości jednego dnia) – 2065 rekordów

Z powyższych danych otrzymano w wyniku działania *parseData.py* 1391 rekordów (plik *DataAnalysis/data/ProcessedData.csv*).

4.2 Założenia

W trakcie testów przyjęto następujące założenia :

- Na wstępie zdecydowano się na usunięcie dodatkowych wyrazów z danch: 'sbquo', 'www', 'http', 'com'. Poniżej przedstawiony został histogram po korekcji.



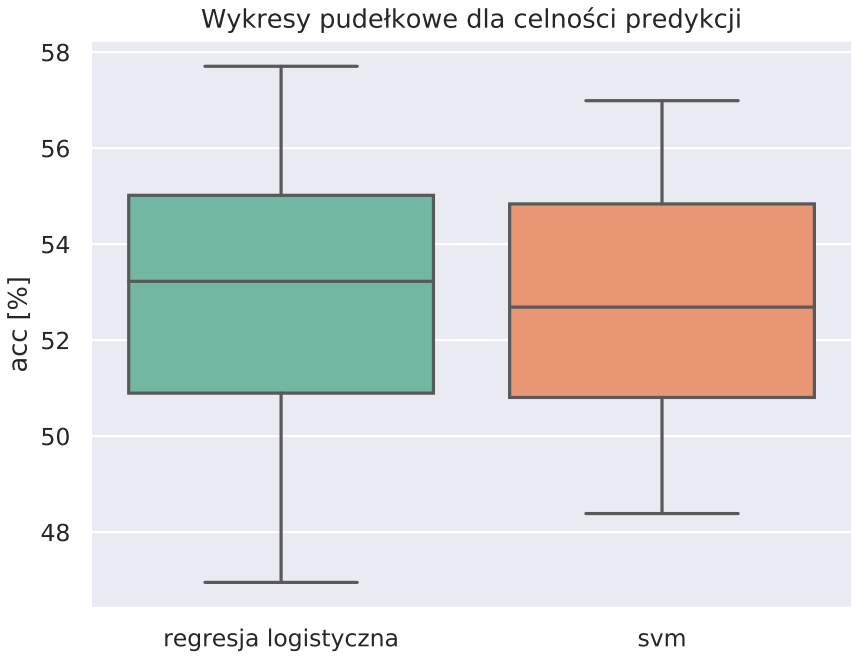
Rys. 3: Histogram ilości wystąpień 20. najczęstszych słów po usunięciu dodatkowych słów

- problem został potraktowany jako problem binarny – każda zmiana kursu jest uznawana za istotny spadek lub wzrost
- do klasyfikacji wykorzystano dwa algorytmy: regresja logistyczna, SVM
- jako metryki oceny klasyfikacji wybrano: celność predykcji, współczynnik F_1
- przeprowadzono 20 testów, z których każdy polegał na podziale całego dostępnego zbioru na podzbiór uczący (80 %) oraz testowy (20 %) metodą losowania Monte Carlo
- jako parametry *PV–DBOW* ustalono:
  - *vector_size* = 300 – rozmiar wektora cech
  - *window* = 10 – maksymalny rozmiar okna
  - *min_count* = 2 – minimalna ilość wystąpień danego wyrazu aby był brany pod uwagę
  - *epochs* = 100 – liczba epok przy uczeniu sieci neuronowej
  - *alpha* = 0.025 – początkowa wartość współczynnika szybkości uczenia
  - *min_alpha* = 0.025 – końcowa wartość współczynnika szybkości uczenia
- podczas retrenowania częsci sieci (zarówno dla przykładów treningowych jak i testowych) wartości odpowiednich parametrów nie zmieniają się.

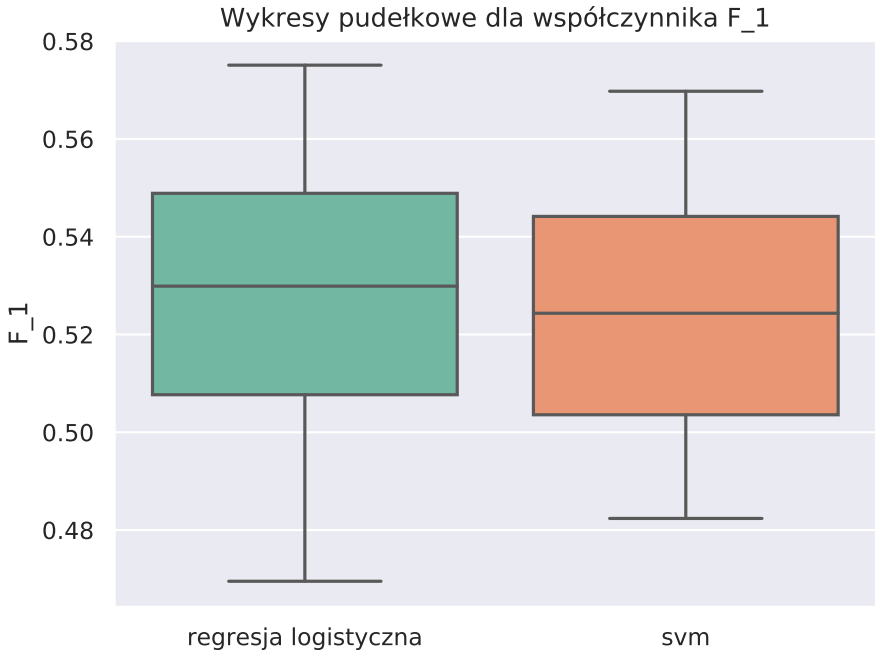
4.3 Statystyki

	minimum		**średnia arytmetyczna**		**mediana**		**maksimum**		**odchylenie standardowe**	
	acc [%]	F_1	acc [%]	F_1	acc [%]	F_1	acc [%]	F_1	acc [%]	F_1
regresja logistyczna	47,0	0.47	52,9	0.53	53.2	0.53	57.7	0.58	3,1	0.03
svm	48,4	0.48	52,7	0.52	52.9	0.52	57.0	0.57	2,6	0.03

Tab. 1: Statystki wyników klasyfikatorów



Rys. 4: Wykresy pudełkowe dla celności predykcji



Rys. 5: Wykresy pudełkowe dla współczynnika F_1

Możemy zauważyć że wyniki dla obu klasyfikatorów są bardzo zbliżone.

Poniżej przedstawione zostały testy statystyczne (dla 20 próbek) sprawdzające czy klasyfikatory generują istotnie statystycznie wynki. Do tego celu posłużył prawostronny test t–Studenta dla celności predykcji:

$$\begin{cases} H_0 : \mu = 50 \\ H_1 : \mu > 50 \end{cases}$$

oraz dla współczynnika F_1

$$\begin{cases} H_0 : \mu = 0.5 \\ H_1 : \mu > 0.5 \end{cases}$$

gdzie μ oznacza średnią arytmetyczną celności predykcji lub współczynnika F_1.
W rezultacie otrzymano

	acc	F_1
regresja logistyczna	$< 2.2 \cdot 10^{-16}$	0.0003063
svm	$< 2.2 \cdot 10^{-16}$	0.0001658

Tab. 2: Uzyskane p–value

Możemy zauważyć że w każdym z przypadków z racji na bardzo małe p–value możemy odrzucić hipotezę zerową na rzecz elternatywnej, więc wszystkie wyniki rzeczywiście są większe od wartości odpowiednio 50 % oraz 0.5, a co za tym idzie klasyfikatory radzą sobie lepiej od klasyfikatora losowego.

4.4 Wnioski

Przykładowe zabiegi mogące poprawić otrzymane rezultaty:

- uzyskanie większej liczby przykładów trenujących
- rozważenie sytuacji w których nie wszystkie zmiany tratujemy jako istotne
- zastosowanie bardziej zaawansowanych metod wstępnego przetwarzania danych tekstowych
- próba zastosowania innego modelu np: *PV–DM* lub połączenie kilku technik
- optymalizacja parametrów modelu
- być może zastosowanie innych rodzajów klasyfikatorów

5 Bibliografia

[1] https://cs.stanford.edu/~quocle/paragraph_vector.pdf

[2] https://pdfs.semanticscholar.org/83fb/367773e90d3bc5d8bd90c1529e6af927c760.pdf?_ga=2.178386470.325574951.1546519475-306362781.1546519475

[3] https://radimrehurek.com/gensim/

[4] https://radimrehurek.com/gensim/models/doc2vec.html

[5] https://www.nltk.org/_modules/nltk/stem/snowball.html

[6] *wykłady przedmiotowe*