Politechnika Warszawska
Wydział Elektryczny

System do analizy sentymentów spółki na podstawie wpisów w mediach społecznościowych.

Autorzy:
Aleksei Haidukevich, nr albumu 295233
Jakub Korczakowski, nr albumu 291079
Marharyta Kruk, nr albumu 295235
Maciej Leszczyński, nr albumu 291085
Piotr Rosa, nr albumu 291112

10 czerwca 2020

Spis treści

1 Etap 1 - Zbieranie danych

czas: 4.11.2019 - 15.12.2019

1.1 Opis

W pierwszym etapie projektu najważniejszym celem było utworzenie narzędzi do zbierania danych z Twittera i Reddita. Zostaną one użyte w celu stworzenia zbioru danych pozwalającego na naukę i testowanie algorytmów. Pobrane wiadomości i nagłówki będą składowane w bazie danych. Ten etap prac zawierał również przygotowanie infrastruktury zdolnej analizować i przechowywać zebrane dane.

1.2 Zadania

1.2.1 Analiza dostępności API mediów społecznościowych.

Odpowiedzialni: **Piotr Rosa, Aleksei Haidukevich**
Po przeanalizowaniu API zdecydowaliśmy się wykorzystać media społecznościowe **Twitter** i **Facebook**. API Twittera jest bardzo rozwinięte i pozwala na swobodny dostęp do tweetów, nawet w przypadku darmowej wersji konta. Zdecydowaliśmy się pobierać tweety wykorzystując bibliotekę Pythona - **Tweepy**. W przypadku Facebooka korzystamy z programu **Face-Pager**. Jest to program pozwalający pobierać wszystkie posty z wybranej strony i zapisać je do pliku z rozszerzeniem csv. Opcjonalna jest dodatkowa konwersja do JSON. Posty można pobierać z całego okresu istnienia strony.

1.2.2 Analiza dostępnej infrasturktury do przetwarzania danych.

Odpowiedzialny: **Jakub Korczakowski**
W docelowej aplikacji przetwarzającej dane w czasie rzeczywistym planujemy użyć:

Apache Hadoop do pobierania tweetów w czasie rzeczywistym,

Apache Spark do przetwarzania danych i uruchamiania modeli,

MS SQL do przechowywania tweetów.

Elementy architektury aplikacji prawdopodobnie ulegną jeszcze zmianie podczas dalszego rozwoju projektu.

1.2.3 Analiza i wybór sposobu wdrożenia aplikacji (rozwiązania chmurowe).

Odpowiedzialni: **Jakub Korczakowski, Maciej Leszczyński**
Po analizie potrzeb naszego projektu zdecydowaliśmy się wykorzystać roz-

wiązanie chmurowe, ponieważ pozwala one na zapewnienie dostępu do zasobów projektowych(takich jak bazy danych czy maszyny wirtualne) dla wszystkich pracujących nad projektem.

Wśród dostawców infrastruktury chmurowej można wyróżnić dwie firmy, które oferują darmowe środki dla studentów. Są to Amazon (AWS) oraz Microsoft (Azure). Z powodu mniejszych możliwości oraz mniejszej ilości środków dostępnych na patformie AWS zdecydowaliśmy się wybrać platformę Azure.

W obrębie tej platformy oprócz klasycznych maszyn wirtualnych dostępne są rozwiązania docelowe przeznaczone do przetwarzania dużych ilości danych, należą do nich:

- HDInsight,
- Azure Databricks,
- Azure Data Lake Services.

Pomimo tego, że te usługi znacznie ułatwają budowę projektów nie zdecydowaliśmy się na ich użycie ze względu na wysoką cenę. Planujemy używać maszyn wirtualnych i za pomocą Dockera, a w przyszłości Kubertenesa zbudować infrastrukturę projektu.

1.2.4 Wybór odpowiedniej bazy do składowanych danych (porównanie SQL i noSQL).

Odpowiedzialni: **Jakub Korczakowski, Maciej Leszczyński**
Do składowania historycznych danych w naszym projekcie będziemy wykorzystywać bazę SQL ze względu na szybkość dostępu do danych.

1.2.5 Stworzenie programu pobierającego dane z Twittera.

Odpowiedzialny: **Piotr Rosa**
Korzystając z biblioteki **Tweepy** udało nam się stworzyć skrypt pobierający wpisy z Twittera dotyczące wybranego słowa kluczowego. API Twittera pozwala na pobieranie naprawdę wielu szczegółów dotyczących wpisów, jednak do naszych celów nie potrzebujemy ich wszystkich. Zdecydowaliśmy się na pobieranie: ID tweeta, datę jego stworzenia, nazwę użytkownika oraz zawartość tweeta.
Pobrane dane przechowujemy w tabelach, osobno dla każdej firmy, z dodatkową kolumną Sentyment, która mówi o tym, czy przesłanie wiadomości jest negatywne, czy pozytywne.
Skrypt ten pobiera dane w interwałach czasowych, przy czym nie wszystkie tweety są zapisywane, a losowana jest jedynie część z nich. Wynika to z faktu, że często pojawia się więcej wpisów niż się spodziewaliśmy, a możemy nie poradzić sobie ze zbyt dużym zbiorem danych.

1.2.6 Stworzenie programu pobierającego dane z Facebooka.

Odpowiedzialny: **Aleksei Haidukevich**
Pobieranie danych z Facebooka przy użyciu programu FacePager pozwala na dostęp do danych niezależnie od dat powstania postów. Dzięki temu, uruchamiając program co określony czas dla wybranych stron, otrzymujemy pliki csv, z których dane przetwarzane są przy użyciu języka Python. Interesujące nas dane są analogiczne do tych w przypadku Twittera.

1.2.7 Instalacja wybranej bazy.

Odpowiedzialny: **Jakub Korczakowski**
Pobrane tweety składujemy w bazie SQL znajdującej się na platformie Azure. Obecnie bazy danych pomieścić może do 2GB danych, jednak w przypadku gdy potrzebne nam będzie więcej przestrzeni możemy w każdej chwili zwiększyć pojemność bazy. Jest to jedna z zalet użycia chmury. Baza danych dostępna jest pod statycznym adresem IP, więc jest łatwo dostępna dla każdego. Konieczne jest jedynie dostosowanie zapory sieciowej w celu umożliwienia dostępu.

1.2.8 Połączenie bazy danych z programami pobierającymi dane.

Odpowiedzialny: **Jakub Korczakowski**
Serwer bazodanowy działa na platformie **Azure**. Zarówno dla danych z Twittera, jak i z Facebooka, łączenie z serwerem odbywa się przy wykorzystaniu biblioteki **pyodbc**. Serwer bazodanowy na Azure posiada statyczny adres, dlatego łączenie się z nią nie jest skomplikowane. W konfiguracji połączenie trzeba podać odpowiedni login, hasło, nazwę bazy danych oraz nazwę sterownika w naszym systemie, który zostanie wykorzystany do tego połączenia. Zazwyczaj sterownik ten jest już zainstalowany, wystarczy jedynie odnaleźć jego nazwę.

1.2.9 Pobranie danych do bazy danych.

Odpowiedzialny: **Maciej Leszczyński**
Baza danych z której korzystamy jest bazą SQL-ową. Po nawiązaniu połączenia, polecenia wykonywane są jak w przypadku korzystania z bazy danych lokalnie. Dane będą zapisywane w bazie danych w czasie rzeczywistym za pomocą opracowanych programów.

1.2.10 Analiza dostępnych zbiorów opisanych klasami, pozwalających na testowanie algorytmu.

Odpowiedzialna: **Marharyta Kruk**
Znaleziony został zbiór pozwalający na trenowanie oraz testowanie algo-

rytmu. Zbiór zawiera 1600000 opisanych tweetów. Dla każdego tweetu został obliczony sentyment.

Zbiór znajduje się pod adresem: adres.

1.3 Wykonanie zadań z pierwszego kamienia milowego

Podczas tego etapu nie byliśmy w stanie wykonać wszystkich zadań, które założyliśmy w statucie projektu. Udało nam się rozpocząć budowę modelu oraz znaleźć odpowiedni zbiór treningowy, pozwalający na analizę sentymentu. Udało się nam również poznać API Twittera oraz rozpocząć pracę nad programami pobierającymi dane. Próba wykorzystania wiadomości z Reddita zakończyła się niepowodzeniem ze względu na brak odpowiednich danych w serwisie. Trudniejsze niż przewidywaliśmy okazało się również zbudowanie klastra na platformie MS Azure. Potrzebowaliśmy więcej czasu, aby zapoznać się z możliwościami platformy, co spowodowało przesunięcie się pracy.

2 Etap 2 - Przetwarzanie danych

czas: 7.01.2020 - 31.03.2020

2.1 Opis

W drugim etapie do najważniejszych celów należą cele niezrealizowane poprawnie w pierwszym etapie, czyli budowa infrastruktury do przetwarzania danych oraz stworzenie programów pobierających dane. Kolejnymi celami będzie stworzenie programów przetwarzających dane i uruchomienie modeli na przygotowanej infrastrukturze. W odniesieniu do pierwszego etapu dokonaliśmy zmiany bazy danych na bazę NoSQL, zdecydowaliśmy się wybrać bazę MongoDB.

2.2 Zadania

2.2.1 Budowa infrastruktury do przetwarzania danych

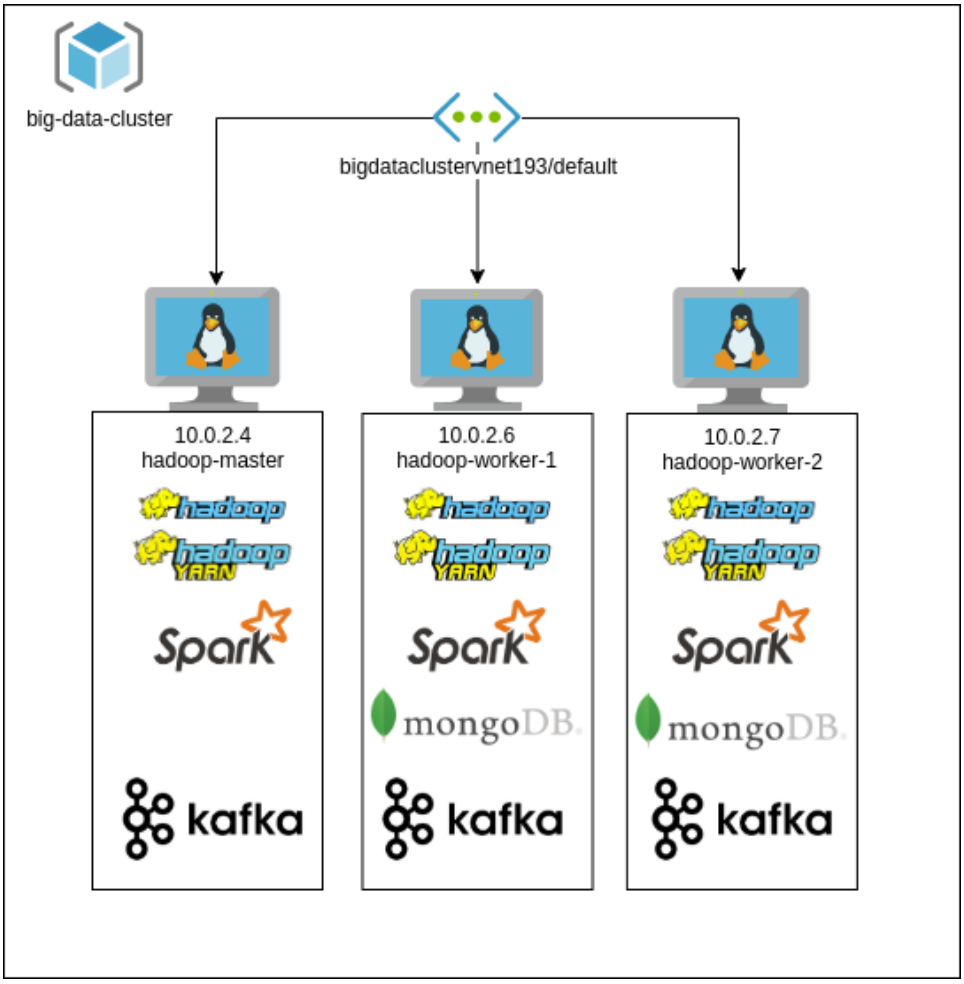
Odpowiedzialny: **Jakub Korczakowski**

W celu utworzenia infrastruktury zdolnej przetwarzać dane zostały utworzone 3 maszyny wirtualne w grupie zasobów **big-data-cluster** na platformie **Azure** w regionie **North Europe**. Są to odpowiednio:

hadoop-master Standard D2s v3 (2 vcpus, 8 GiB memory),

hadoop-worker-1 Standard B2s (2 vcpus, 4 GiB memory),

hadoop-worker-2 Standard B2s (2 vcpus, 4 GiB memory).



Rysunek 1: Diagram przygotowanej infrastruktury.

Ze względu na specyfikę platformy **Azure** wielkość i rodzaj maszyn może zostać przeskalowany w razie potrzeb. Wszystkie maszyny są dostępne pod publicznymi adresami IP poprzez protokół ssh. Dodatkowo maszyna **hadoop-master** posiada środowisko graficzne i można dostać się do niej również poprzez protokół RDP na porcie 3389.

Ilość maszyn jest zdeterminowana częściowo poprzez limit nałożony na subskrypcję studencką. Limit wynosi 6 vcpus na strefę regionalną maszyn.

2.2.2 Instalacja narzędzi do przetwarzania danych

Odpowiedzialny: **Jakub Korczakowski**

Apache Hadoop Został zainstalowany na klastrze. Wartość replikacji pli-

ków została zmieniona na **2** ze względu na ilość dostępnych DataNode.

YARN Został skonfigurowany w klastrze.

Apache Spark Został zainstalowany na klastrze oraz skonfigurowany w trybach działania **standalone** oraz poprzez **YARN**. Został połączony ze środowiskiem **Hadoop**.

MongoDB Baza została zainstalowana na maszynie **hadoop-master**. Została skonfigurowana w sposób umożliwiający odwołanie się do niej z sieci lokalnej(poprzez ip **10.0.2.4**). Poprzez odpowiednie opcje konfiguracyjne możliwe jest połączenie z **Apache Spark**.

Apache Kafka Broker Apache Kafka został zainstalowany jako klaster na wszystkich maszynach. Dodatkowo został zainstalowany Apache Zookeeper, który nadzoruje działanie brokera.

Rola maszyn w systemach.

	Apache Hadoop	YARN	Apache Spark
hadoop-master	NameNode	ResourceManager	Master
hadoop-worker-1	DataNode	NodeManager	Slave
hadoop-worker-2	DataNode	NodeManager	Slave

2.2.3 Baza Danych

Odpowiedzialny: **Jakub Korczakowski**

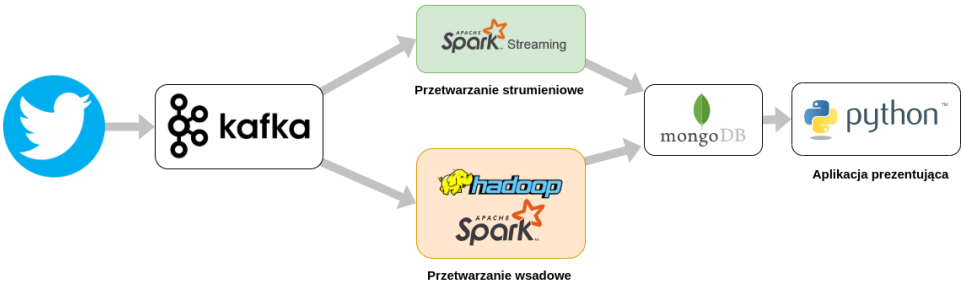
Baza danych została zainstalowana na maszynach **hadoop-woker-1** oraz **hadoop-woker-2**. Pierwsza z maszyn odpowiada za Twittera, a druga za Facebooka.

2.2.4 Potok przetwarzania danych w projekcie

Odpowiedzialny: **Jakub Korczakowski**

Podczas tego etapu udało nam się również zaplanować dokładniej potok przetwarzania danych w naszym projekcie. Planowany potok widoczny jest na poniższym rysunku.

Do pobierania danych w czasie rzeczywistym w naszym projekcie użyjemy Apache Kafka, który za pomocą skryptów napisanych w języku Python łączył się z Twitter API i pobierał dane. Przetwarzanie strumieniowe planujemy przeprowadzić za pomocą Apache Spark Streaming, a wyniki analiz zapisać w MongoDB. Przetwarzanie strumieniowe będzie obejmować głównie analizę sentymentu pobieranych tweetów. Wyniki z bazy będą na końcu prezentowane przez napisaną przez nas aplikację.



Rysunek 2: Diagram potoku danych w projekcie.

2.2.5 Udoskonalenie programu do pobierania danych z Twittera

Odpowiedzialny: **Piotr Rosa**

Program pobierający dane działa w języku Python z wykorzystaniem biblioteki Tweepy. Pobrane dane zapisywane są z wykorzystaniem systemu Avro. Pliki zapisane w tym formacie mają strukturę podobną, umożliwiającą łatwe przekształcenie, do formatu JSON. Główną wadą zapisu danych w JSON jest fakt, że nazwy pól zapisywane są dla każdego rekordu, co znacznie zwiększa zużycie pamięci, przez co nie nadaje się on dla dużych zbiorów danych. W przypadku Avro, schemat definiujący każdy rekord, zapisany jest jeden raz dla każdego pliku. Aby umożliwić współpracę programu z platformą Hadoop, tzn. zapis zebranych plików z danymi do systemu plików HDFS, wykorzystujemy bibliotekę Pythona hdfs, a zapis plików w formacie Avro umożliwia pakiet z tej biblioteki hdfs.ext.avro. Program, ze względu na brak możliwości nieustannego działania maszyn, uruchamiany jest domyślnie co dwa dni. Początkowo planowaliśmy pobieranie danych w odstępach tygodniowych, ponieważ Twitter pozwala na pobranie danych historycznych z okresu tygodnia, jednak istnieją również limity zapytań. Limit ten to 1800 zapytań w ciągu 15 minut, przy czym każde zapytanie zwraca maksymalnie 100 tweetów. Nie wystarcza to do naszych celów i konieczne byłoby czekanie około godziny na pobranie wszystkich interesujących nas wpisów, dlatego zdecydowaliśmy się pobierać je częściej. Dane zapisywane są w HDFS w strukturze: `data/<nazwa_firmy>/<nazwa_firmy>#.avro`, gdzie `#` oznacza kolejny numer pobrania. Program wykorzystuje pomocniczy plik, w którym zapisywana jest liczba dotychczas wykonanych pobrań danych, zdefiniowane są nazwy firm, które śledzimy oraz numer ID ostatniego tweeta dla każdej z nich, na podstawie którego określamy jak długo pobierać tweety, aby ich nie duplikować. Wybraliśmy osiem firm, kurierskich oraz dostarczających żywność, które będziemy śledzić.

Kroki przy tworzeniu programu: Na maszynie, którą posłużymy się do pobierania danych zainstalowaliśmy potrzebne biblioteki Pythona:

```
pip install tweepy
```

```
pip install hdfs[avro]
```

Następnie stworzyliśmy plik konfiguracyjny /.hdfscli.cfg, potrzebny do łączenia się z HDFS. Później został stworzony testowy program zapisujący dane do pliku w formacie .avro do HDFS oraz program odczytujący ten plik. Po uruchomieniu programu zapisującego plik, sprawdziliśmy zawartość HDFS. Wykonaliśmy kolejno polecenia: hdfscli –alias=dev CLIENT.list('test/') CLIENT.status('test/testfile.avro')

Oto ich wyniki:

```
stud@hadoop-worker-1:~/scripts/tests$ hdfscli --alias=dev

Welcome to the interactive HDFS python shell.
The HDFS client is available as CLIENT.

CLIENT.list('test/')
[u'testfile.avro']
CLIENT.status('test/testfile.avro')
{u'group': u'supergroup', u'permission': u'644',
u'blockSize': 134217728, u'accessTime': 1586095573925,
u'pathSuffix': u'', u'modificationTime': 1586095574082,
u'replication': 2, u'length': 283, u'childrenNum': 0,
u'owner': u'stud', u'storagePolicy': 0, u'type': u'FILE',
u'fileId': 16815}
Oraz wynik programu odczytującego ten plik z HDFS:
{u'username': u'user1', u'value': 1, u'title': u'test1'}
{u'username': u'user2', u'value': 2, u'title': u'test2'}
```

Jak widać plik został poprawnie zapisany, istnieje możliwość odczytania go oraz widać, że został on zduplikowany. Gdy wszystko działało poprawnie stworzyliśmy właściwy program.

Obserwowane firmy to: DHL, Uber Eats, FedEx, Grubhub, DoorDash, Postmates, Woolworths.

2.2.6 Stworzenie modelu do analizy sentymentów

Odpowiedzialna: **Marharyta Kruk**

Model został stworzony na zbiorze Twitter Sentiments, zawierającym 1.6 milion tweetów o *niegatywnych*, *pozytywnych* oraz *neutralnych* sentymentach. Model do analizy sentymentów Reddita nie został stworzony z powodu trudności zebrania niezbędnych danych. Model, nauczony na zbiorze tweetow, został przyznany jako przygodny do analizy dowolnych zdań. Zbiór

danych został początkowo przetworzony, zostały wyrzucony stop listy, zostawione stemmery. Wejsciem do sieci służa przetworzone słowa, każde słowo jest przedstawione liczbą. Sieć przedstawia sobą prostą sieć sekwencyjną i składa się z warstw:

- Embedding, podstawą dla której jest nauczony i zapisany model Word2Vec.
- Dropout
- LSTM
- Dense

Na wyjściu otrzymuję się liczba, określająca stopień sentymentu. W algorytmu liczy się, że wynik poniżej 0.4 odnosi się do negatywnego sentymentu, powyżej 0.7 – do pozytywnego, reszta jest określana jako sentyment neutralny. Algorytm osiąga precyzję 80%

2.2.7 Stworzenie programu do pobierania danych z Facebooka

Odpowiedzialni: **Maciej Leszczyński, Aleksei Haidukevich**
Dokonano różnych prób pobierania komentarzy z-pod postów na Facebook, lista narzędzi które probowaliśmy:

- https://github.com/strohne/Facepager (ma tylko graficzną powłokę)
- https://github.com/sriniskanda/Extracts-comments-and-posts-from-facebook
- https://github.com/rugantio/fbcrawl
- https://github.com/piocalderon/facebook-comment-scraper
- https://github.com/weijiekoh/scrape-fb-posts-and-comments
- https://github.com/chribsen/facebook-pages-scraper
- https://github.com/jpryda/facebook-multi-scraper

Nie udało się wyciągnąć dane z różnych powodów (głownie nie są te programy dostosowane do nowych wersji Facebook Graph API, albo po prostu nie działają). Użycie większości z nich oraz ręczna próba wysyłania requestu w sposób opisany w dokumentacji API Facebook kończy się komunikatem o tym, że do pobierania danych z Facebook aplikacja ma przejść przez app review (może trwać miesiące) oraz posiadać "terms of service"i temu podobne rzeczy.

Wniosek po tej pracy: choć komentarze mają duży potencjał pod względem analizy sentymentu, musimy zrezygnować z Facebooka i używać tweetów jako jedyne źródło danych, co z kolei pozwoli skupić się raczej na architekturze systemu i narzędziach do przechowywania i przetwarzania danych.

2.3 Wykonanie zadań z drugiego kamienia milowego

W statucie błędnie założyliśmy, że utworzenie infrasturuktury oraz potoku danych zajmie nam tylko czas na początku projektu. Dzięki wcześniejszemu poznaniu platformy Azure udało nam się stworzyć klaster zdolny przetwarzać nasze dane. Poczyniliśmy również duże postępy w kierunku skończenia potoku danych. Udało nam się również stworzyć model analizy sentymentu. W statucie nie przewidzieliśmy głównie nakładu pracy potrzebnego do wykonania zadań z pierwszego etapu, mocno go zaniżając. Z drugiej strony zakładaliśmy, że na budowę algorytmu potrzebna będzie nam uwaga całego zespołu przez cały drugi etap, co okazało się przecenieniem ilości czasu potrzebnego na wykonanie tego zadania. W kolejnym, ostatnim etapie kluczowe będzie dokończenie przetwarzania danych oraz budowa aplikacji prezentującej nasze wyniki.