POLITECHNIKA WARSZAWSKA

WYDZIAŁ MATEMATYKI I NAUK INFORMACYJNYCH

Metody i systemy analizy Big Data

Raport końcowy z projektu

Analiza platformy Github

Wykonali:
Artur Gajowniczek,
Mateusz Dorobek,
Michał Kosmider,
Stanisław Pawlak

Warszawa 2019

Spis treści

1 Cel projektu

Celem projektu jest wykorzystanie danych udostępnianych przez GitHub do analizy projektów rozwijanych na tej platformie. Jednym z głównych założeń jest opracowanie algorytmu do predykcji sukcesu rozwijanych projektów oraz czasu ich życia. Projekt uznawany jest jako „martwy", jeśli przez ostatnie 3 miesiące nie było żadnego commita.

Obok głównego celu predykcji sukcesu projektu, dodatkowym celem jest analiza struktury projektów i ich członków. Analiza struktury obejmuje analizę udziału języków programowania w projekcie oraz problemów (issues) z nimi związanych. Analiza członków projektu ma charakter eksploracyjny, na przykład wyliczenie „odległości" współpracy między poszczególnymi osobami (liczba Erdosa), mający na celu odkrycie ukrytych zależności w grafie relacji złożonym z repozytoriów, użytkowników i ich aktywności na platformie.

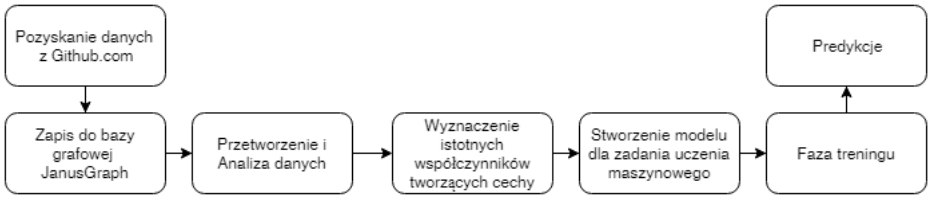
2 Uzasadnienie biznesowe

Mozliwość predykcji pewnych właściwości trendów lub tendencji w projekcie pozwala na monitorowanie rozwoju projektu oraz sygnalizowanie pewnych zdarzeń, co pozwala na minimalizację podejmowanego ryzyka. Analiza issue w projektach githubowych może przynieść takie informacje jak zależność ilości błędów w oprogramowaniu od wybranych technologi. W praktyce może to stanowić podstawę do określenia stosu technologicznego w przyszłych projektach.

Dodatkowo na podstawie analizy sieci połączeń użytkowników (czy na podstawie follow-owania, czy też klikniętych gwiazdek, lub forków, aż po contribution i commity w projektach) można stworzyć graf kontaktów w krórym zauważalne mogą być widoczne klastry.

Obecnie coraz więcej startupów i firm korzysta ze zdalnych repozytoriów dla systemów kontroli wersji. Analiza największego z nich - GitHuba pozwala badać projekty na podstawie historii ich powstawania i dalszego istnienia. Stanowi to szansę na uzyskanie wysokowartościowych informacji dotyczących projektów na podstawie ich repozytoriów.

3 Proces przetwarzania danych



Pierwszym etapem projektu jest pobranie danych z witryny GitHub, które będą stanowić podstawę do fazy analizy i fazy uczenia modelu. Dane zostaną pobrane przez nowe API

wykorzystujące GraphQL. Szczegółowy opis źródła danych opisany jest w sekcji 4.

Dane pobierane są iteracyjnie skryptem w języku Python (sekcja 5) i zapisywane do rozproszonej bazy grafowej *JanusGraph* z konfiguracją opartą o *ElasticSearch* oraz bazę danych *Cassandra* (sekcja 6). Analiza relacji w grafie oraz wyodrębnienie kluczowych cech repozytoriów wykonywane są przy pomocy *Gremlin*a (sekcja 7 i 8).

Wyselekcjonowane cechy poddawane są obróbce (sekcja 9), tak aby można było wykorzystać je do wytrenowania modelu predykcyjnego (sekcja 10).

4 Źródło danych

Aby pobrać dane wykorzystujemy Github GraphQL API v4. Umożliwia ono pobieranie wybranych, określonych fragmentów danych w poszczególnych zapytaniach, co pozwala na bardziej efektywny przesył danych. Istnieje również starsza wersja API v3, wykorzystująca wzorzec REST. Obydwa te interfejsy obarczone są pewnymi ograniczeniami. W przypadku REST, liczba zapytań ograniczona jest do 5000 zapytań/h. Nowszy interfejs ma ograniczenie do 5000 pkt./h. Zapytania nie mogą też być wykonywane równolegle. Samo API ma bardzo przyjazną dokumentację, oraz udostępniony sandbox z GUI do testowania skryptów. GraphQL to język zapytań opracowany przez Facebooka.

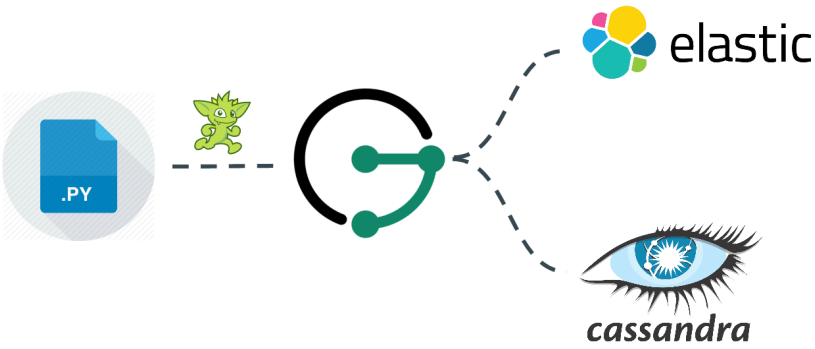
5 Ładowanie danych

Ładowanie danych zostało zrealizowane w postaci skryptu napisanego w Pythonie. Wykonanie rozpoczyna się od zbioru predefiniowanych wierzchołków (na przykład od trzech wybranych repozytoriów). W kolejnych krokach skrypt iteracyjne pobiera wierzchołki powiązane różnorodnymi relacjami z tymi które już znajdują się w bazie danych. W ten sposób odkrywany jest coraz szerszy graf połączeń. Między innymi pobierane są repozytoria, użytkownicy i komentarze. Komunikacja z bazą danych wykorzystuje interfejs do interakcji z bazami grafowymi nazwany *Gremlin* (11). W ten sposób implementacja jest w dużym stopniu niezależna od wykorzystanej bazy danych. Pozwala to na przykład na ułatwioną zamianę bazy danych na inną wspierającą *Gremlin*a.

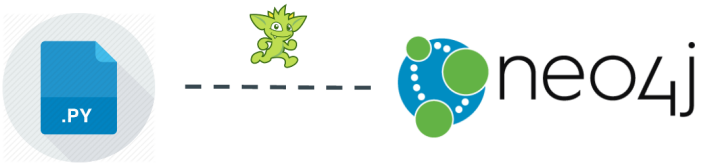
6 Baza danych

Ze względu na grafową strukturę platformy Github oraz aby zapewnić skalowalność rozwiązania wykorzystano rozproszoną bazę grafową *JanusGraph*. Umożliwia ona przechowywanie danych na wielu węzłach klastra, oraz pozwala na transakcyjne wykonywanie skomplikowanych zapytań przechodzenia grafu.

(a) Wersja wykorzystująca tylko *JanusGraph* i *Cassandra*.



(b) Wersja wykorzystująca *ElasticSearch*.



(c) Wersja wykorzystująca *Neo4j*.

Rysunek 1: Trzy wersje metody przechowywania i odpytywania grafu powiązań.

JanusGraph wspiera wiele technologi, dzięki czemu można wedle uznania i potrzeb zmieniać silniki wyszukiwania, wykonywania zapytań, czy backend w postaci bazy danych wykorzystywaną do zapisu danych przez JanusGraph.

W projekcie wykorzystano *JanusGraph* w konfiguracji z *ElasticSearch* oraz bazą danych *Cassandra*. Aby przyspieszyć częste wyszukiwania dodatkowo zaimplementowano skrypt indeksujący w *Groovy*.

Finalna konfiguracja bazy danych jest jednym z trzech podejść jakie zostały zaimplementowane. Pierwotne podejście wykorzystywało jedynie *JanusGraph* skonfigurowany z *Cassandrą*. To rozwiązanie niestety było bardzo wolne i znacząco opóźniało pobieranie danych. W nadziei na przyspieszenie powstała wersją oparta na *Neo4j*, która spowodowała znaczne przyśpieszenie. Wersja która została finalnie wykorzystana skutkowała porównywalną prędkością pobierania grafu. Wszystkie konfiguracje zostały schematycznie przedstawione na diagramie 4.

7 Operacje na grafie

Do operowania na grafie wykorzystano język zapytań *Gremlin*. Bezpośrednia analiza była wykonywana poprze konsolę lub dzięki implementacji *Gremlin*a dla języka Python (*gremlin-python*). Posiada on szerokie możliwości zastosowań, z których wykorzystano jedynie małą część. Do wykorzystanych operacji na grafie należą m. in.:

- wyszukiwanie węzłów przy pomocy etykiet (użytkowników, repozytoriów, języków, komentarzy, issues),
- wyszukiwania wybranych właściwości węzłów i krawędzi (np. firm użytkowników, dat poszczególnych akcji, stosunku wykorzystywanych języków),
- zliczania własności w grafie (np. liczba kontrybutorów, liczba followers, liczba osób śledzących repozytorium),
- określanie odległości węzłów w grafie,
- wyświetlanie ścieżek pomiędzy węzłami.

8 Selekcja cech

Faza analizy polegała przede wszystkim na selekcji cech obecnych w zależnościach grafu, oraz poszczególnych węzłach i krawędziach. Miała na celu wyodrębnienie danych potrzebnych do wytrenowania modelu uczenia maszynowego.

Priorytetowe znaczenie posiadały wszelkie cechy bezpośrednio związane z repozytoriami, np. liczba gwiazdek, wykorzystane technologie, liczba issues, liczba kamieni milowych

czy liczba kontrybutorów. Następnie wybierano cechy związane z węzłami powiązanymi z danym repozytorium jak np. ile osób będących kontrybutorami jest obecnie zatrudniona, czy ilu posiadają *followers*.

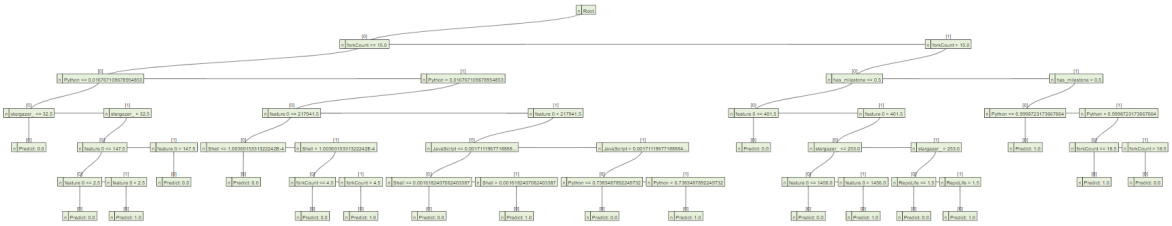
Ostatecznie dla przetworzonych repozytoriów (mających zaciągnięte wszystkie zależności) uzyskano ponad 200 cech, z których znaczącą część stanowiły wielkości dotyczące wykorzystywanych języków programowania.

9 Przygotowanie danych

Dane wymagały wstępnego przygotowania przed ich użyciem w modelu. Pierwszym krokiem było ich oczyszczenie. Przede wszystkim było to usunięcie kolumn wykorzystywanych na etapie pobierania danych i operacji na grafie. Następnie usunięte zostały wadliwe rekordy (brakujące wartości w kolumnie z datą ostatniego commita, gdyż uniemożliwia to utworzenie zmiennej objaśnianej). Dodatkowo oczyszczenia wymagały rekordy, które reprezentowały forki repozytoriów, gdyż jeżeli został utworzony fork repozytorium, a nie został dodany do niego żaden commit, to data ostatniego commita była wcześniejsza niż data utworzenia repozytorium. Utworzyliśmy binarną zmienną objaśnianą (Określającą czy wystąpił commit przez ostatnie 3 miesiące). Wybraliśmy 10 najczęściej występujących języków, a pozostałe (150 języków) przenieśliśmy do kolumny *OtherLanguages*. Połączyliśmy też kolumny które opisywały podobne cechy (np.: *contributed-to-created*, czy *contributed-to-follows* i jeszcze kilka innych tego typu wybierając maksimum z tych kolumn). Ostatnim krokiem było utworzenie zmiennych binarnych dla niektórych zmiennych ciągłych o mocno skośnym rozkładzie. Wykorzystaliśmy typowe narzędzia do przetwarzania dostępne dla języka Python.

10 Uczenie maszynowe

Do stworzenia modelu użyliśmy PySpark. Algorytm który wykorzystaliśmy do predykcji to *DecisionTreeClassifier*. Osiągneliśmy wynik *Accuracy = 90%*. Ze względy na nierównomierny rozkład zmiennej objaśnianej miara precyzji nie jest zbyt informatywna dlatego zmierzyliśmy też AUC dla krzywej ROC które wyniosło 0.85 co oznacza stosunkowo dobry wynik naszej klasyfikacji.

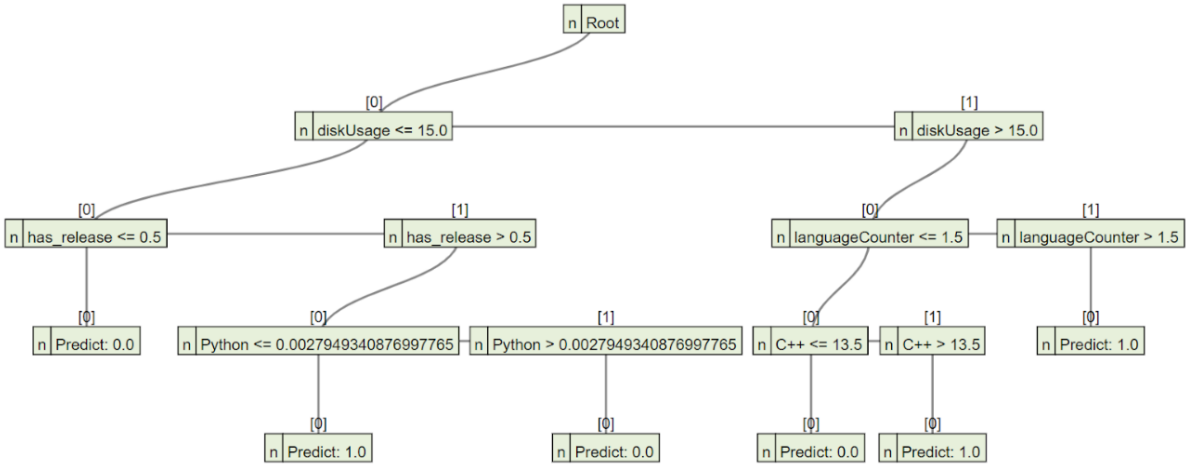


Rysunek 2: Wynik klasyfikacjiz wykorzystaniem Decyzyjnego Drzewa Klasyfikacyjnego.

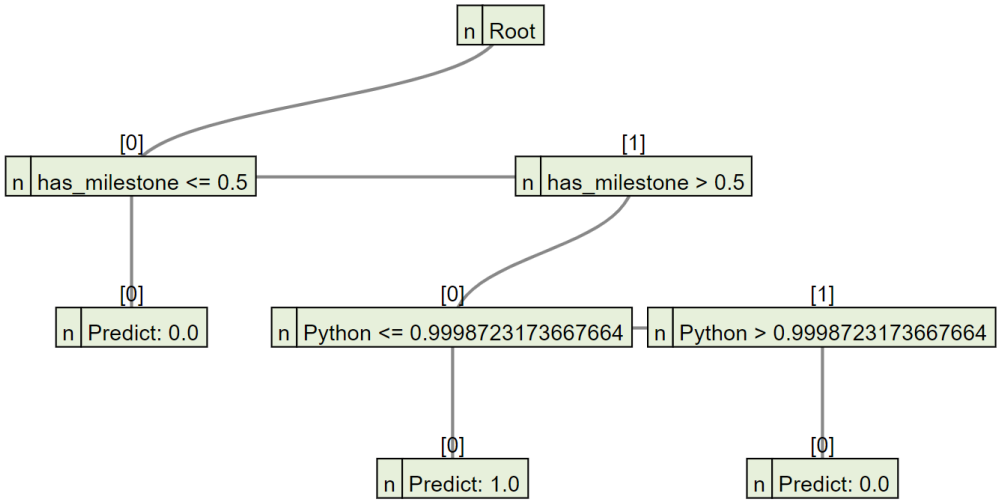
Wnioski jakie udało się wyciągnąć z analizy drzew klasyfikacyjnych są następujące.

- Duże znaczenie miała ilość forków - im ich jest więcej tym większa szansa, ze repozytorium jest aktywne i ma wysokie prawdopodobieństwo na commit w najbliższym czasie,
- Jeżeli w repozytorium występuje jeden język programowania (wartość 1.0) szansa na commit jest niższa niż w repozytoriach, gdzie języków jest więcej. Wniosek jaki ,może z tego płynąć jest taki, że repozytoria z jednym językiem programowania często są nieduże i projekt z jakim były związane został zakończony.
- Projekty w których były milestony często miały większą szansę na przetrwanie.
- Im większy rozmiar w pamięci zajmowały repozytoria tym większa szansa na to że

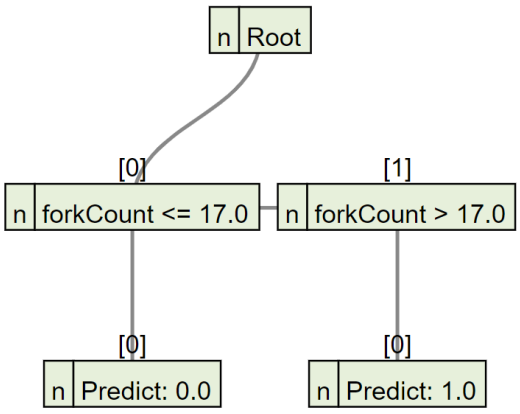
Pierwszy wytrenowany został model przy wykorzystaniu algorytmu XGBoost. Wynik jaki osiągnął w zadaniu klasyfikacji był znacznie niższy niż finalny algorytm. XGBoost został zewaluowany wykorzystując metryki F1 score (wartość 0.37) oraz balanced accuracy (62%). Poniżej zamieszczone są drzewa klasyfikacjyjne uzyskane poprzez zastosowanie mniejszych wartości parametru *maxDepth*.



(a) Wpływ rozmiaru na dysku na wynik klasyfikacji.

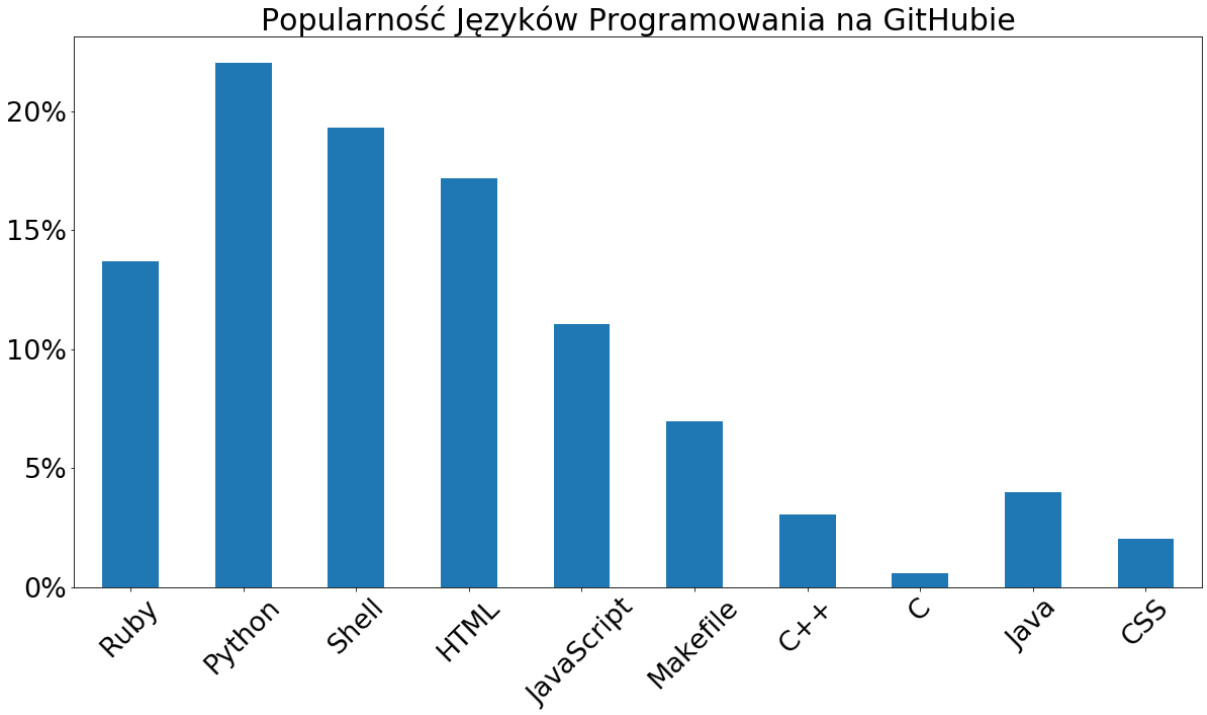


(b) Dla drzewa o głębokości 2 obecność milestonów w większości decydowała o wyniku klasyfikacji.

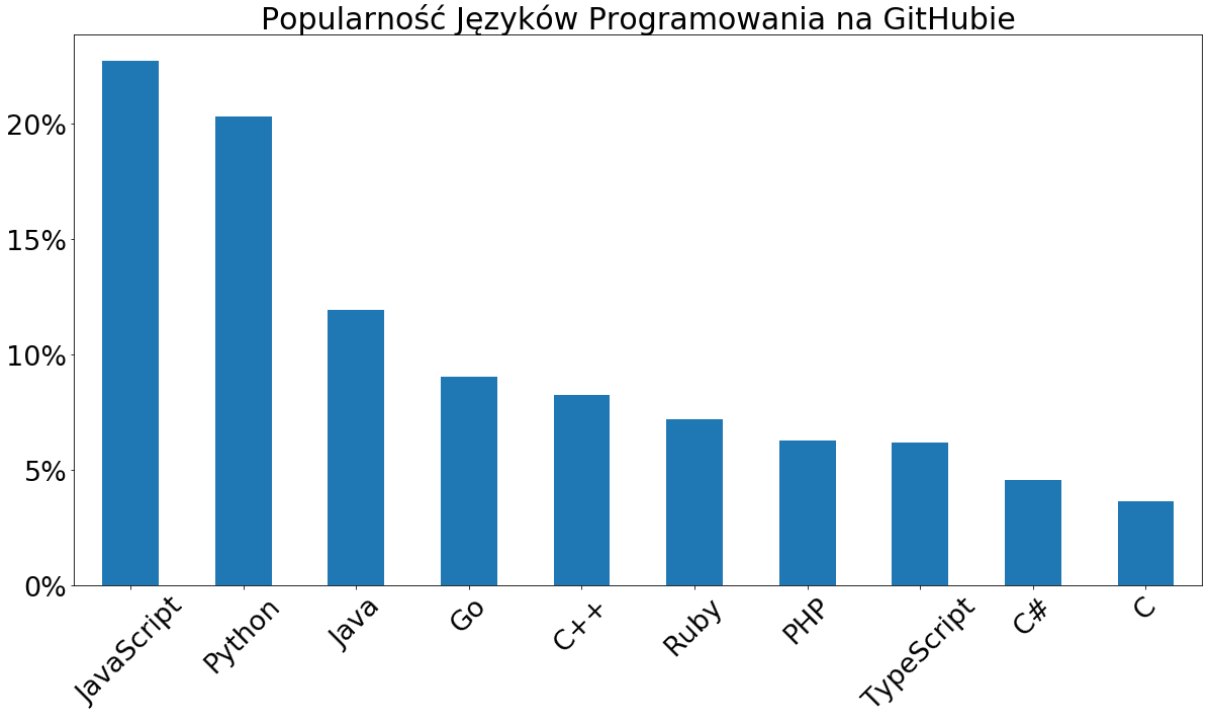


(c) Im więcej forków tym większa szansa na commit.

Rysunek 3: Porównanie wpływu cech na klasyfikację dla drzew o różnej głębokości



(a) Wyniki dla naszego zbioru pobranych repozytori



(b) Prawdziwa dystrybucja języków w GitHubie.

Rysunek 4: Procentowy udział różnych języków programowania na GitHubie.

Widać że faktyczną przewagę ma JavaScript. Różnica wynika z tego, że nasze dane były pobierane w sposób rekurencyjny poczynając od repozytorium TensorFlow - frameworku, który jest przeznaczony głównie do Pythona. Statystyczną poprawność można uzyskać poprzez losowe wybieranie repozytorii - korzeni do naszego skryptu pobierającego dane.

11 Wykorzystywane narzędzia

Do realizacji celu projektu postanowiliśmy posłużyć się następującymi narzędziami:

- JanusGraph – do przechowywania danych,
- ElasticSearch – do przeszukiwania grafu,
- Cassandra – do przechowywania sparsowanych danych,
- TinkerPop (Gremlin) – do przetwarzania zapytań do grafowego zbioru danych,
- PySpark – biblioteka Pythonowa do rozproszonego uczenia maszynowego.

12 Opis podziału pracy

W trakcie trwania projektu można było wyróżnić 4 obszary pracy:

- Konfiguracja maszyny wirtualnej oraz środowiska pracy
- Skrypt do pobierania danych
- Wstępna selekcja cech
- Przetwarzanie danych i moduł analityczny

Konfiguracją maszyny wirtualnej oraz środowiska pracy w chmurze (Azure) zajmował się przede wszystkim Stanisław Pawlak. Autorem algorytmu pobierania danych oraz architektury bazy danych jest Michał Kośmider. Skrypt do wstępnej selekcji cech został napisany przez Stanisława Pawlaka. Zapytania do API GitHuba stworzył Mateusz Dorobek. Autorami skryptu do przetwarzania danych oraz Machine Learningowego modułu analitycznego są Mateusz Dorobek oraz Artur Gajowniczek.

13 Wnioski

- Narzędzia Big-Data są często lepiej przystosowane do współpracy z językami działającymi na maszynie wirtualnej Javy. Na przykład niektóre operacje mogą nie być możliwe do wykonania używając jedynie Pythona. Dlatego do pracy przy projektach Big-Data preferowane są języki takie jak Java, Scala czy Groovy,
- Pobieranie danych z Githuba okazało się bardzo skomplikowanym i czasochłonnym zajęciem ze względu na dużą liczbę błędów, trudną selekcję pobieranych węzłów oraz monitorowanie ich wielkości, Podczas tworzenia tego projektu natrafiliśmy na wiele nieoczekiwanych błędów w API *Githuba*. Część z nich została zgłoszona przez oficjalne kanały, a część była już znana i jeszcze nie naprawiona.

- Wykorzystanie *Gremlin*a jako interfejsu pozwoliło na prostsze zmiany konfiguracji baz danych, co umożliwiło przetestowanie wielu różnych rozwiązań,
- Dane należało poddać długiemu procesowi obróbki i selekcji, aby wykorzystać je skutecznie do uczenia maszynowego,
- Wykorzystanie zasobów chmurowych (Azure HDP 2.4 Sandbox) pozwoliło pokonać problemy wynikające z niedostatecznych zasobów do przetwarzania dużych zbiorów danych kosztem długiego czasu konfiguracji – wykorzystywaliśmy technologie pierwotnie nie wchodzące w skład Sandboxa,
- Wykorzystanie *ElasticSearch* pozwoliło na znacznie szybsze pobieranie danych,
- Przeprowadzenie uczenia maszynowego na uzyskanych danych okazało się trudne, jednak można przypuszczać że gdyby dane były zbierane periodycznie przez dłuższy okres czasu to poprzez posiadanie informacji w różnych chwilach czasowych możliwości byłyby znacznie większe.
- Aby wykonać predykcję zmiennej z przyszłości należy posiadać historyczny stan i aktualną wartość predykowanej zmiennej. W naszym przypaku okazało się to niemożliwe, ponieważ GitHub API pozwala na pobranie jedynie aktualnego stanu repozytorium.
- Wiele z pobranych danych okazały się mieć stałą wartość dla wszystkich repozytoriów, co spowodowało niską użyteczność tych danych dla uczenia maszynowego.
- GitHubowe API w postaci języka zapytań GraphQL okazało się bardzo przyjemne do pracy. Na stronie z dokumentacją znajduje się webowy sandbox do testowania zapytań.

Literatura

[1] https://hadoop.apache.org/

[2] https://developer.github.com/v4/

[3] https://janusgraph.org/

[4] http://tinkerpop.apache.org/

[5] https://hortonworks.com/

[6] https://www.tensorflow.org/

[7] https://spark.apache.org/