Przetwarzanie języka naturalnego

Uczenie maszynowe lub sztuczna inteligencja opiera się na danych, które mogą być ustrukturyzowane lub nieustrukturyzowane. Przetwarzanie języka naturalnego (NLP) to dziedzina algorytmów skoncentrowana na przetwarzaniu nieustrukturyzowanych danych. Ta część koncentruje się na nieustrukturyzowanych danych w formacie tekstowym w języku naturalnym. Organizacje zawsze mają duże korpusy nieustrukturyzowanych danych tekstowych, w postaci dokumentów słownych, plików PDF, treści wiadomości e-mail lub dokumentów internetowych. Dzięki postępowi technologicznemu organizacje zaczęły polegać na dużych ilościach informacji tekstowych. Na przykład firma prawnicza ma wiele informacji w postaci papierów dłużnych, umów prawnych, nakazów sądowych, dokumentów prawnych i tak dalej. Takie zasoby informacyjne składają się z informacji tekstowych, które są specyficzne dla domeny (w tym przypadku legalne). Jest to konieczne aby wykorzystać te cenne zasoby tekstowe i przekształcić informacje w wiedzę, potrzebujemy inteligentnych maszyn, aby mogły zrozumieć tekst w obecnej postaci, bez żadnej interwencji człowieka. NLP dla dużych zbiorów danych używa ton danych tekstowych z różnych źródeł, aby określić relacje i wzorce w treściach otrzymanych z tych źródeł. Pomaga w identyfikacji trendów, które zostaną wykorzystane w przypadkach użycia, takich jak silniki rekomendacji. W tym rozdziale przedstawiono podstawowe pojęcia związane z NLP wraz z praktycznymi przykładami. Możemy podzielić NLP na dwa rodzaje podejść, nadzorowane NLP i NLP bez nadzoru. Podejście NLP do nauki nadzorowanej polega na zastosowaniu algorytmów nauki nadzorowanej, takich jak Naive Bayes i Random Forests. W tych algorytmach modele są tworzone na podstawie przewidywanych danych wyjściowych przekazanych im w celu szkolenia zestawu danych wejściowych. Oznacza to, że nadzorowane metody uczenia się nie są samouczące się, lecz trenują i dostosowują modele w oparciu o dostarczone im docelowe wyniki. Algorytmy uczenia bez nadzoru nie opierają się na tym, że dostarczono im docelowy wynik szkolenia modelu. Wyciągają wnioski z danych wejściowych, z rekordów podanych im w wyniku wielu iteracji w porównaniu do uczenia się danych z wyników poprzednich iteracji oraz dostrajania wag i parametrów w celu optymalizacji wyników. Nawracające sieci neuronowe (RNN) są jednym z powszechnych algorytmów uczenia się bez nadzoru stosowanych w środowisku naturalnym przetwarzanie języka. Omówimy wszystkie te techniki. Ogółem omówimy następujące tematy:

* Podstawy przetwarzania języka naturalnego

* Wstępne przetwarzanie tekstu

* Ekstrakcja funkcji

* Stosowanie technik NLP

* Wdrażanie analizy sentymentów

Podstawy przetwarzania języka naturalnego

Zanim przedstawimy niektóre z głównych kroków związanych z NLP, ważne jest ustalenie definicji NLP. Mówiąc najprościej, NLP to zbiór procesów, algorytmów i narzędzi wykorzystywanych przez inteligentne systemy do interpretacji danych tekstowych zapisanych w języku ludzkim w celu uzyskania praktycznych informacji. Wzmianka o danych tekstowych czyni jeden fakt dotyczący NLP bardzo oczywistym. NLP polega na interpretacji nieustrukturyzowanych danych. NLP organizuje nieustrukturyzowane dane tekstowe i stosuje wyrafinowane metody rozwiązywania wielu problemów, takich jak analiza sentymentów, klasyfikacja dokumentów i podsumowanie tekstu. W tej sekcji omówimy niektóre podstawowe kroki związane z NLP. W kolejnych sekcjach zajmiemy się tymi krokami.

* Rodzaj uczenia maszynowego: NLP można wykonać przy użyciu nadzorowanych algorytmów uczenia się lub jako algorytmy uczenia się bez nadzoru. Nadzorowane algorytmy uczenia obejmują Naive Bayes, SVM i Random Forest. Algorytmy uczenia bez nadzoru obejmują sieci neuronowe Feed Forward (Multi Layer Perceptron) i Recurrent Neural Network (RNN). Należy tutaj zauważyć, że etapy wstępnego przetwarzania i ekstrakcji funkcji są takie same dla obu klas algorytmów. Różni się tym, jak trenujesz swój model. Uczenie się pod nadzorem wymaga danych wyjściowych oznaczonych jako dane wejściowe, a uczenie się bez nadzoru przewidywałoby wynik bez żadnych oznakowanych wyników.

* Przetwarzanie wstępne : Ten krok jest wymagany, ponieważ surowy tekst naturalny nie może być używany w systemach NLP. Spowoduje to złe lub niezbyt dokładne dane wyjściowe. Niektóre z typowych kroków wstępnego przetwarzania tekstu obejmują usuwanie słów stop, zastępowanie wielkich liter i usuwanie znaków specjalnych. Innym częstym krokiem w przetwarzaniu tekstu jest część tagowania mowy, zwana również adnotacją. Stosowana jest również normalizacja tekstu w postaci wyprowadzania i lematyzacji.

* Wyodrębnianie funkcji: Aby dowolny algorytm ML działał na tekście, teksty te muszą zostać przekonwertowane na jakąś formę wprowadzania numerycznego. Wyodrębnianie cech wykorzystuje popularne techniki przekształcania tekstu wejściowego na numeryczny w postaci wektorów.

* Szkolenie modelowe: Szkolenie modelowe to proces ustanawiania lub znajdowania funkcji matematycznej, której można użyć do przewidywania wyniku na podstawie danych wejściowych. Proces znajdowania funkcji wymaga wielu iteracji i dostrajania parametrów.

* Weryfikacja modelu: ten etap polega na weryfikacji modeli wynikających z procesu szkolenia modeli. Zasadniczo dzielisz swój zestaw danych szkoleniowych na stosunek 80:20. 80% danych jest wykorzystywanych do szkolenia modeli, a 20% danych służy do sprawdzania poprawności modelu. W przypadku rozbieżności dokładnie dostosuj kroki tworzenia modelu i ponownie uruchom weryfikacje.

* Wdrażanie modeli i interfejsy API: Po zweryfikowaniu modeli wdrażasz swoje modele, dzięki czemu można ich używać do przewidywania wyników w kontekście aplikacji korporacyjnych. Możesz zapisać te modele w miejscu przechowywania, w którym można je odczytać w pamięci i można je zastosować do zestawu danych, aby przewidzieć jego wynik. Podczas przetwarzania rozproszonego są one na ogół zapisywane w rozproszonym systemie plików Hadoop, aby procesy wsadowe Hadoop mogły odczytać i zastosować te modele. W przypadku aplikacji internetowych są one przechowywane w postaci plików marynowanych w Pythonie, a te pliki marynowane są odczytywane i przetwarzane przy każdym żądaniu prognozy. Chociaż, aby aplikacje mogły z niego korzystać, trzeba by na nim nałożyć warstwy API. Te warstwy API mogą być spokojnymi interfejsami API lub mogą mieć postać spakowanych słoików rozmieszczonych w miejscu, w którym hostowane są aplikacje. Kiedy Interfejsy API są widoczne, mogą być używane przez różne aplikacje internetowe, aplikacje mobilne, mechanizmy analityczne lub BI.

Wstępne przetwarzanie tekstu

Wstępne przetwarzanie danych to proces czyszczenia i przygotowywania tekstu do klasyfikacji i wyprowadzania znaczenia. Ponieważ nasze dane mogą zawierać dużo szumu, nieistotne części, takie jak tagi HTML, muszą zostać wyeliminowane lub ponownie dostosowane. Na poziomie słów może istnieć wiele słów, które nie mają większego wpływu na ogólną semantię kontekstu tekstowego. Wstępne przetwarzanie tekstu składa się z kilku etapów, takich jak wyodrębnianie, tokenizacja, usuwanie słów zatrzymanych, wzbogacanie tekstu oraz normalizacja z pobieraniem i lematyzacją. Oprócz tych, niektóre podstawowe i ogólne techniki poprawiające dokładność obejmują konwersję tekstu na małe litery, usuwanie liczb (w zależności od kontekstu), usuwanie interpunkcji, usuwanie białych spacji (czasami zwiększają szum w sygnale wejściowym), i eliminując rzadkie warunki, które są

rzadkimi terminami w dokumencie. W kolejnych sekcjach szczegółowo przeanalizujemy niektóre z tych technik.

Usuwanie słów stop

Słowa zatrzymane to słowa występujące częściej w zdaniu, które powodują, że tekst jest cięższy i mniej ważny dla analizy, należy je wyłączyć z wprowadzania. Posiadanie słów stop w tekście dezorientuje algorytm, ponieważ te słowa stop nie mają kontekstowego znaczenia i nie zwiększają cech wymiarowych wektorów terminów. Dlatego konieczne jest usunięcie tych słów stop, aby uzyskać większą dokładność modelu. Przykłady słów stop to i, am, is . Jednym ze sposobów usunięcia słów końcowych jest utworzenie wstępnie skompilowanej listy słów końcowych, a następnie usunięcie tych słów końcowych z dokumentu (tekst używany do szkolenia modelu).

Dzięki Spark możemy korzystać z biblioteki StopWordsRemover, która ma własną listę domyślnych słów kluczowych dla wielu języków naturalnych. Możemy również dostarczyć listę słów kluczowych z parametrem stopWords. Inny sposób usunięcia mniej znaczących słów z dokumentu opiera się na ich częstotliwości występowania; jeśli częstotliwość tego słowa jest niska, możemy usunąć te słowa, jest to również znane jako przycinanie. Oto przykładowy kod do korzystania z biblioteki Spark. Dzięki tej bibliotece proces usuwania słów zatrzymujących jest zrównoleglony i możemy szybko wykonać usuwanie słów zatrzymanych na dużej ilości danych w sposób rozproszony:

```
import java.util.Arrays;

import java.util.List;

import org.apache.spark.ml.feature.StopWordsRemover;

import org.apache.spark.sql.Dataset;

import org.apache.spark.sql.Row;

import org.apache.spark.sql.RowFactory;

import org.apache.spark.sql.types.DataTypes;

import org.apache.spark.sql.types.Metadata;

import org.apache.spark.sql.types.StructField;

import org.apache.spark.sql.types.StructType;

StopWordsRemover remover = new StopWordsRemover()

.setInputCol("raw")

.setOutputCol("filtered");

List<Row> data = Arrays.asList(

RowFactory.create(Arrays.asList("I", "am", "removing", "the", "stop",

"words")),

RowFactory.create(Arrays.asList("from", "a", "large", "volume",

"of","data"))

);
```

```
StructType schema = new StructType(new StructField[]{

new StructField(

"raw", DataTypes.createArrayType(DataTypes.StringType), false,

Metadata.empty())

});

Dataset<Row> dataset = spark.createDataFrame(data, schema);

remover.transform(dataset).show(false);
```

Stemming

Różne formy słowa często przekazują zasadniczo to samo znaczenie. Rozważ przykład wyszukiwarki, gdy użytkownik wyszukuje buty lub gdy szuka butów. Intencja użytkownika jest taka sama, a wynikiem wyszukiwania nadal będą buty różnych marek. Ale obecność obu słów może mylić modele. Aby uzyskać większą dokładność, musimy przekonwertować te różne formy słowa w formacie wiersza. Stemming konwertuje słowo w tekście na format surowy. Na przykład wprowadzanie, wprowadzanie i wprowadzanie wszystkich przekształca się w wprowadzać po zrobieniu. Celem tej metody jest usunięcie różnych sufiksów w celu zmniejszenia liczby słów. Pomaga to również modelowi uniknąć nieporozumień podczas szkolenia. Dostępnych jest wiele algorytmów generujących, takich jak porter, snowball i Lancaster. Większość algorytmów pochodnych w poniższych sekcjach jest dostępna w wielu językach naturalnych.

Porter stemming

Porter jest jedną z form algorytmu, który usuwa sufiksy ze słów podstawowych lub terminów ze słownika angielskiego. Głównym celem Porter Stemmer jest poprawą wydajności ćwiczenia modelowego NLP. Czyni to, usuwając przyrostki ze słowa i doprowadzając je do swojej podstawowej formy. W ten sposób liczba terminów jest zmniejszona, a ślad pamięci i złożoność przestrzeni terminów są również zminimalizowane. Porter nie jest oparty na słowniku. Nie używa żadnego słownika rdzeni do identyfikowania sufiksów, które należy usunąć. Opiera się na zestawie ogólnych zasad. Niektóre osoby postrzegają to jako wadę, ponieważ jego działanie jest dość proste i nie zajmuje się drobiazgowym kontekstem drobiazgowości angielskich słów. Porter jest używany dla jego prostoty i szybkości. Porter ma pięć kroków, które są stosowane do słowa, dopóki jeden z nich nie spełni. Rozważmy na przykład krok 1 w porterze, który jest wyjaśniony w następujących blokach:

SSESS -> SS - This rule converts SSESS suffix of the word into SS.

For example, prepossess - > preposs

IES -> I - This rule converts IES suffix of the word into I.

For example, ties -> ti

SS -> SS - If the word has SS as suffix this won’t change.

For example, Success -> Success

S -> - If the word has S as suffix this would remove the suffix.

For example, Pens -> Pen

Snowball stemming

Jest to również znane jako Porter2. Algorytm Porter2 jest zaimplementowany jako angielski Stemmer (oparty na Snowball). Algorytm ten został opracowany jako struktura do użycia w językach innych niż angielski. Jest to dokładniejsze niż algorytmy portera. Przykład reguły śnieżki podano w następujący sposób:

ied or ies -> replace by i if preceded by more than one letter, otherwise

by ie.

ties -> tie,

cries -> cri

So as we can see with porter ties we stemmed into ti whereas with snowball it becomes tie.

Lancaster stemming

Bardzo agresywny algorytm wyprowadzający, czasem do błędu. W przypadku portera i kuli śnieżnej reprezentacje oparte na rdzeniach są zazwyczaj dość intuicyjne dla czytelnika, a nie w przypadku Lancastera, ponieważ wiele krótszych słów zostanie całkowicie zaciemnionych. Najszybszy algorytm tutaj, znacznie zmniejszy Twój zestaw słów, ale jeśli chcesz więcej rozróżnić, nie jest to narzędzie do użycia. Przykład reguły Lancaster podano w następującym bloku:

ies -> y - Ta reguła przekształca przyrostek słowa w y.

płacze -> płacze

Więc z Lancasterem pojawiającym się, gdy widzimy płacz, który stał się bardziej płaczem.

Lovins

W 1968 r. Lovins JB opublikował ten algorytm. Podejście Lovinsa jest nieco inne, ale zaczyna się od usunięcia przyrostków ze słowa. Dochodzi do wniosku w dwuetapowym procesie. Najpierw usuwa najdłuższy możliwy przyrostek ze słowa. To jest inglepass algorytm, który usuwa pojedynczy największy sufiks ze słowa. Po drugie, stosuje zestaw reguł dotyczących najdłuższego przyrostka, aby przekształcić go w słowo. Algorytm ten oparty jest na regułach i słownikach. Jest szybszy i zwykle wymaga mniej pamięci. Jest w stanie konwertować słowa takie jak wchodzenie w get lub słowa takie jak mysz do myszy. Czasami ten algorytm może być niedokładny z powodu wielu sufiksów niedostępnych w słowniku. Co więcej, często nie tworzy słowa ze słowa wywodzącego się ze słowa lub nawet jeśli słowo jest tworzone, może nie mieć takiego samego znaczenia jak słowo oryginalne.

Dawson stemming

Ten stemmer rozszerza to samo podejście co stempel Lovinsa z listą ponad tysiąca sufiksów w języku angielskim. Oto ogólny algorytm dla stemmera Dawson:

1. Uzyskaj słowo wejściowe

2. Uzyskaj pasujący sufiks

2a. Pula sufiksów jest indeksowana odwrotnie według długości

2b. Pula sufiksów jest indeksowana do tyłu przez ostatni znak

3. Usuń najdłuższy przyrostek ze słowa z dokładnym dopasowaniem.

4. Przekoduj słowo za pomocą tabeli mapowania

5. Konwertuj rdzeń na prawidłowe słowo.

Zalety stemmer Dawson są następujące:

*Obejmuje szerszy zakres sufiksów, a tym samym zapewnia dokładniejszy wyjściowy wynik.

*Jest to algorytm jednoprzebiegowy, dzięki czemu jest wydajny

Lemmatyzacja

Lemmatyzacja różni się nieco od tworzenia. Pytanie zazwyczaj usuwa znaki końcowe ze słowa, oczekując, że otrzymają prawidłowe słowo podstawowe. Czasami jednak powoduje usunięcie sufiksów, które dodają znaczenia słowu. Lemmatyzacja próbuje przezwyciężyć to ograniczenie rodzenia. Próbuje znaleźć bazę formy słowa, zwana lematem, oparta na słownictwie słów, które ono posiada, oraz na analizie morfologicznej słów. Korzysta ze słownika wiedzy leksykalnej WordNet, aby uzyskać prawidłową podstawową formę słowa. Ma to jednak swoje ograniczenia, na przykład wymaga tagowania części mowy, w przeciwnym razie potraktuje wszystko jak rzeczownik.

N-gramy

N-gram to ciągła sekwencja N-słów lub tokenów w danym zdaniu lub ciągła sekwencja tekstu. N definiuje się jako wartość całkowitą rozpoczynającą się od 1. Tak więc, N-Gram może być Uni-Gram (N = 1), Bi-Gram (N = 3) lub Tri-Gram (N = 3). Algorytmy lub programy N-gram identyfikują wszystkie ciągłe sąsiednie sekwencje słów w danym tokenie zdania. Jest to funkcjonalność oparta na systemie Windows, zaczynająca się od lewej pozycji słowa, a następnie przesuwając okna o jeden krok. Zobaczmy to z przykładowym zdaniem: To jest książka AI Big Data. Zobacz następujący przykład przykładów Uni-Gram, Bi-Gram i Tri-Gram:

N-gramów używa się do opracowywania wydajnych funkcji, które są przekazywane do nadzorowanych modeli zarabiania na maszynach, takich jak SVM i Naive Bayes, do szkolenia i prognozowania. Chodzi o użycie tokenów, takich jak Bi-Gramy, zamiast tylko Uni-Grams, aby te modele uczenia maszynowego mogły się efektywnie uczyć. Używanie N-gramów ma tendencję do przechwytywania kontekstu, w którym słowa są używane razem w danym dokumencie. Jak pokazano w poprzednim przykładzie, Tri-Gramy mogą nadać Twojemu algorytmowi uczenia maszynowego szerszy kontekst, dzięki czemu można lepiej przewidzieć następny zestaw słów. Jednak, jaka powinna być optymalna wartość N, należy ustalić na podstawie zestawu danych i po przeprowadzeniu wystarczającej eksploracji i analizy danych. Większa wartość N nie zawsze oznacza lepszy wynik. Powinieneś podejmować bardzo świadome decyzje dotyczące wartości N.

Ekstrakcja funkcji

Jak wspomniano wcześniej, system NLP nie rozumie wartości ciągów. Potrzebują danych numerycznych do budowy modeli, czasem nazywane są również funkcjami numerycznymi. Wyodrębnianie funkcji w NLP polega na konwertowaniu zestawu informacji tekstowych na zestaw funkcji numerycznych. Każdy algorytm uczenia maszynowego, który zamierzasz trenować, potrzebowałby funkcji w liczbowych formach wektorowych, ponieważ nie rozumie ciągu. Istnieje wiele sposobów przedstawienia tekstu jako wektorów numerycznych. Niektóre z takich sposobów to One hot encoding, TF-IDF, Word2Vec i CountVectorizer.

Jedno gorące kodowanie

Jednym z kodowań na gorąco jest binarna rzadka wektorowa reprezentacja tekstu. W tym kodowaniu wynikowy wektor binarny ma wartość zerową, z wyjątkiem pozycji lub indeksu tokena, w którym jest jeden. Spójrzmy na to z przykładem. Załóżmy, że są dwa zdania: To jest książka AI Big Data. Ta książka wyjaśnia algorytmy AI na Big Data. Unikatowymi tokenami (rzeczownikami) dla wcześniejszych zdań byłyby {dane, AI, książka, algorytmy}. Jedna reprezentacja kodowania na gorąco dla tych tokenów wyglądałaby następująco:

	$data$	AI	$book$	$Algorithms$
$data$	1	0	0	0
AI	0	1	0	0
$book$	0	0	1	0
$Algorithms$	0	0	0	1

Kodowana rzadka reprezentacja wektorowa wyglądałaby następująco:

$$data = [1, 0, 0, 0]$$
$$AI = [0, 1, 0, 0]$$
$$book = [0, 0, 1, 0]$$
$$Algorithms = [0, 0, 0, 1]$$

TF-IDF

Metoda ekstrakcji cech TF-IDF wykorzystuje iloczyn skalarny częstotliwości terminu (TF) i odwrotnej częstotliwości dokumentu (IDF) do obliczenia wektora liczbowego tokena lub terminu. TFIDF nie tylko oblicza znaczenie słowa w określonym dokumencie, ale także mierzy jego znaczenie w innych dokumentach korpusu. Co więcej, próbuje znormalizować każde słowo, które jest zbyt częste w całym korpusie. TF lub Termin Częstotliwość to występowanie terminu w dokumencie. Możemy użyć biblioteki HashingTF w Spark do obliczenia częstotliwości tego terminu. HashingTF tworzy rzadki wektor odpowiadający każdemu dokumentowi reprezentującemu indeks i częstotliwość. Na przykład, jeśli weźmiemy pod uwagę wyodrębnienie funkcji za pomocą ekstrakcji HashingTF w ciąg tekstowy metody, a następnie TF każdego słowa we wcześniejszym dokumencie używającym HashingTF byłby następujący:

```scala
import org.apache.spark.ml.feature.{HashingTF, IDF, Tokenizer}

val exampleData = spark.createDataFrame(Seq(
  (0.0, "extraction of the feature using HashingTF extraction method")
)).toDF("label", "sentence")

val tokenizer = new Tokenizer().setInputCol("sentence").setOutputCol("words")
val tokensData = tokenizer.transform(exampleData)

val hashingTF = new HashingTF()
  .setInputCol("words").setOutputCol("rawFeatures").setNumFeatures(10)
val features = hashingTF.transform(tokensData)
features.select("rawFeatures").show(truncate=false)
```

Dane wyjściowe HashingTF:

```
+-----------------------------------------+
|rawFeatures                              |
+-----------------------------------------+
|(10,[0,3,4,5,6,8],[1.0,1.0,2.0,1.0,1.0,2.0])|
+-----------------------------------------+
```

Na poprzednim zrzucie ekranu widzimy, że pierwsza tablica to wyodrębnione funkcje z dokumentu, a druga tablica to Array [SparseVector], która reprezentuje indeks i częstotliwość. Na przykład słowo ekstrakcyjne występuje dwukrotnie w dokumencie, dzięki czemu możemy zobaczyć, że częstotliwość tego słowa wynosi 2. W przypadku HashingTF tablica słów tokenizowanych może nie być w tej samej sekwencji co tablica wektorowa. TF mierzy znaczenie słowa tylko w określonym dokumencie, a nie w odniesieniu do całego zbioru dokumentów. Co więcej, zbyt częste słowa w dużym dokumencie mogą nie być tak ważne w odniesieniu do całego korpusu. Może to utrudnić prognozowanie, ponieważ słowa, które pojawiają się rzadziej, mogą mieć większe znaczenie w odniesieniu do całego korpusu. Tutaj pojawia się IDF; reprezentuje odwrotność udziału dokumentów, w których można znaleźć rozważany termin. Im mniejsza liczba zawartych dokumentów w stosunku do wielkości korpusu, tym wyższy współczynnik. Powodem, dla którego ten stosunek nie jest stosowany bezpośrednio, ale zamiast jego logarytmu, jest to, że w przeciwnym razie skuteczna kara za punktowanie w dwóch dokumentach byłaby zbyt ekstremalna. Oto przykładowy przykład wspólnego obliczania TF-IDF:

```scala
import org.apache.spark.ml.feature.{HashingTF, IDF, Tokenizer}

val exampleData = spark.createDataFrame(Seq(
  (0.0, "extraction of the feature using HashingTF extraction method")
)).toDF("label", "sentence")

val tokenizer = new Tokenizer().setInputCol("sentence").setOutputCol("words")
val tokensData = tokenizer.transform(exampleData)

val hashingTF = new HashingTF()
  .setInputCol("words").setOutputCol("TF").setNumFeatures(10)
val features = hashingTF.transform(tokensData)
val idf = new IDF().setInputCol("TF").setOutputCol("IDF")
val idfModel = idf.fit(features)
val rescaledData = idfModel.transform(features)
rescaledData.select("label", "TF","IDF").show(truncate=false)
```

Dane wyjściowe kodu IDF są następujące:

```
+-----+-----------------------------------------+-----------------------------------------+
|label|TF                                       |IDF                                      |
+-----+-----------------------------------------+-----------------------------------------+
|0.0  |(10,[0,3,4,5,6,8],[1.0,1.0,2.0,1.0,1.0,2.0])|(10,[0,3,4,5,6,8],[0.0,0.0,0.0,0.0,0.0,0.0])|
+-----+-----------------------------------------+-----------------------------------------+
```

Celem TF-IDF jest znalezienie słów o większym znaczeniu. Algorytm śledzi lokalne znaczenie słowa w dokumencie za pomocą obliczeń TF oraz globalne znaczenie słowa w całym korpusie szkoleniowym za pomocą obliczeń IDF. Na koniec oba obliczenia są mnożone, aby uzyskać ostateczną wagę słowa. Zachęcamy jednak do zapoznania się ze sposobem, w jaki można to zastosować w systemie NLP, ponieważ zachowanie w rankingu TF-IDF może nie dać odpowiednich wyników w twoim przypadku użycia. Możesz zastosować wiele korekt do korpusu, aby uzyskać pożądane zachowanie. Poniżej znajduje się wzór matematyczny dla TF-IDF:

Wzór do obliczania częstotliwości (TF)

$$tf_{t,d} = n_{t,d} / \sum_{i=0}^{i=N} n_{i,d}$$

Gdzie jest termin lub słowo w dokumencie, d, $n_{t,d}$. ,jest liczbą terminów t w dokumencie .

$$d. \sum_{i=0}^{i=N} n_{i,d}$$

jest liczbą wszystkich terminów w dokumencie.

Wzór na obliczenie odwrotnej częstotliwości dokumentów (IDF):

$$idf_t = \log_{10}(N/df_t)$$

Gdzie dft to częstotliwość terminów w dokumencie i całkowita liczba dokumentów w korpusie. Wzór masy TF-IDF to:

$$w_{t,d} = (1 + (1 + tf_{t,d})).idf_t$$

CountVectorizer

CountVectorizer i CountVectorizerModel działa na liczbę słów (tokenów). Wykorzystuje słowa w dokumentach tekstowych do budowy wektorów zawierających liczbę tokenów. Zawiera postanowienia dotyczące używania słownika słów do identyfikowania tokenów, które można traktować jako dane wejściowe do algorytmów. Jeśli słownik nie jest dostępny, CountVectorizer używa własnego estymatora do budowania słownictwa. Na podstawie tego słownictwa generuje CountVectorizerModel, rzadkie reprezentacje dokumentów szkoleniowych. Ten model działa jako dane wejściowe dla algorytmów NLP, takich jak LDA. CountVectorizer zlicza częstotliwości słów dla dokumentu, podczas gdy TF-IDF nadaje nam znaczenie tego słowa w odniesieniu do całego korpusu. CountVectorizer jest jednym z narzędzi służących do konwersji tekstu na wektor, który można

przekazać jako funkcję do modelu uczenia maszynowego. Podobnie jak TF-IDF, model ten wytwarza również rzadkie reprezentacje dokumentów w zakresie słownictwa. Na przykład, jeśli weźmiemy pod uwagę wyodrębnienie funkcji za pomocą ciągu tekstowego metody wyodrębniania Countvectorizera, wówczas wynik wyglądałby mniej więcej tak:

```scala
import org.apache.spark.ml.feature.{CountVectorizer, CountVectorizerModel,Tokenizer}


val exampleData = spark.createDataFrame(Seq(
  (0.0, "extraction of the feature using countvectorizer extraction method")
)).toDF("label", "sentence")

val tokenizer = new Tokenizer().setInputCol("sentence").setOutputCol("words")
val tokensData = tokenizer.transform(exampleData)

val cvModel: CountVectorizerModel = new CountVectorizer()
  .setInputCol("words")
  .setOutputCol("features")
  .setVocabSize(3)
  .setMinDF(1)
  .fit(tokensData)

cvModel.transform(tokensData).select("words","features").show(false)
```

Dane wyjściowe kodu CountVector:

```
+-----------------------------------------------------------------------+-------------------------+
|words                                                                  |features                 |
+-----------------------------------------------------------------------+-------------------------+
|[extraction, of, the, feature, using, countvectorizer, extraction, method]|(3,[0,1,2],[2.0,1.0,1.0])|
+-----------------------------------------------------------------------+-------------------------+
```

Widzimy we wcześniejszym przykładzie, że tablica pierwszych słów to wyodrębnione cechy z dokumentu, podobne do TF-IDF, ale drugą tablicą cech jest macierz [SparseVector], która reprezentuje indeks i częstotliwość słów uporządkowaną od najwyższej do najniższego. Również tutaj 3 jest rozmiar słownictwa, co oznacza, że CountVectorizer wybiera i jest równy odrębnym słowom w dokumencie, który w naszym przypadku wynosi 3. Możesz to dostosować w Spark.

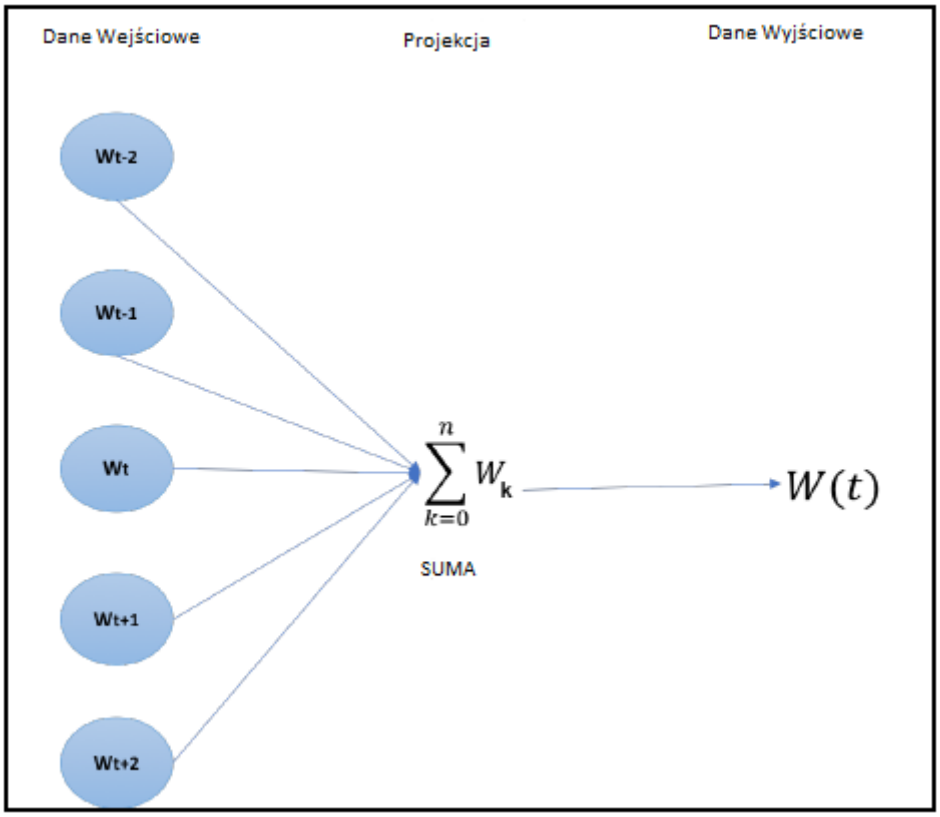
Word2Vec

W typowej funkcji wyodrębniania z tekstu wektory numeryczne są tworzone na podstawie nadanych im unikalnych etykiet. Jednak te unikatowo oznakowane rzadkie wektory nie reprezentują kontekstu, w którym pojawiło się każde słowo. Innymi słowy, nie określa konkretnie lub reprezentują związek danego słowa z innymi słowami w korpusie. Oznacza to, że algorytmów uczenia się bez nadzoru, które uczą się na podstawie przetwarzania danych, nie można znacznie wykorzystać. Algorytmy te nie mogą wykorzystywać relacji ani informacji kontekstowych o tym słowie. Dlatego opracowano nową klasę algorytmów do wyodrębniania funkcji, która zachowuje informacje o kontekście lub relacji między słowami. Ta nowa klasa algorytmów nosi nazwę algorytmów wyodrębniania funkcji osadzania wyrazów. Te klasy algorytmów reprezentują rzadkie wektory w ciągłe modele przestrzeni wektorowej (VSM). W VSM podobne słowa są odwzorowane na pobliskie punkty, dzięki czemu tworzą grupę podobnych słów. Word2Vec to metoda predykcyjna oparta na algorytmach osadzania słów, które mogą być zaimplementowane na dwa sposoby, model ciągłej torby słów (CBOW) i model Skip-Gram.

CBOW

Większość modeli predykcyjnych opiera się na słowach lub kontekstach, które pojawiły się w poprzednich słowach. Na podstawie uczenia się na podstawie wcześniejszych słów przewidują następne słowo. CBOW, w przeciwieństwie do tego, używa N słów przed i po danym słowie, aby przewidzieć wynik.

Wykorzystuje ciągłą reprezentację worka słów, aby przewidzieć wynik. Jednak porządek nie ma tu znaczenia. CBOW przyjmuje kontekst w postaci okna słów i przewiduje słowo. Poniższy rysunek przedstawia działanie CBOW:



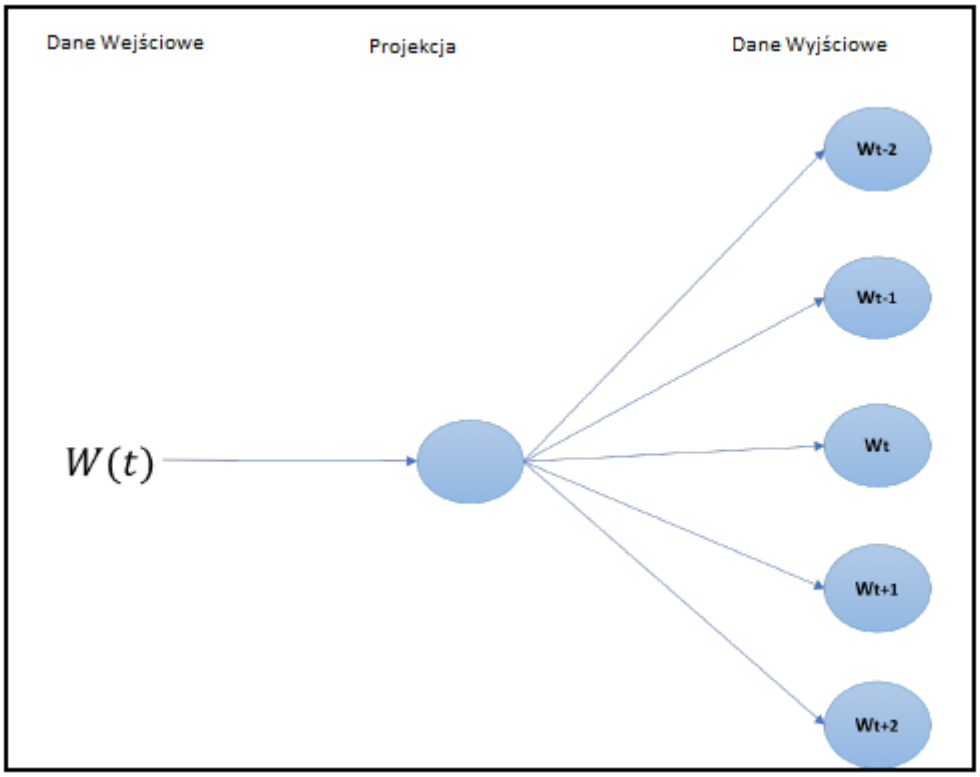
Na podstawie poprzedniego schematu CBOW można sformalizować jako:

$$J_{\varnothing} = \frac{1}{T}\sum_{t=1}^{T} \log_p(W_t|W_{t-n},\ldots,W_{t-1},\ldots W_{t+1},\ldots W_{t+n})$$

Poprzednia formuła opiera się na oknie n słów wokół słowa docelowego. t oznacza krok czasowy. Okno słów obejmuje poprzednie i następne słowa.

Model Skip-Gram

Model Skip-Gram działa przeciwnie do modelu CBOW. Przewiduje kontekst na podstawie bieżącego słowa. Innymi słowy, wykorzystuje świat centralny do przewidywania słów pojawiających się przed i po głównym słowie. Poniższy rysunek przedstawia model Skip-Gram:



Na podstawie poprzedniego schematu Skip-Gram można sformalizować jako:

$$J_{\varnothing} = \frac{1}{T}\sum_{t=1}^{T}\sum_{-n\leq j\leq n}\log_p(W_{t+j}|W_t)$$

Model pominięcia oblicza i podsumowuje logarytmiczne prawdopodobieństwa poprzedniego i następnego, n, słów otaczających słowo docelowe, W_t. Oto kod do obliczenia Word2Vec przy użyciu modelu Skip-Gram w Spark:

```
import org.apache.spark.ml.feature.{Word2Vec,Tokenizer}
import org.apache.spark.ml.linalg.Vector
import org.apache.spark.sql.Row

val exampleData = spark.createDataFrame(Seq(
  (0.8, "extraction of the feature using word2Vec extraction method")
)).toDF("label", "sentence")

val tokenizer = new Tokenizer().setInputCol("sentence").setOutputCol("words")
val tokensData = tokenizer.transform(exampleData)

val word2Vec = new Word2Vec()
  .setInputCol("words")
  .setOutputCol("features")
  .setVectorSize(3)
  .setMinCount(0)
val model = word2Vec.fit(tokensData)
val result = model.transform(tokensData)
result.select("words","features").show(false)
```

Kod wyjściowy Word2Vec Skip-Gram Spark wygląda następująco:

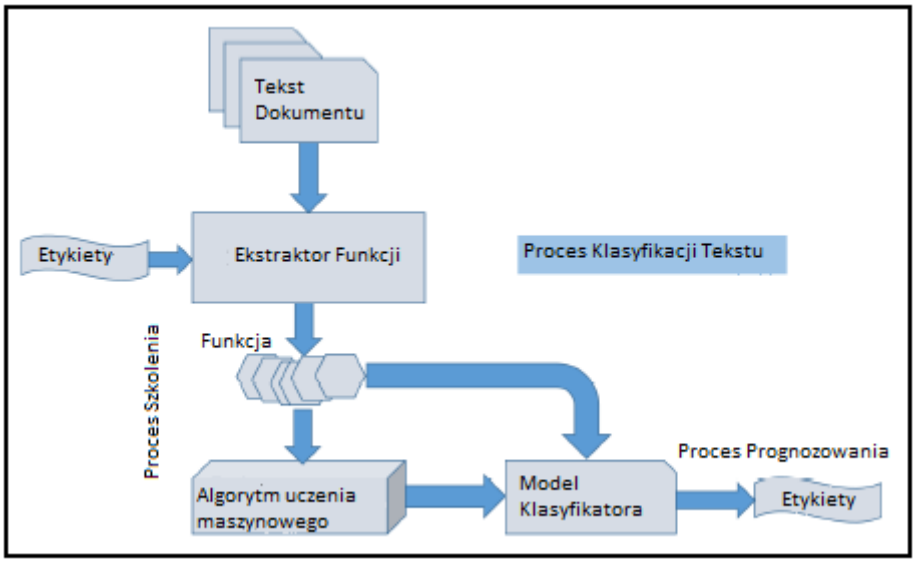
```
+--------------------------------------------------------------+--------------------------------------------------------------------------------------------------------------------------------------------------------+
|words                                                         |features                                                                                                                                                |
+--------------------------------------------------------------+--------------------------------------------------------------------------------------------------------------------------------------------------------+
|[extraction, of, the, feature, using, word2vec, extraction, method]|[-0.036735267378389835,-0.017351628514086734,0.014259896153816953]|
+--------------------------------------------------------------+--------------------------------------------------------------------------------------------------------------------------------------------------------+
```

Stosowanie technik NLP

Zasadniczo w przypadku dowolnej klasy problemów NLP najpierw stosuje się wstępne przetwarzanie tekstu i techniki wyodrębniania funkcji. Po zmniejszeniu hałasu w tekście i możliwości wyodrębnienia funkcji z tekstu, wykonujesz różne algorytmy uczenia maszynowego, aby rozwiązać różne klasy problemów NLP. W tej sekcji zajmiemy się jednym z takich problemów, zwanym klasyfikacją tekstu.

Klasyfikacja tekstowa

Klasyfikacja tekstu jest jednym z bardzo częstych przypadków użycia NLP. Klasyfikacja tekstu może być stosowana w przypadkach użycia, takich jak wykrywanie spamu e-mail, identyfikacja hierarchii produktów detalicznych i analiza sentymentów. Ten proces jest zwykle problemem klasyfikacyjnym, w którym próbujemy zidentyfikować konkretny temat ze źródła w języku naturalnym dużej ilości danych. W obrębie każdej grupy danych możemy omawiać wiele tematów, dlatego ważne jest, aby sklasyfikować artykuł lub informacje tekstowe w logiczne grupy. Pomagają nam w tym techniki klasyfikacji tekstu. Te techniki wymagają dużej mocy obliczeniowej, jeśli ilość danych jest ogromna to zaleca się stosowanie rozproszonych ram obliczeniowych do klasyfikacji tekstu. Na przykład, jeśli chcemy sklasyfikować dokumenty prawne istniejące w repozytorium wiedzy w Internecie, możemy zastosować techniki klasyfikacji tekstu do logicznego rozdzielenia różnych rodzajów dokumentów. Poniższa ilustracja przedstawia typowy proces klasyfikacji tekstu, który odbywa się w dwóch etapach:



Zobaczmy teraz, jak można przeprowadzić klasyfikację tekstu za pomocą Spark. Nasz kod podzielimy na cztery części: wstępne przetwarzanie tekstu, wyodrębnianie funkcji, szkolenie / weryfikacja modelu i przewidywanie. Użyjemy algorytmu Naive Bayesa do szkolenia modeli i prognozowania. Ale zanim zagłębimy się w kod, omówimy, jak działa NB. Podamy również krótki przegląd innego algorytmu, Random Forest, którego można użyć do klasyfikacji tekstu.

Wprowadzenie do algorytmu Naive Bayesa

Klasyfikator Naive Bayes (NB) jest bardzo silnym algorytmem do zadań klasyfikacyjnych. NB jest bardzo dobry w przypadkach, w których do analizy tekstu używamy przetwarzania języka naturalnego. Podobnie jak w przypadku nazwy, Naive oznacza niezależną zależność lub jej brak, a algorytm NB zakłada, że nie ma związku między cechami. Jak sama nazwa wskazuje, działa na podstawie twierdzenia Bayesa. Czym więc jest twierdzenie Bayesa? Twierdzenie Bayesa określa prawdopodobieństwo wystąpienia zdarzenia w przyszłości na podstawie zdarzeń, które już miały miejsce. Ten rodzaj prawdopodobieństwa nazywany jest również prawdopodobieństwem warunkowym. Prawdopodobieństwo to zależy od kontekstu, a kontekst zależy od wiedzy o zdarzeniach, które już miały miejsce. Oto matematyczne wyrażenie twierdzenia Bayesa:

$$P(A/B) = \frac{P(B/A)P(A)}{P(B)}$$

Dla dowolnych dwóch zdarzeń, A i B, twierdzenie Bayesa oblicza P (A | B) (prawdopodobieństwo zdarzenia A występującego po wystąpieniu zdarzenia B) z P (B | A) (prawdopodobieństwo wystąpienia zdarzenia B, biorąc pod uwagę, że zdarzenie A już się wydarzyło). Naive Bayes próbuje klasyfikować punkty danych na klasy. Oblicza prawdopodobieństwo każdego punktu danych należącego do klasy. Następnie porównuje się każde z prawdopodobieństw, aby uzyskać najwyższe prawdopodobieństwo, i określa się drugie największe prawdopodobieństwo. Klasa najwyższego prawdopodobieństwa jest uważana za klasę podstawową, a drugie najwyższe prawdopodobieństwo za klasę wtórną. Jeśli masz wiele klas - na przykład załóżmy, że klasyfikujemy owoce jako jabłko, banan, pomarańcza lub mango, to mamy więcej niż dwie klasy, w których klasyfikujemy owoce - jest to znane jako MultiNomial Naive Bayes, a jeśli miałby tylko dwie klasy - na przykład e-mail jako spam lub nie-spam - byłby to Binomial MultiNomial Naive Bayes. Algorytm NB byłby bardziej przejrzysty w następującym przykładzie:

A pathology lab is performing a test of a disease, D, with two results, Positive or Negative. They guarantee that their test result is 99% accurate: if you have the disease, you will test positive 99% of the time. If you don't have the disease, you will test negative 99% of the time. If 3% of all the people have this disease and test gives the positive result, what is the probability that you have the disease?

Aby rozwiązać powyższy problem, będziemy musieli użyć prawdopodobieństwa warunkowego. Poniższe obliczenia matematyczne pokazują, w jaki sposób matematyczne prawdopodobieństwo warunkowe byłoby stosowane:

Probability of people suffering from Disease D, P(D) = 0.03 = 3%

Probability that test gives "positive" result and patient have the disease, P(Pos | D) = 0.99 =99%

Probability of people not suffering from Disease D, P(~D) = 0.97 = 97%

Probability that test gives "positive" result and patient does have the

disease, P(Pos | ~D) = 0.01 =1%

For calculating the probability that the patient actually have the disease i.e, P(D | Pos) we will use Bayes theorem:

P(D | Pos) = (P(Pos | D) * P(D)) / P(Pos)

We have all the values of numerator but we need to calculate P(Pos):

P(Pos) = P(D, pos) + P(~D, pos)

= P(pos|D)*P(D) + P(pos|~D)*P(~D)

= 0.99 * 0.03 + 0.01 * 0.97

= 0.0297 + 0.0097

= 0.0394

Let's calculate, P(D | Pos) = (P(Pos | D) * P(D)) / P(Pos)

= (0.99 * 0.03) / 0.0394

= 0.753807107

Poprzedni przykład pokazuje, że istnieje około 75% szansy na wystąpienie choroby u pacjenta.

Losowy las

Random Forest to klasa algorytmów zaliczana do kategorii algorytmów nadzorowanego uczenia. Opiera się na lasach drzew, które w niektórych kontekstach przypominają drzewa decyzyjne. Algorytmy losowego lasu mogą być stosowane zarówno w przypadku problemów z klasyfikacją, jak i regresją. Drzewo decyzyjne podaje zestaw reguł używanych w modelach budynków, które można wykonać na podstawie testowego zestawu danych dla prognozy. W drzewach decyzyjnych najpierw obliczamy węzeł główny. Aby obliczyć węzeł główny, używamy zdobywania informacji. Na przykład, jeśli chcesz przewidzieć, czy twój przyjaciel przyjmie ofertę pracy, czy nie. Musisz podać zestaw danych szkoleniowych z ofert, które zaakceptowali, do drzewa decyzyjnego. Na tej podstawie drzewo decyzyjne przedstawi zestaw reguł, których będziesz używać w przewidywaniu. Powiedzmy więc, że regułą może być pensja> 50 tys., Wtedy twój przyjaciel zaakceptuje ofertę. Algorytm drzewa decyzyjnego może się nakładać, ponieważ jest bardzo elastyczny. Aby uniknąć nadmiernego dopasowania tego modelu w drzewie decyzyjnym, możemy wykonać przycinanie. Poniżej znajduje się pseudokod algorytmu Random Forest:

1. Losowo wybierz k funkcji spośród wszystkich m funkcji. Gdzie k << m.

2. Spośród funkcji k obliczyć węzeł d, używając najlepszego punktu podziału.

3. Podziel węzeł na węzły potomne przy użyciu najlepszego podziału.

4. Powtarzaj kroki od 1 do 3, aż zostanie osiągnięta l liczba węzłów.

5. Zbuduj las, powtarzając kroki od 1 do 4 dla liczby n razy, aby utworzyć liczbę n drzew.

Po wytrenowaniu modelu na podstawie poprzednich kroków, w celu przewidywania musimy przejść funkcje testowe przez wszystkie reguły utworzone przez różne drzewa w lesie. Jeśli chcemy zrozumieć na przykładzie, załóżmy, że chcesz kupić telefon komórkowy i postanowiłeś zapytać znajomych, który telefon jest dla Ciebie najlepszy. W takim przypadku Twoi znajomi mogą zadać Ci losowe pytanie dotyczące funkcji, które lubisz, i zasugerować odpowiedni telefon. Tutaj każdy przyjaciel jest drzewem, a dzięki połączeniu wszystkich przyjaciół tworzymy las. Po zebraniu sugestii od znajomych (drzew, pod względem algorytmu Losowy Las), policzysz, który typ telefonu ma najwięcej głosów, i możesz go kupić. Podobnie w Losowym lesie każde drzewo będzie przewidywać inną zmienną docelową, którą zsumujemy w odniesieniu do tego klucza. Kluczem o największej liczbie, przewidywanym na podstawie maksymalnej liczby drzew, jest ostateczna zmienna docelowa.

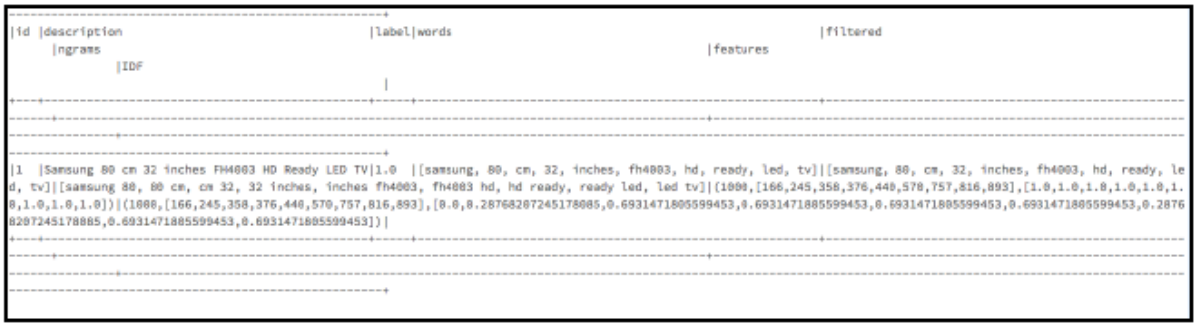
Przykład kodu klasyfikacji tekstu Naive Bayes

Poniższy kod przedstawia sposób przeprowadzania klasyfikacji tekstu za pomocą algorytmu NB:

```
import org.apache.spark.ml.{Pipeline, PipelineModel}
import org.apache.spark.ml.classification.{NaiveBayes, NaiveBayesModel}
import org.apache.spark.ml.feature.{StringIndexer, StopWordsRemover,
HashingTF, Tokenizer, IDF, NGram}
import org.apache.spark.ml.linalg.Vector
import org.apache.spark.sql.Row
//Sample Data
val exampleDF = spark.createDataFrame(Seq(
(1,"Samsung 80 cm 32 inches FH4003 HD Ready LED TV"),
(2,"Polaroid LEDP040A Full HD 99 cm LED TV Black"),
(3,"Samsung UA24K4100ARLXL 59 cm 24 inches HD Ready LED TV Black")
)).toDF("id","description")
exampleDF.show(false)
//Add labels to dataset
val indexer = new StringIndexer()
.setInputCol("description")
.setOutputCol("label")
val tokenizer = new Tokenizer()
.setInputCol("description")
.setOutputCol("words")
val remover = new StopWordsRemover()
.setCaseSensitive(false)
.setInputCol(tokenizer.getOutputCol)
.setOutputCol("filtered")
val bigram = new
NGram().setN(2).setInputCol(remover.getOutputCol).setOutputCol("ngrams")
val hashingTF = new HashingTF()
.setNumFeatures(1000)
.setInputCol(bigram.getOutputCol)
.setOutputCol("features")
```

```
val idf = new IDF().setInputCol(hashingTF.getOutputCol).setOutputCol("IDF")

val nb = new NaiveBayes().setModelType("multinomial")

val pipeline = new

Pipeline().setStages(Array(indexer,tokenizer,remover,bigram,

hashingTF,idf,nb))

val nbmodel = pipeline.fit(exampleDF)

nbmodel.write.overwrite().save("/tmp/spark-logistic-regression-model")

val evaluationDF = spark.createDataFrame(Seq(

(1,"Samsung 80 cm 32 inches FH4003 HD Ready LED TV")

)).toDF("id","description")

val results = nbmodel.transform(evaluationDF)

results.show(false)
```

Poniższy zrzut ekranu przedstawia wyniki:



Wdrażanie analizy sentymentów

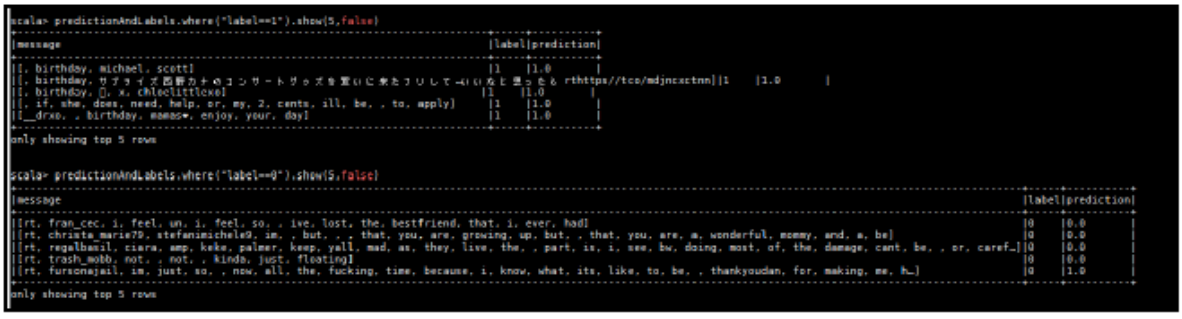
W poniższym fragmencie kodu zaimplementowaliśmy analizę sentymentów opartą na teorii NLP omówionej w tym rozdziale. Wykorzystuje biblioteki SPARK w rekordach Tweetera JSON do trenowania modeli w zakresie rozpoznawania nastrojów, takich jak szczęśliwy lub nieszczęśliwy. Wyszukuje słowa kluczowe typu „szczęśliwy” w wiadomościach na Twitterze, a następnie oznacza je wartością 1, wskazując, że ta wiadomość reprezentuje szczęśliwy nastrój. Inne wiadomości są oznaczone wartością 0, która reprezentuje niezadowolenie. Wreszcie algorytm TF-IDF jest stosowany do modeli pociągów:

```
import org.apache.spark.sql.functions._

import org.apache.spark.ml.classification.LogisticRegression

import org.apache.spark.ml.Pipeline

import org.apache.spark.ml.classification.MultilayerPerceptronClassifier

import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator

import scala.util.{Success, Try}
```

```
import sqlContext.implicits._
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
var tweetDF = sqlContext.read.json("hdfs:///tmp/sa/*")
tweetDF.show()
var messages = tweetDF.select("msg")
println("Total messages: " + messages.count())
var happyMessages =
messages.filter(messages("msg").contains("happy")).withColumn("label",lit("
1"))
val countHappy = happyMessages.count()
println("Number of happy messages: " + countHappy)
var unhappyMessages = messages.filter(messages("msg").contains("
sad")).withColumn("label",lit("0"))
val countUnhappy = unhappyMessages.count()
println("Unhappy Messages: " + countUnhappy)
var allTweets = happyMessages.unionAll(unhappyMessages)
val messagesRDD = allTweets.rdd
val goodBadRecords = messagesRDD.map(
row =>{
val msg = row(0).toString.toLowerCase()
var isHappy:Int = 0
if(msg.contains(" sad")){
isHappy = 0
}else if(msg.contains("happy")){
isHappy = 1
}
var msgSanitized = msg.replaceAll("happy", "")
msgSanitized = msgSanitized.replaceAll("sad","")
//Return a tuple
(isHappy, msgSanitized.split(" ").toSeq)
}
```

```
)
val tweets = spark.createDataFrame(goodBadRecords).toDF("label","message")
// Split the data into training and validation sets (30% held out for
validation testing)
val splits = tweets.randomSplit(Array(0.7, 0.3))
val (trainingData, validationData) = (splits(0), splits(1))
val tokenizer = new
RegexTokenizer().setGaps(false).setPattern("\\p{L}+").setInputCol("msg").se
tOutputCol("words")
val hashingTF = new
HashingTF().setNumFeatures(1000).setInputCol("message").setOutputCol("featu
res")
val idf = new IDF().setInputCol(hashingTF.getOutputCol).setOutputCol("IDF")
val layers = Array[Int](1000, 5, 4, 3)
val trainer = new MultilayerPerceptronClassifier().setLayers(layers)
val pipeline = new Pipeline().setStages(Array(hashingTF,idf,trainer))
val model = pipeline.fit(trainingData)
val result = model.transform(validationData)
val predictionAndLabels = result.select("message","label","prediction")
predictionAndLabels.where("label==0").show(5,false)
predictionAndLabels.where("label==1").show(5,false)
```

Dane wyjściowe są następujące:



Poprzednie wdrożenie jest bardzo podstawową formą analizy sentymentalnej opartej na NLP i powinno być postrzegane jako prosty przykład zrozumienia analizy sentymentalnej. W tym przykładzie można zastosować bardziej zaawansowane techniki, aby uczynić go bardziej dostosowanym do aplikacji klasy korporacyjnej.

Często Zadawane Pytania

P: Jakie są najczęstsze przypadki przetwarzania języka naturalnego?

O: Przetwarzanie języka naturalnego jest gałęzią algorytmów uczenia maszynowego, które przetwarzają dane tekstowe w celu uzyskania istotnych informacji. Kilka typowych przypadków użycia NLP to odpowiadanie na pytania zadawane przez użytkownika, analiza sentymentalna, tłumaczenie języka na język obcy, wyszukiwarki i klasyfikacje dokumentów. Kluczową kwestią do zrozumienia tutaj jest to, że jeśli chcesz przeprowadzić analizę / uczenie maszynowe na danych reprezentowanych przez format tekst / zdania / słowo, NLP jest właściwą drogą.

P: W jaki sposób ekstrakcja funkcji ma znaczenie dla NLP?

O: Algorytmy uczenia maszynowego działają na formach matematycznych. Wszelkie inne formularze, takie jak Tekst, należy przekształcić w formularze matematyczne, aby zastosować algorytmy uczenia maszynowego. Wyodrębnianie obiektów to przekształcanie formularzy, takich jak teksty / obrazy, w funkcje numeryczne, takie jak Wektory. Te cechy liczbowe działają jako dane wejściowe do algorytmów uczenia maszynowego. Techniki takie jak TF-IDF i Word2Vec są używane do konwersji tekstu na funkcje numeryczne. W skrócie, ekstrakcja funkcji jest obowiązkowym krokiem do wykonania NLP dla danych tekstowych.

Podsumowanie

Dokonaliśmy przeglądu jednej z najważniejszych technik ewolucji inteligentnych maszyn, aby zrozumieć i zinterpretować ludzki język w jego naturalnej formie. Omówiliśmy niektóre ogólne koncepcje w NLP przykładowym kodem i przykładami. Konieczne jest, aby technika NLP i nasze rozumienie tekstu były coraz lepsze wraz z coraz większą liczbą zasobów danych wykorzystywanych do szkolenia. Łącząc NLP z ontologicznym światopoglądem, inteligentne maszyny mogą czerpać znaczenie z zasobów tekstowych w skali internetowej i ewoluować do systemu know-all, który może uzupełnić ludzką zdolność do zrozumienia ogromnej ilości wiedzy i wykorzystać ją we właściwym czasie z najlepsze możliwe działania w oparciu o kontekst.