Uniwersytet Warmińsko-Mazurski

Analiza danych z użyciem Apache Spark

Spark MLlib

Paweł Procaj

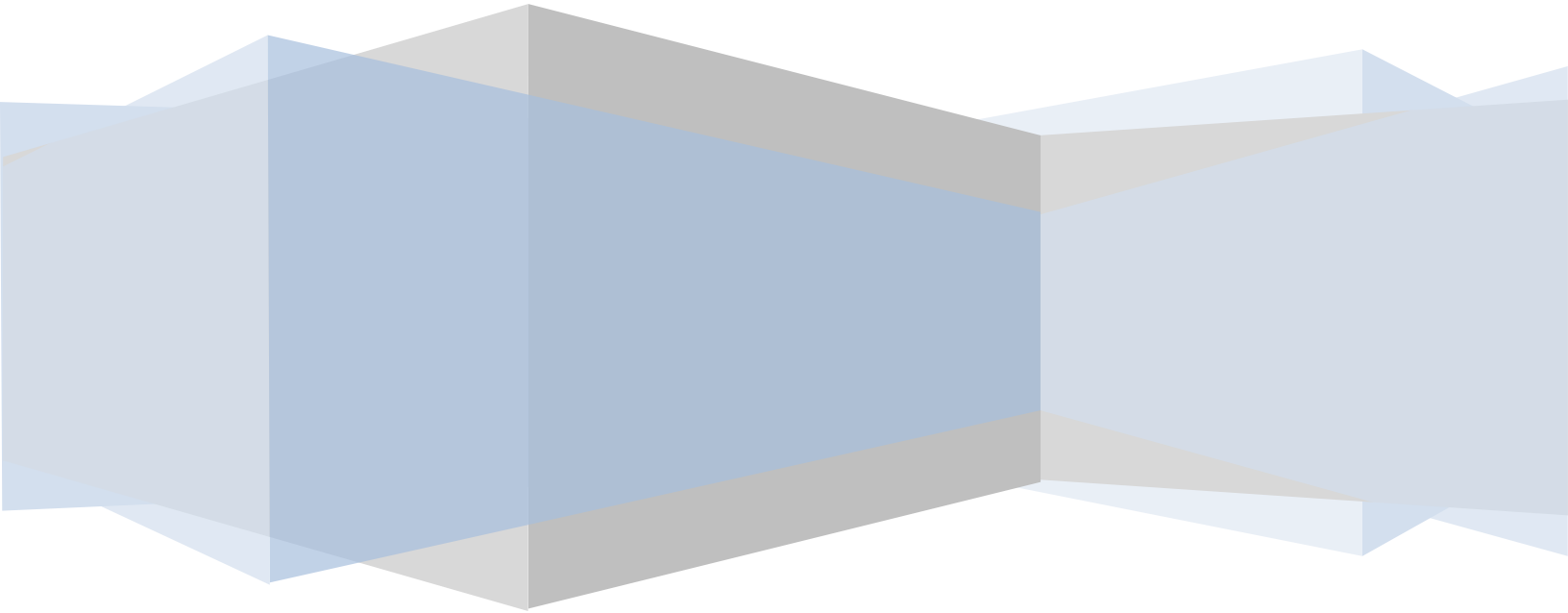


Table of Contents

Spark Machine Learning Library (MLlib)

Spark MLlib lub inaczej Spark ML to biblioteka do zastosowań z dziedziny uczenia maszynowego. Dostarcza następujących narzędzi:

- Algorytmy uczenia maszynowego: popularne algorytmu uczenia takie jak klasyfikacja, regresja, klastrowanie oraz filtrowanie kolaboratywne.
- Określanie cech: wyodrębnianie cech, transformacje, redukcja wymiarów oraz selekcja i filtrowanie.
- Potoki: narzędzia do tworzenia, wykonywania oraz tuningowania potoków uczenia maszynowego.
- Trwałość (persystencja): zapisywanie i ładowanie algorytmów, modeli oraz potoków.
- Różne nadzędzia z dziedziny algebry liniowej, statystyk, obróbki danych.

Struktury danych

Podział typów danych w bibilotece MLlib przedstawia poniższy rysunek:

Ramka danych / DataFrame

Podstawową strukturą danych w interfejsie programistycznym biblioteki MLlib jest ramka danych, czyli DataFrame dostępna w module Spark SQL. Na jej bazie tworzone są inne typy danych, wykorzystywane w bibliotece MLlib.

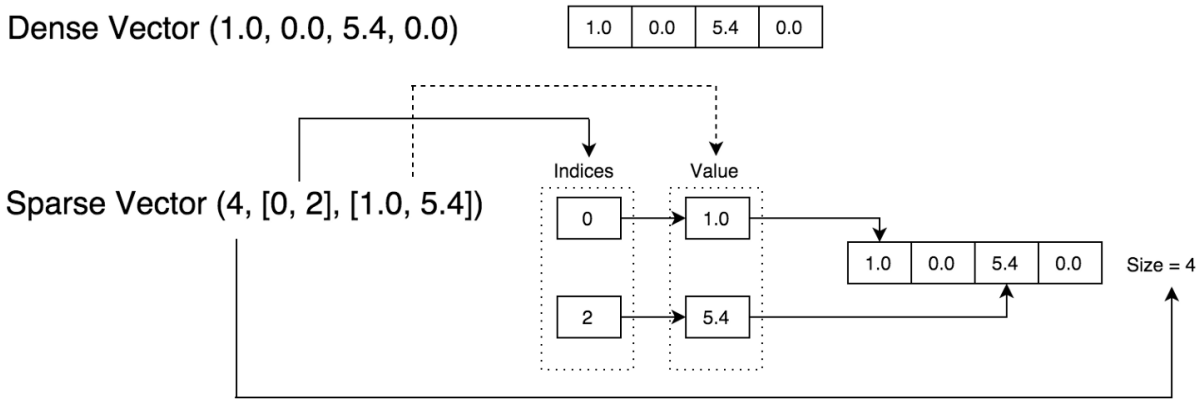
Typy danych dostępne w pakiecie pyspark.ml

PySpark MLlib zawiera wiele własnych struktur danych, umieszczonych w pakiecie pyspark.ml:

- **Wektory lokalne** – Wektor lokalny ma indeksy całkowite zaczynające się od 0 oraz wartości typu double, przechowywane na pojedynczej maszynie.
    - **DenseVector** - wektory gęste, przechowujące każdą wartość cechy.
    - **SparseVector** - wektory rzadkie, przechowujące tylko wartości niezerowe, by oszczędzić miejsce. Wektory można tworzyć za pomocą klasy pyspark.ml.linalg.Vectors.
- **LabeledPoint** – oznaczony punkt danych dla nadzorowanych algorytmów uczenia takich jak klasyfikacja i regresja. Składa się na niego etykieta (wartość zmiennoprzecinkowa) oraz wektor cech. Umieszczony w pakiecie ml.regression.
- **Rating** – ocena produktu przez użytkownika, stosowana w pakiecie ml.recommendation do rekomendacji produktu.
- **Macierze lokalne** – przechowywane na pojedynczej maszynie, z indeksami całkowitymi oraz wartościami typu double, podobnie jak wektory występują macierze gęste i rzadkie.
- **Macierze rozproszone** – dostępne w pakiecie spark.mllib.

Wektory

Przykład wektora gęstego oraz rzadkiego przedstawia poniższy rysunek:



Macierze lokalne

Rodzaje macierzy lokalnych:

- **macierze gęste** - wartości komórek są przechowywane w pojedynczej tablicy typu double w porządku kolumnowym (najpierw kolumny)
- **macierze rzadkie** – ich wartości niezerowe są przechowywane w formaciie CSC (skompresowana rzadka kolumna) w porządku kolumnowym (najpierw kolumny)

Przykładowo macierz gęsta:

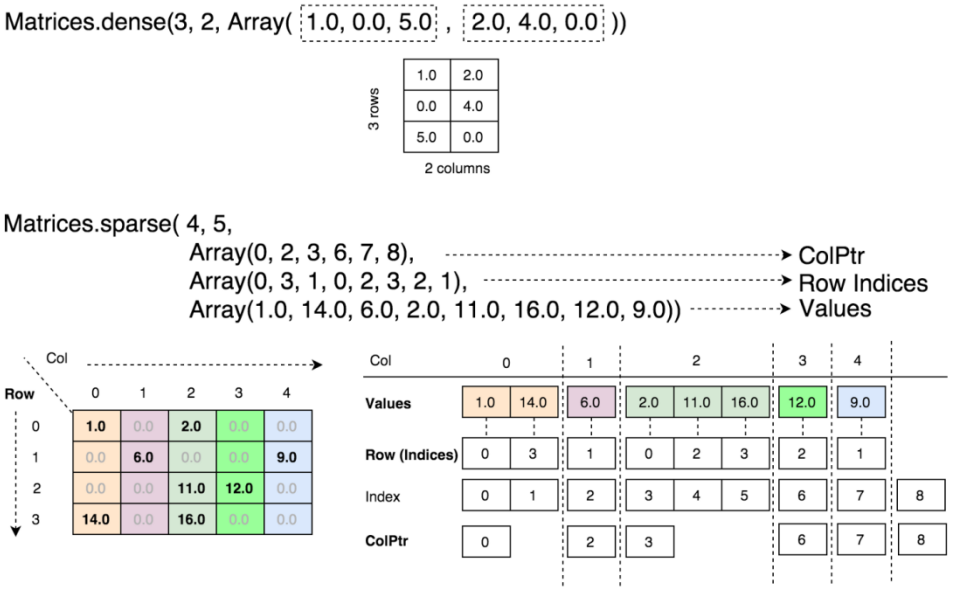
| 1.0 | 2.0 |

| 3.0 | 4.0 |

| 5.0 | 6.0 |

jest trzymana w 1-wymiarowej tablicy [1.0, 3.0, 5.0, 2.0, 4.0, 6.0] jako macierz rozmiaru (3, 2).

Przykład macierzy:



Czytanie danych w formacie LibSVM (Library for Support Vector Machines)

W praktyce często używa się danych treningowych zapisanych w formacie rzadkim (sparse). MLlib wspiera czytanie przykładów treningowych przechowywanych w formacie LIBSVM, który jest domyślnym formatem używnym przez biblioteki LIBSVM oraz LIBLINEAR. Jest to format tekstowy, w którym każda linia reprezentuje wektor cech posiadających etykietę.

Format ma postać:

label index1:value1 index2:value2 ...

gdzie indeksy zaczynają się od 1 (1-based) i są w porządku rosnącym.

Po załadowaniu indeksy cech są konwertowane na zaczynające się od 0 (0-based).

W PySparku do czytania przykładów treningowych w formacie LIBSVM używana jest funkcja MLUtils.loadLibSVMFile.

Przykład użycia.

```
from pyspark.mllib.util import MLUtils

examples = MLUtils.loadLibSVMFile(sc, "data/mllib/sample_libsvm_data.txt")
```

Potoki

Potoki są składową interfejsu API Sparka i służą do łączenia wielu algorytmów w jeden potok czy też przepływ pracy.

Na całość modułu potoków składają się następujące części:

- **DataFrame**: Jako kolekcję danych moduł uczenia maszynowego Sparka używa struktury danych DataFrame, pochodzącej z modułu Spark SQL, która może zawierać wartości o różnych typach danych. Przykładowo obiekt DataFrame mogłby zawierać różne kolumny trzymające tekst, wektory cech, etykiety (LabeledPoint) czy predykcje.
- **Transformer**: Transformer jest algorytmem, który potrafi przekształcić jedną ramkę danych DataFrame w inną. Przykładowo model ML jest typu Transformer, który przekształca DataFrame z cechami na DataFrame z predykcjami.
- **Estimator**: Estymator jest algorytmem, który może być wytrenowany z użyciem danych z DataFrame, by docelowo przekształcić się w Transformer. Przykładowo algorytm uczący się jest obiektem typu Estimator, który trenuje się na danych z DataFrame i produkuje model wyjściowy.r
- **Pipeline**: Pipeline (potok) łączy razem w łańcuch wiele obiektów typu Transformer oraz Estimator definiując w ten sposób przepływ pracy.
- **Parameter**: Wszystkie obiekty typu Transformer oraz Estimator współdzielą interfejs API do ustawiania, zapisywania i czytania parametrów.

Komponenty potoku

Obiekty Transformers

Transformer jest abstrakcją, która obejmuje dwa typy:

- transformer cech,
- modele uczące się.

Technicznie Transformer implementuje metodę transform(), która konwertuje jedną ramkę DataFrame na inną, w ogólności dołączając jedną lub więcej kolumn. Na przykład:

- Transformer cech bierze obiekt ramkę DataFrame, czyta kolumnę (np. tekst), mapuje ją na nową kolumnę (wektor cech) i na wyjściu produkuje obiekt DataFrame z dołączoną zmapowaną kolumną.
- Model uczący się bierze obiekt ramkę DataFrame, czyta kolumnę zawierającą wektory cech, dokonuje predykcji wartości etykiety dla każdego wektora cech i na wyjściu produkuje nową ramkę DataFrame z dołączoną kolumną z predykcjami etyiket.

Estymatory

Estimator to pewna abstrakcja algorytmu uczącego się lub dowolnego algorytmu, który trenuje się na danych. Technicznie Estimator implementuje metodę fit(), która akceptuje obiekt DataFrame i produkuje obiekt Model, który jest równocześnie typu Transformer. Przykładowo algorytm uczący się

taki jak LogisticRegression jest typu Estimator, a wywołanie jego metody fit() trenuje model LogisticRegressionModel, który jest obiektem typu Model i stąd również typu Transformer.

Właściwości komponentów potoku

- Metody Transformer.transform() oraz Estimator.fit() są bezstanowe.
- Każda instancja obiektu typu Transformer lub Estimator ma unikalny identyfikator, który jest użyteczny przy określaniu parametrów.

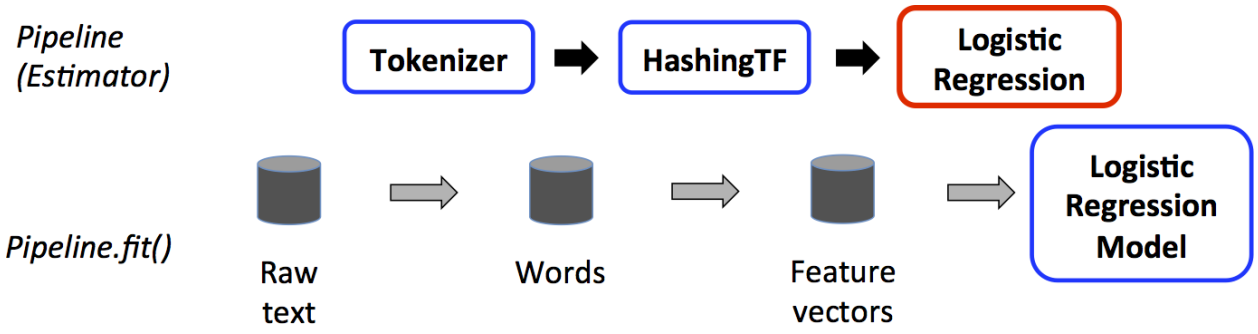
Potok / Pipeline

W uczeniu maszynowym powszechne jest uruchamianie sekwencji algorytmów przetwarzających dane i uczących się na ich podstawie. Przykładowo przepływ pracy związany z przetwarzaniem prostego dokumentu tekstowego może obejmować następujące etapy:

- Podziel każdy z dokumentów / tekstów na słowa.
- Skonwertuj każdy słowo dokumentu na liczbowy wektor cech.
- Naucz model predykcyjny korzystając z wektorów cech i etykiet.

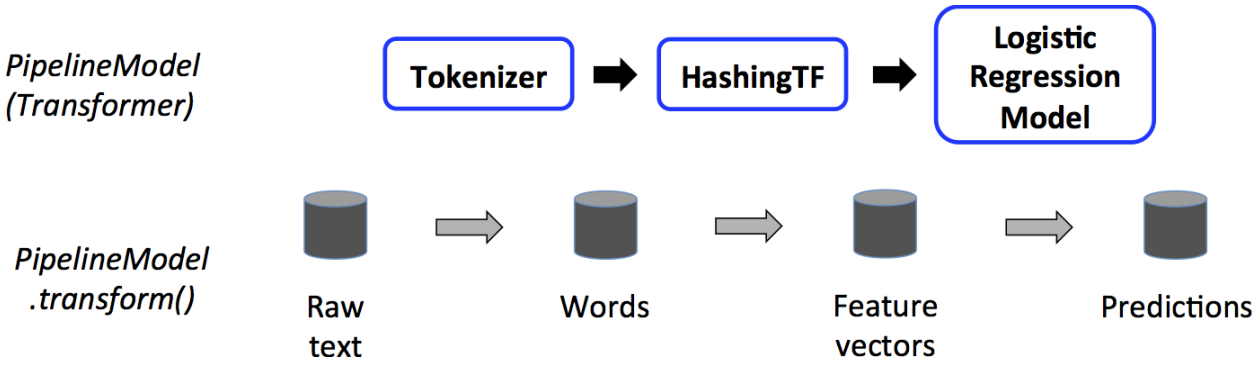
Pipeline składa się z obiektów PipelineStage będących typu Transformer bądź Estimator, które muszą być uruchomione w ustalonym porządku

Przykładowa ilustracja obrazująca obróbkę tekstu z użyciem algorytmu regresji logistycznej na etapie trenowania modelu.



Na rysunku mamy Pipeline z trzema etapami. Pierwsze dwa (Tokenizer oraz HashingTF) są typu Transformer (niebieski), a trzeci (LogisticRegression) jest typu Estimator (czerwony). Dolny wiersz reprezentuje dane przepływające przez potok, gdzie cylindry to ramki DataFrame. Pipeline.fit() jest wołana na oryginalnej, pierwszej ramce DataFrame, posiadającej surowy dokumenty tekstowe i etykiety. Metoda Tokenizer.transform() dzieli surowe dokumenty na słowa, dodając nową kolumnę ze słowami do ramki DataFrame. Metoda HashingTF.transform() konwertuje słowa na wektory cech, dodając je jako nową kolumnę w DataFrame. Ponieważ LogisticRegression jest typu Estimator to Pipeline najpierw woła LogisticRegression.fit() by utworzyć model LogisticRegressionModel. Jeśli Pipeline miałby więcej estymatorów Estimator, to mógłby zawołać metodę LogisticRegressionModel.transform() na ramce DataFrame przed przekazaniem jej DataFrame do następnego etapu.

Utworzony model PipelineModel jest wykorzystywany na etapie testowania / sprawdzania, co przedstawia rysunek poniżej.



Parametery

Obiekty klas Estimator oraz Transformer mają identyczne API do określania parametrów.

Parametr to pojedynczy nazwany parameter plus jego dokumentacja, natomiast mapa parametrów to zbiór par (parametr, wartość).

Dwa sposoby przekazywania parametrów do algorytmów:

1. Parametry ustawiane na instancji, np. jeśli lr to instancja klasy LogisticRegression można wywołać wtedy: lr.setMaxIter(10) lr.fit() by zmienić liczbę iteracji.
2. Parametry przekazywane jako słownik do metody fit() lub transform(). Parametry tego typu nadpiszą parametry wcześniej ustawione na instancji.

Zapisywanie i ładowanie potoków

Mechanizmy persystencji są dostępne również dla obiektów modułu ML. Można zatem:

- Zapisać do pliku i ponownie wczytać poszczególne konfiguracje algorytmów (metody save i load na obiektach).
- Zapisać i załadować potoki.
- Zapisane modele mogą być wczytywane w innym języku (oprócz R).

Wyodrębnianie, przekształcanie oraz wybieranie cech

Przetwarzając dane z użyciem Spark ML napotkamy na następujące etapy:

1. Ekstrakcja / wyodrębnianie: wyodrębnianie cech z surowych danych.
2. Transformacja / przekształcanie: skalowanie, konwertowanie lub modyfikacja cech.

3. Selekcja / wybieranie: wybieranie podzbioru z większego zbioru cech.
4. Locality Sensitive Hashing (LSH): Klasa algorytmów łącząca transformację cech z innymi algorytmami.

Ekstraktory cech

TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF (częstość terminu– odwrotna częstość w dokumentach) to metoda cechowania szeroko stosowana w przetwarzaniu tekstu. Ma ona za zadanie określić ważność danego słowa w zbiorze dokumentów.

Dwa ważne oznaczenia:

- TF(t,d) = ile razy dane słowo (termin) t występuje w dokumencie d
- DF(t,D) = w ilu dokumentach zbioru D występuje dane słowo (termin) t

Idea jest prosta:

gdybyśmy wyliczając ważność słowa brali tylko pod uwagę jego częstość występowania, to moglibyśmy ją przeszacować, gdyż mogłyby to być słowa często spotykane, ale niosące mało informacji o dokumencie, np. „a", „the", „of". Jeśli słowo występuje w zbiorze bardzo często może to oznaczać, że nie niesie za wiele informacji o danym dokumencie.

Odwrotna częstość w dokumntach jest liczbową miarą tego jak wiele informacji dostarcza dane słowo:

$$\mathrm{IDF(t, D)} = ln \frac{|D| + 1}{DF(t, D) + 1}$$

Ostateczna wartość TF-IDF wynosi:

TFIDF(t, d, D) = TF(t, d) * IDF(t, D)

W bibliotece Spark ML występują dwa ekstraktory wyliczające wartość wektora TF-IDF:

1. HashingTF
2. CountVectorizer.

HashingTF

HashingTF to Transformer, który bierze zestaw słów i konwertuje go na wektor cech o stałej długości.

Działanie:

- W przetwarzaniu tekstu zbiór słów / terminów bywa strukturą będącą workiem słów, czyli tzw. hash-mapą, którą właśnie wykorzystuje transformer HashingTF.

- Surowa cecha / słowo jest mapowana na indeks w tzw. tablicy hashującej nakładając na nią funkcję hashującą (tu MarmurHash3).
- Następnie obliczane są częstości słowa na podstawie zmapowanych indeksów.
- Ryzyko kolizji w funkcji hashującej można zmniejszyć zwiększając liczbę kubełków tablicy hashującej.
- Wymiar docelowej cechy domyślnie ma wartość 218 = 262144 kubełki.
- Opcjonalnie można zastosować przełącznik binarny, który dla nie-zerowych wystąpień słowa ustawi wartość 1, nie zwiększając licznika.

Algorytm mechanizmu hashowania:

```
function hashing_vectorizer(features : array of string, N : integer):
    x := new vector[N]
    for f in features:
        h := hash(f)
        x[h mod N] += 1
    return x
```

CountVectorizer

CountVectorizer oraz CountVectorizerModel mają za zadanie skonwertować kolekcję dokumentów tekstowych na wektory liczników słów / tokenów w nich zawartych.

Działanie:

- Jeśli nie istnieje słownik wstępny to CountVectorizer może być użyty jako Estimator do ekstrakcji słownika i generacji modelu CountVectorizerModel.
- Model ten produkuje rzadkie reprezentacje dokumentów używając zbudowanego słownika.
- Może być dalej przekazany do algorytmów takich jak LDA (Latent Dirichlet Allocation).
- Podczas procesu trenowania CountVectorizer będzie wybierał najczęstsze słowa (w ilości vocabSize) uporządkowane wg częstości słowa w zbiorze dokumentów.
- Opcjonalny parameter minDF może wpływać na proces trenowania przez określanie minimalnej liczby (lub frakcji jeśli < 1.0) dokumentów, w których dany termin musi się pojawić, by być włączonym do słownika.
- Inna opcja to przełącznik binarny kontrolujący wektor wyjściowy.
- Ustawiony na true sprawia, że wszystkie niezerowe liczniki będą ustawione na 1.
- Użyteczny w dyskretnych modelach probabilistycznych opartych bardziej o liczniki binarne niż całkowite.

FeatureHasher

Ekstraktor FeatureHasher rzutuje zbiór cech kategorialnych bądź liczbowych na wektor cech o określonym wymiarze, zwykle znacząco mniejszym niż przestrzeń oryginalnej cechy). Jest to zasługa mapowania z użyciem tablicy hashującej, by określić indeksy cech wektora.

Działanie:

- FeatureHasher operuje na wielu kolumnach jednocześnie.
- Każda kolumna może zawieraćdane kategorialne bądź numeryczne.
- Kolumny liczbowe: użyta jest wartość hash nazwy kolumny, by zmapować indeks. Domyślnie cechy numeryczne nie są traktowane jako kategorialne, więc aby to zmienić należy ustawić parametr categoricalCols na tablicę kolumn liczbowych.
- Kolumny String:dla cech kategorialnych użyty jest hash wartości łańcucha “column_name=value” jako indeks wartości w wektorze,ze wskaźnikiem1.0. W ten sposób są one kodowane podobnie jak robi to OneHotEncoder z parametremdropLast=false).
- Kolumny Boolean: Wartości Boolean (true / false) traktowane są tak samo jak reprezentacje hash “column_name=true” or “column_name=false”, ze wartością wskaźnika 1.0.
- Wartości brakujące (Null) są ignorowane (wartość 0.0 w wektorze cech).
- Jako funkcja hashująca użyta jest również MarmumrHash3, podobnie jak w HashingTF.

Feature Transformers

Tokenizer

Tokenizacja to podział tekstu (zdania) na pojedyncze tokeny (terminy, słowa).

Dwie klasy tokenizerów:

1. Podstawowa implementacja (klasa pyspark.ml.feature.Tokenizer) dzieli zdania na słowa (lowercased) a jako separatora używa pojedynczych białych znaków.
2. Bardziej złożony RegexTokenizer pozwala użyć wyrażenia regularnego jako separatora tokenów (domyślnie jest to ‘\\s+’). Ma również drugi tryb, w którym można określić, co dokładnie jest tokenem, ustawiając parametr gaps = false.

StopWordsRemover

Słowa “Stop words” to takie, które powinny być wyłączane z wejścia, ponieważ przykładowo występują często i nie mają za dużo znaczenia.

StopWordsRemover filtruje wejściową cechy (zdania), usuwając z nich wszystkie słowa typu “stop words” (parameter stopWords).

Dla kilku języków Spark posiada standardową listę takich słów. Można się do niej odwołać ten sposób:

```
StopWordsRemover.loadDefaultStopWords(language)
```

gdzie language to jedna z wartości: “danish”, “dutch”, “english”, “finnish”, “french”, “german”, “hungarian”, “italian”, “norwegian”, “portuguese”, “russian”, “spanish”, “swedish” and “turkish”.

N-gram

Transformer konwertujący wejściową tablicę obiektów typu string (np. wyjście Tokenizera) na tablicę n-gramów, gdzie n to liczba słów w każdym n-gramie. N-gram jest zatem sekwencją n-tokenów (zwykle słów) dla pewnej liczby n.

Cechy:

- Wartości nullowe są ignorowane.
- Każdy wyjściowy n-gram to kolejne n słów rozdzielonych spacją.
- Gdy n liczba elementów tablicy wejściowej to mniej niż n wtedy zwracana jest pusta tablica.

Binarizer

Binaryzacja to process progowania cech liczbowych do cech binarnych (0/1).

Cechy transformera Binarizer:

- Standardowe parametery inputCol oraz outputCol, a także threshold jako wartość progu binaryzacji.
- Wartości większe niż próg stają się binarną wartością 1.0.
- Wartości równe lub mniejsze niż próg to wartość 0.0
- Jako inputCol wspierane są zarówno wektory gęste jak i rzadkie.

StringIndexer

StringIndexer koduje kolumnę etykiet (typ string) na kolumnę indeksów etykiet.

Cechy:

- Numeracja indeksów: [0, numLabels)
- Porządek wg częstości wystąpień etykiety.
- Najczęściej występujące etykiety mają indeks 0.
- Niewidoczne etykiety będą miały wartość numLabels, jeśli wymagją zachowania.
- Jeżeli wejściowa kolumna jest numeryczna, rzutowana jest na string i indeksowana jako typ string.

IndexToString

Symetryczna do StringIndexer jest IndexToString , która mapuje kolumnę indeksów etykiet na kolumnę zawierającą oryginalne etykiety w postaci stringów.

Powszechnym użyciem jest następujący schemat:

Stwórz indeksy z etykiet z użyciem StringIndexer.

Wytrenuj model z użyciem indeksów etykiet.

Używając IndexToString wyciągnij indeksy i oryginalne etykiety z kolumny z przewidywanymi indeksami.

OneHotEncoder

Kodowanie typu one-hot mapuje kolumnę z indeksami etykiet na kolumnę z binarnymi wektorami, z co najwyżej jedną wartością.

Cechy:

- Ten rodzaj kodowania pozwala używać cech kategorialnych algorytmom oczekującym wartości liczbowych ciągłych (np. w regresji logistycznej).
- Na wyjściu jest wektor rzadki.

VectorIndexer

VectorIndexer pomaga indeksować cechy kategorialne na indeksy kategorii i potrafi konwertować je automatycznie.

Schemat działania:

- Weź kolumnę wejściową typu Vector oraz parametr maxCategories.
- Zdecyduj, które cechy powinny być kategorialne bazując na liczbie ich zbioru różnych wartości, gdzie cechy z maksymalnie maxCategories są uważane za kategorialne.
- Oblicz indeksy kategorii (0-based) dla każdej cechy kategorialnej.
- Zindeksuj cechy kategorialne oraz przekształć oryginalną cechę na indeksy.

VetorIndexer pozwala algorytmom takim jak Drzewa Decyzyjne oraz Tree Ensembles traktować cechy kategorialne jako liczbowe, zwiększając wydajność.

VectorAssembler

VectorAssembler jest transformerem, który łączy daną listę kolumn w pojedynczy wektor kolumn.

Cechy:

- Użyteczny przy łączeniu cech surowych i cech wygenerowanych przez inne transformer w pojedynczy wektor cech, w celu wytrenowania modelów ML takich jak regresja logistyczna i drzewa decyzyjne.
- VectorAssembler akceptuje wszystkie typy liczbowe, boolean oraz Vector.
- W każdym wierszu wartości kolumn wejściowych są łączone w wektor wyjściowy w określonym porządku.

Feature Selectors

VectorSlicer

VectorSlicer bierze wektor cech i wyciąga z niego podzbiór cech dla wybranych indeksów.

Wektor wyjściowy zawiera kolumny w porządku podanych indeksów.