Metody i narzędzia analizy dużych zbiorów danych

Projekt 2

Inteligentny system analizy danych
oparty na platformie Apache Spark

Wykonali:
Paweł Suchanicz
Anna Zybek

Apache Spark stał się jednym z najczęściej używanych i obsługiwanych narzędzi open-source'owych do uczenia maszynowego i analizy danych.

Tematem projektu było zbudowanie przynajmniej dwóch modeli z wybranego przez siebie problemu zagadnień uczenia maszynowego. Projekt miał być oparty na platformie Apache Spark wykorzystujący strukturę Spark DataFrame.

W naszym projekcie zajęliśmy się Regresją.

1. Opis problemu regresji

Analiza regresji jest narzędziem pozwalającym badać zależność między zmiennymi. W klasycznej postaci wymaga się, aby zmienne objaśniane (zależne) były ilościowe. Zmienne, które służą do obliczenia wartości innej zmiennej są nazywane zmiennymi objaśniającymi (niezależnymi).

Jednym z modeli, którym posłużyliśmy się w tym projekcie jest model regresji liniowej. Ten model zakłada, że pomiędzy zmienną zależną, a zmiennymi niezależnymi istnieje jakaś zależność liniowa. Jeśli istnieje taka zależność mamy wówczas model regresji liniowej. W naszym modelu wykorzystaliśmy więcej niż jedną zmienną objaśniającą. Taki model jest nazywany modelem regresji wielorakiej.

Kolejnym modelem regresji był model drzew decyzyjnych. Drzewa decyzyjne są popularną rodziną metod klasyfikacji i regresji Przy ich pomocy w oparciu o pomiary zmiennych objaśniających dany obiekt jest przypisywany do klasy zmiennej zależnej. Drzewa decyzyjny mogą być wykorzystywane zarówno w klasyfikacji jak i regresji, te wykorzystywane w regresji nazywamy drzewami regresyjnymi. Służą one do przewidywania wartości zmiennej docelowej, charakteryzuje je zmienna docelowa typu ciągłego.

2. Opis danych

Wybrane przez nas dane opisują informacje statystyczne z aktów urodzenia w USA. Statystyki urodzeń w USA są zbierane przez National Center for Health Statistics.
Dane zawarte są w pliku CSV. Plik waży 508 MB. Wszystkie dane dotyczą urodzeń z roku 2018.
Większość zmiennych charakteryzującą się wartością tekstową została zakodowana numerycznie.

Struktura pliku CSV (wypisana większość cech)

Kolumna	Opis	Przykład
attend	Pracownik odbierający poród [id]	1
bfacil	Miejsce urodzenia	1
bmi	Wskaźnik masy ciała (BMI)	33.3
cig_0	Palenie papierosów przed ciążą	0
dbwt	Waga dziecka (zaraz po urodznieu) [g]	3785
dlmp_mm	Ostatni miesiąc miesiączki	3
dlmp_yy	Ostatni rok miesiączki	2017
dmar	Stan cywilny	1
dob_mm	Miesiąc urodzenia	1
dob_tt	Godzina urodzenia	1958
dob_wk	Dzień tygodnia	3
dob_yy	Rok urodzenia	2018
fagecomb	Wiek ojca	27
feduc	Edukacja ojca	7
fhispx	Latynoskie pochodzenie ojca	0
ill_br	Czas od ostatniego porodu	24
ilop_r	Odstęp czasu od ostatniej innej ciąży	30
imp_sex	In vitro	N
ip_gon	Rzeżączka	N
mager	Wiek matki	26
meduc	Edukacja matki	6
mhispx	Latynoskie pochodzenie matki	0
mm_aicu	Intensywna terapia	N

no_infec	Brak infekcji	1
m_ht_ln	Wysokość matki [cal]	63
m_mmorb	Brak chorób u matki	1
no_risks	Brak zgłoszonych czynników ryzyka	1
precare	Miesiąc rozpoczęcia opieki prenatalnej	2
previs	Liczba wizyt prenatalnych	15
priordead	Liczba poronień	0
priorlive	Liczba urodzonych dzieci	2
pwgt_r	Waga kobiety przed ciążą	126
restatus	Status pobytu	1
rf_cesar	Cesarskie cięcie	N
sex	Płeć dziecka	M

Link do pobrania danych: https://www.kaggle.com/des137/us-births-2018
Rozmiar danych: 508 MB (3801534 rekordów).
Zbiór powstał na podstawie surowych danych dostępnych na rządowej stronie iinternetowej Centers for Disease Control and Prevencion z USA :
https://www.cdc.gov/nchs/data_access/vitalstatsonline.htm#Tools

W kolejnych punktach przedstawione zostaną implementacje poszczególnych metod oraz ich wynik

3. Środowisko

Cały projekt – zbudowane modele zostały zaprojektowane na platformie DataBricks. Platforma ta automatycznie tworzy kontekst SparkContext (dostępny pod zmienną ***sc)***.

Utworzony został klaster dla wersji Databricks *6.2 ML (includes Apache Spark 2.4.4, Scala 2.11)*

Po przygotowaniu klastra, na którym później pracowaliśmy, należało załadować dane, poprzez załadowanie pliku *csv* na platformę Databricks. Została utworzona struktura Spark DataFrame, przypominająca SQL-ową tabelę. Ustawione zostały odpowiednie typy danych i nazwy kolumn.

4. Analiza i preprocessing danych

```
births = spark.table("births_2018")
```

Rys. 1 Wczytanie struktury DataFrame do zmiennej

```
# wybranie kolumn gdzie bmi > 0
data = (
  births
    .where(births['bmi'] > 0)
)
print("Liczba wierszy: %d" % (data.count()))

```

Rys. 2 Wybranie kolumn o wartościach numerycznych

Przed zbudowanie modeli regresji, potrzebne było odpowiednie przygotowanie danych. Ważnym usunięcie wierszy zawierających puste wartości ze zbioru danych.

```
# usunięcie kolumn z pustymi wartościami
births_data = data.dropna(how='any')
print("Liczba usuniętych pusty wierszy: ", data.count() - births_data.count())
print("Liczba wierszy:", births_data.count())
```

Rys. 3 Usunięcie wartości pustych

Liczba usuniętych pusty wierszy: 666965

Liczba pozostałych wierszy: 3134569

Następnie należało zamienić dane tekstowe na liczbowe oraz za pomocą obiektu VectorAssembler wybrać kolumny, które będą odgrywać rolę zmiennych objaśniających w modelu i kolumnę, która będzie badana. Do zbudowania modeli wykorzystane zostały wszystkie zmienne. Zmienną objaśnianą była zmienna **bmi**, a pozostałe zmienne to zmienne objaśniające.

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import VectorAssembler

string_columns = ['ip_gon', 'mm_aicu', 'sex', 'ld_indl', 'mtran', 'rf_cesar', 'rf_cesarn']
indexers = [StringIndexer(inputCol=column, outputCol=column+"_index").fit(births_data) for column in string_columns]

vector_assembler = VectorAssembler(inputCols = list(set(births_data.columns) - set(string_columns) - set(['bmi'])),
outputCol = 'features')\


pipeline = Pipeline(stages=indexers+[vector_assembler])
births_data = pipeline.fit(births_data).transform(births_data).select(['features', 'bmi'])
```

Rys. 4 Przygotowanie danych do modelu

Następnie dane podzieliliśmy na dane treningowe i dane testowe.

```
splits = births_data.randomSplit([0.7, 0.3])
trainingData = splits[0]
testData = splits[1]
```

Rys. 4 Podzia ł na zbiór treningowy i testowy

5. Regresja Liniowa

Pierwszą metodą, od której zaczęliśmy naszą analizę danych jest regresja liniowa – wieloraka, bo do zbudowania modelu użyliśmy więcej niż jednej zmiennej niezależnej.

Nasz model regresji wielorakiej został zbudowany za pomocą funkcji *LinearRegression* z biblioteki *pyspark.ml.regression.* Do modelu wykorzystano 70% danych - danych treningowych.

```
#budowa modelu regresji liniowej
from pyspark.ml.regression import LinearRegression
linear = LinearRegression(featuresCol = "features", labelCol='bmi', maxIter=10)
linear_model = linear.fit(trainingData)
print("Wspolczynniki: " + str(linear_model.coefficients))
print("Wyraz wolny: " + str(linear_model.intercept))

trainingSummary = linear_model.summary
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
print("r2: %f" % trainingSummary.r2)
```

Aby sprawdzić jak nasz model został dopasowany do danych, sprawdziliśmy wartość błędu średniokwadratowego i R^2.

```
Wspolczynniki: [0.0093847871398208,-0.01430771709471023,-0.0009532922237878123,-0.22934903951870014,-0.128025376184871,-0.
00921749226829123,0.005888035964223794,0.11585333547658187,0.05638830642295204,-0.23280808289244032,-0.16321758787732216,-
0.31659080104939213,0.17024854524213012,0.37083902779858313,-0.007183944758154973,0.13661624521367152,0.04448283489184088
5,0.022627088030289504,-0.00010705933538690982,0.03324672254141502,8.498102210952242e-05,0.0025975917772709032,0.000315109
7767791913,-0.08454232825667088,0.0001944288175778484,0.00760008724617661,-0.040806924952264775,0.07864000907697899,0.0114
34450339933377,0.02941455743872528,0.0022342786049987426,0.09423245392305972,-0.6619725133999105,0.020766704096745192,-0.0
02863392638263541,-0.024535241067764366,-9.319654119060187e-06,-1.1974998856899183e-05,-0.004925764002119331,0.05947268370
3591246,-2.4301634349925568e-05,0.013942191769621049,-0.004008268761379692]
Wyraz wolny: -34.302840409225134
RMSE: 4.437239
r2: 0.873265
```

RMSE mierzy różnice między wartościami przewidywanymi przez model a wartościami rzeczywistymi. Dla naszego modelu błąd ten wynosi około. 4.62. Wynik ten nie jest zbyt duży, możemy sądzić, że nasz model jest dobrze dopasowany, zwłaszcza gdy zestawimy ten błąd na tle wartości rzeczywistych bmi takich jak średnia, wartość minimalna i maksymalna.

```
1 trainingData.describe().show()
```

▸ (1) Spark Jobs

```
+-------+-----------------+
|summary|              bmi|
+-------+-----------------+
|  count|          2194303|
|   mean|28.75045834779453|
| stddev|12.46422306631803|
|    min|             13.0|
|    max|             99.9|
+-------+-----------------+
```

Przetestowaliśmy nasz model na danych testowych (30% wszystkich danych). Poniżej zamieszczamy wyniki predykcji dla 5 obserwacji i otrzymany błąd średniokwadratowy modelu dla danych testowych:

```
1 from pyspark.ml.evaluation import RegressionEvaluator
2 #Testowanie na danych testowych
3 linear_predictions = linear_model.transform(testData)
4 linear_predictions.select("prediction","bmi","features").show(5)
5 linear_evaluator = RegressionEvaluator(predictionCol="prediction", labelCol="bmi",metricName="r2")
6 print("R Squared (R2) dla danych testowych = %g" % linear_evaluator.evaluate(linear_predictions))
7 test_result = linear_model.evaluate(testData)
8 print("Root Mean Squared Error (RMSE) dla danych testowych = %g" % test_result.rootMeanSquaredError)
```

```
+------------------+----+--------------------+
|        prediction| bmi|            features|
+------------------+----+--------------------+
|22.235696985624273|18.1|[1.0,0.0,1.0,1.0,...|
|30.170772877937317|29.3|[1.0,0.0,1.0,1.0,...|
|22.606531320999544|21.5|[1.0,0.0,1.0,1.0,...|
|  36.0535495082813|39.5|[1.0,0.0,1.0,1.0,...|
| 18.18375866539978|17.0|[1.0,0.0,1.0,1.0,...|
+------------------+----+--------------------+
only showing top 5 rows

R Squared (R2) dla danych testowych = 0.875647
Root Mean Squared Error (RMSE) dla danych testowych = 4.41832
```

6. Regresja za pomocą drzew decyzyjnych

Kolejną metodą, była regresja z wykorzystaniem drzew decyzyjnych. Do zbudowania tego modelu regresji wykorzystana została klasa DecisionTreeRegressor z pakietu *pyspark.ml.regression.* Skorzystano z wcześniej już przygotowanych i podzielonych na zbiory treningowe i testowe danych - tych samych co przypadku regresji liniowej.

```
# Drzewa decyzyjne
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorIndexer

# Wykrycie zmiennych kategorycznyc, jezeli jest 5 lub mniej różnych wartości dla kolumny to traktujemy
# ją jako zmienną kategoryczną
featureIndexer = VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=5)
decisionTreeReg = DecisionTreeRegressor(featuresCol ='indexedFeatures', labelCol = 'bmi')
pipeline = Pipeline(stages=[featureIndexer, decisionTreeReg])

decisionTreeReg_model = pipeline.fit(trainingData)
```

Przetestowaliśmy nasz model na danych testowych (30% wszystkich danych). Poniżej zamieszczamy wyniki predykcji dla 5 obserwacji i otrzymany błąd średniokwadratowy:

```
#Testowanie na danych testowych
dt_predictions = decisionTreeReg_model.transform(testData)
print(dt_predictions)
rmse = RegressionEvaluator(labelCol="bmi", predictionCol="prediction", metricName="rmse").evaluate(dt_predictions)
r2 = RegressionEvaluator(predictionCol="prediction", labelCol="bmi", metricName="r2").evaluate(dt_predictions)
dt_predictions.select("prediction", "bmi", "features").show(5)
print("Root Mean Squared Error (RMSE) na danych testowych = %g" % rmse)
print("R Squared (R2) dla danych testowych = %g" % r2)
# summary only
print(decisionTreeReg_model)
```

```
+------------------+----+-------------------+
|        prediction| bmi|           features|
+------------------+----+-------------------+
|19.750216456721397|18.1|[1.0,0.0,1.0,1.0,...|
|27.337666835720857|29.3|[1.0,0.0,1.0,1.0,...|
|19.750216456721397|21.5|[1.0,0.0,1.0,1.0,...|
| 35.58505428725543|39.5|[1.0,0.0,1.0,1.0,...|
|19.750216456721397|17.0|[1.0,0.0,1.0,1.0,...|
+------------------+----+-------------------+
only showing top 5 rows

Root Mean Squared Error (RMSE) na danych testowych = 2.76482
R Squared (R2) dla danych testowych = 0.951306
```

Jak widać powyżej współczynnik R squared jest większy niż przy regresji liniowej, RMSE jest mniejsze, więc drzewa decyzyjne były lepszym wyborem dla naszych danych niż regresja liniowa.