BIG DATA

CURS 12

Apache Spark MLlib

Extragerea, transformarea și selectarea caracteristicilor

Extragerea, transformarea și selectarea caracteristicilor

- Determinarea (construirea) caracteristicilor este un pas important pentru modelarea datelor
  - De acest pas depinde succesul sau eșecul modelului
- Spark conține algoritmi pentru lucrul cu caracteristici
- Algoritmii pentru lucrul cu caracteristici se clasifică în:
  - Algoritmi de extragere a caracteristicilor din datele brute
  - Algoritmi de transformare: scalare, conversie, modificare a caracteristicilor
  - Algoritmi de selecție a unei submulțimi dintr-o mulțime mai mare de caracteristici
  - Algoritmi LSH (*Locality Sensitive Hashing*) ce combină aspecte ale transformărilor de caracteristici cu alți algoritmi

Algoritmi de extragere a caracteristicilor

TF-IDF (*Term Frequency – Inverse Document Frequency*) – metodă de vectorizare a caracteristicilor utilizată în Text mining

- determină importanța unui termen pentru un document dintr-o colecție.
- t = termen, d = document, D = colecția (corpusul)
- $TF(t, d)$ – frecvența termenului (de câte ori apare termenul t în documentul d)
- $DF(t, D)$ – frecvența la nivel de document (în câte documente apare termenul t)
- $IDF(t, D)$ – măsură numerică a relevanței unui termen = $\log((|D|+1)/(DF(t,D) + 1))$
- $TFIDF(t, d, D) = DF(t, D) \cdot IDF(t, D)$

- Există diferite variante ale definițiilor pentru frecvența unui termen și frecvența unui document

Algoritmi de extragere a caracteristicilor

- În MLlib avem metode separate pentru TF şi IDF
  - TF: *HashingTF*, *CountVectorizer*
  - IDF: *IDF*
- *HashingTF* şi *CountVectorizer* – algoritmi cunoscuţi ce pot fi folosiţi pentru a genera vectorii de frecvenţă a termenilor
  - Convertesc documentele în reprezentări numerice
  - Rezultatul poate fi folosit ca intrare a altor algoritmi (de exemplu, *Cosine Distance* – pentru sistemele de recomandări)
- *HashingTF* vs. *CountVectorizer*
  - *Hashing TF* – ireversibil / *CountVectorizer* – parţial reversibil
  - *HashingTF* necesită o singură parcurgere a datelor şi nu are nevoie de memorie suplimentară pe lângă cea necesară *input*-ului original şi vectorului / CountVectorizer necesită mai mule parcurgeri şi mai multă memorie

Algoritmi de extragere a caracteristicilor

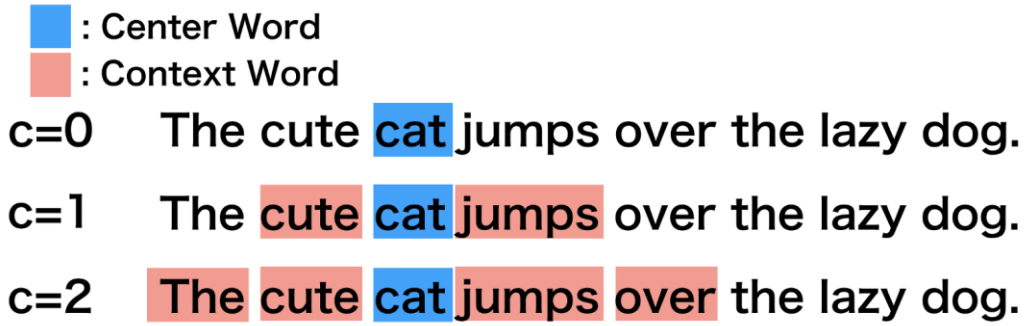
- Exemplu:

```
tokenizer = Tokenizer(inputCol="sentence", outputCol="words")
vectorizer  = CountVectorizer(inputCol="words", outputCol="rawFeatures")

idf = IDF(inputCol="rawFeatures", outputCol="features")

pipeline = Pipeline(stages=[tokenizer, vectorizer, idf])

model = pipeline.fit(sentenceData)
```

Algoritmi de extragere a caracteristicilor

Word2Vec

- Metodă cunoscută pentru implementarea încapsulărilor de cuvinte (*word embeddings*) – tehnică ce permite identificarea similarităților de cuvinte în cadrul unui corpus
- Se creează prin identificarea cuvintelor ce apar într-o fereastră context
- Se pot alcătui ferestre context de diferite lungimi pentru o propoziție:



[https://cbail.github.io/textasdata/word2vec/rmarkdown/word2vec.html]

Algoritmi de extragere a caracteristicilor

- Există 2 tipuri de modele de încapsulare:
  - CBOW *(Continuous Bag Of Words*) – citește cuvintele din fereastra context și realizează o predicție asupra cuvântului central
    - Util în aplicații precum căutarea web predictivă
  - *Skip-Gram Model* – realizează o predicție asupra cuvintelor din context, dar fiind dat cuvântul central
    - Exemplul din figura anterioară corespunde acestui model
    - Util pentru identificarea *pattern*-urilor în cadrul textelor
- Exemplu: demo

Algoritmi de transformare a caracteristicilor

Tokenizer

- Clasa *Tokenizer* oferă funcționalitatea de bază a diviziunii unui text în termeni individuali (cuvinte)
  - Exemplificată în *pipeline*-urile anterioare
- *RegexTokenizer* – permite o separare mai avansată a unui text în termeni individuali, pe baza expresiilor regulate
  - Parametrul *pattern* este utilizat pentru identificarea delimitatorilor pentru divizarea textului, dacă parametrul gap are valoarea *True*
  - Parametrul gap având valoarea *False* indică faptul că *pattern*-ul reprezintă *token*-i (nu separatori)

Algoritmi de transformare a caracteristicilor

- Exemplu (demo)

```
from pyspark.ml.feature import Tokenizer, RegexTokenizer
from pyspark.sql.functions import col, udf
from pyspark.sql.types import IntegerType

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('Features').getOrCreate()

sentenceDataFrame = spark.createDataFrame([
    (0, "Python is a programmimg language used by Apache Spark Python API"),
    (1, "Python and SQL can be used in Spark"),
    (2, "Python,SQL,Java,are,supported,languages")
], ["id", "sentence"])

tokenizer = Tokenizer(inputCol="sentence", outputCol="words")

regexTokenizer = RegexTokenizer(inputCol="sentence", outputCol="words", pattern="\\W")
# sau pattern="\\w+", gaps=False

countTokens = udf(lambda words: len(words), IntegerType())

tokenized = tokenizer.transform(sentenceDataFrame)
tokenized.select("sentence", "words")\
    .withColumn("tokens", countTokens(col("words"))).show(truncate=False)

regexTokenized = regexTokenizer.transform(sentenceDataFrame)
regexTokenized.select("sentence", "words") \
    .withColumn("tokens", countTokens(col("words"))).show(truncate=False)
```

Algoritmi de transformare a caracteristicilor

StopWordsRemover

- *Stop words* – cuvinte ce trebuie, în general, excluse, deoarece apar frecvent și nu contribuie la semnificația textului
- *StopWordRemover* preia, ca intrare, o secvență de șiruri de caractere (care poate fi rezultatul unui *Tokenizer*) și elimină toate cuvintele de tip *stop word*.
  - Lista de *stop words* se specifică în parametrul *stopWords*
  - Există o listă implicită de *stop words* pentru unele limbi (*StopWordsRemover.loadDefaultStopWords*(*lang*))

Algoritmi de transformare a caracteristicilor

Binarizer

- Mapează caracteristici numerice la valorile 0/1, în funcție de un prag (parametrul *threshold*)
  - Valorile <= decât pragul sunt mapate la valoarea 0, iar cele > pragul sunt mapate la valoarea 1
- Exemplu:

```
from pyspark.ml.feature import Binarizer

dataFrame = spark.createDataFrame([
    (0, 0.1),
    (1, 1.5),
    (2, 1.2),
    (3, 2.5)
], ["id", "num_feature"])

binarizer = Binarizer(threshold=1.25, inputCol="num_feature", outputCol="binarized_feature")

binarizedDataFrame = binarizer.transform(dataFrame)

print("Transformarea Binarizer cu pragul = %f" % binarizer.getThreshold())
binarizedDataFrame.show()
```

Algoritmi de transformare a caracteristicilor

Bucketizer

- Exemplu

```
from pyspark.ml.feature import Bucketizer

data = [(0, 18.0), (1, 19.0), (2, 8.0), (3, 5.0), (4, 2.0)]
df = spark.createDataFrame(data, ["id", "age"])
print(df.show())

splits = [-float("inf"),3, 10,float("inf")]
result_bucketizer = Bucketizer(splits=splits, inputCol="age",outputCol="result").transform(df)
result_bucketizer.show()
```

Algoritmi de transformare a caracteristicilor

StringIndexer

- Atribuie categoria 0 valorii celei mai frecvente, 1 următoarei valori în ordinea descrescătoare a frecvenței etc.
- Exemplu:

```
from pyspark.ml.feature import StringIndexer

df = spark.createDataFrame(
    [(0, "a"), (1, "b"), (2, "c"), (3, "a"), (4, "a"), (5, "c")],
    ["id", "category"])

indexer = StringIndexer(inputCol="category", outputCol="categoryIndex")
indexed = indexer.fit(df).transform(df)
indexed.show()
```

```
+---+--------+-------------+
| id|category|categoryIndex|
+---+--------+-------------+
|  0|       a|          0.0|
|  1|       b|          2.0|
|  2|       c|          1.0|
|  3|       a|          0.0|
|  4|       a|          0.0|
|  5|       c|          1.0|
+---+--------+-------------+
```

Algoritmi de transformare a caracteristicilor

OneHotEncoder

- Exemplu:

```
from pyspark.ml.feature import OneHotEncoder, StringIndexer

df = spark.createDataFrame(
    [(0, "a"), (1, "b"), (2, "c"), (3, "a"), (4, "a"), (5, "c")],
    ["id", "category"])

indexer = StringIndexer(inputCol="category", outputCol="categoryIndex")
indexed = indexer.fit(df).transform(df)

encoder = OneHotEncoder(inputCol="categoryIndex", outputCol="categoryVec",dropLast=False) # default setting: dropLast=True
model = encoder.fit(indexed)
encoded = model.transform(indexed)
encoded.show()
```

```
+---+--------+-------------+-------------+
| id|category|categoryIndex|  categoryVec|
+---+--------+-------------+-------------+
|  0|       a|          0.0|(3,[0],[1.0])|
|  1|       b|          2.0|(3,[2],[1.0])|
|  2|       c|          1.0|(3,[1],[1.0])|
|  3|       a|          0.0|(3,[0],[1.0])|
|  4|       a|          0.0|(3,[0],[1.0])|
|  5|       c|          1.0|(3,[1],[1.0])|
+---+--------+-------------+-------------+
```

Algoritmi de transformare a caracteristicilor

Imputer

- Estimator ce completează valorile absente din dataset, folosind în general media sau mediana valorilor coloanelor respective
  - Coloanele asupra cărora se poate aplica sunt coloane numerice.
  - Nu suportă caracteristici categoriale

```
from pyspark.ml.feature import Imputer

df = spark.createDataFrame([
    (1.0, float("nan")),
    (2.0, float("nan")),
    (float("nan"), 3.0),
    (4.0, 4.0),
    (5.0, 5.0)
], ["a", "b"])

#default setting: strategy="mean"
imputer = Imputer(inputCols=["a", "b"], outputCols=["out_a", "out_b"])
model = imputer.fit(df)

model.transform(df).show()
```

```
+---+---+-----+-----+
|  a|  b|out_a|out_b|
+---+---+-----+-----+
|1.0|NaN|  1.0|  4.0|
|2.0|NaN|  2.0|  4.0|
|NaN|3.0|  3.0|  3.0|
|4.0|4.0|  4.0|  4.0|
|5.0|5.0|  5.0|  5.0|
+---+---+-----+-----+
```

Algoritmi de transformare a caracteristicilor

VectorIndexer

- Permite indexarea caracteristicilor categoriale în *dataset*-uri de vectori
- Decide care sunt caracteristicile categoriale şi converteşte valorile iniţiale în indici de categorii
- Indexarea caracteristicilor categoriale pemite algoritmilor bazaţi pe arbori să prelucreze caracteristicile categoriale în mod adecvat, îmbunătăţind performanţa.
- Exemplu (demo):
  - Citirea unui dataset de puncte etichetate, utilizarea lui VectorIndexer pentru a decide ce caracteristici trebuie considerate ca fiind categoriale
  - Se transformă valorile caracteristicilor categoriale în indici
  - Datele astfel transformate pot fi transmise unui algoritm de tip arbore de decizie (care lucrează cu caracteristici categoriale).

Algoritmi de selecție a caracteristicilor

VectorSlicer

- Este un transformator ce preia, ca intrare, un vector de caracteristici și furnizează un nou vector de caracteristici, ce conține o submulțime a caracteristicilor inițiale
  - Util pentru extragerea caracteristicilor dintr-un vector coloană
- Vectorul de intrare este un vector cu indici specificați, fie ca întregi, fie ca nume
- În funcție de problemă, doar anumite coloane pot fi relevante
  - Folosim *VectorSlice* pentru a nu include pozițiile coloanelor irelevante

Algoritmi de selecție a caracteristicilor

- Exemplu:

```
from pyspark.ml.feature import VectorSlicer
from pyspark.ml.linalg import Vectors
from pyspark.sql.types import Row

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('Features_selectors').getOrCreate()


df = spark.createDataFrame([
    Row(userFeatures=Vectors.sparse(3, {0: -2.0, 1: 2.3})),
    Row(userFeatures=Vectors.dense([-2.0, 2.3, 0.0]))])

slicer = VectorSlicer(inputCol="userFeatures", outputCol="features", indices=[1])

output = slicer.transform(df)

output.select("userFeatures", "features").show()
```

```
+--------------------+-------------+
|        userFeatures|     features|
+--------------------+-------------+
|(3,[0,1],[-2.0,2.3])|(1,[0],[2.3])|
|      [-2.0,2.3,0.0]|        [2.3]|
+--------------------+-------------+
```

Algoritmi de selecție a caracteristicilor

ChiSqSelector

- Selecția χ^2 de caracteristici
- Se aplică pe date etichetate cu caracteristici categoriale
- Utilizează testul de independență χ^2 pentru a determina caracteristicile pe care le selectează
- Există 5 metode de selecție:
  - *numTopFeatures* – alege un număr fixat de caracteristici, dintre cele mai relevante conform unui test χ^2 - aceasta este metoda implicită de selecție, cu numărul implicit de caracteristici = 50
  - *percentile* – similar lui *numTopFeatures*, cu deosebirea că este aleasă o fracțiune din toate caracteristicile (nu un număr fixat)
  - *fpr* – alege toate caracteristicile ale căror valori *p* se află sub un anumit prag, controlând astfel rata de selecție *false positive*
  - *fdr* – utilizează procedura Benjamini-Hochberg pentru a alege caracteristicile a căror rată *false discovery* se află sub un anumit prag
  - *fwe* – alege caracteristicile ale căror valori *p* se află sub un prag

Algoritmi de selecție a caracteristicilor

- Exemplu

```
from pyspark.ml.feature import ChiSqSelector
from pyspark.ml.linalg import Vectors

df = spark.createDataFrame([
    (7, Vectors.dense([0.0, 0.0, 18.0, 1.0]), 1.0,),
    (8, Vectors.dense([0.0, 1.0, 12.0, 0.0]), 0.0,),
    (9, Vectors.dense([1.0, 0.0, 15.0, 0.1]), 0.0,)], ["id", "features", "clicked"])

selector = ChiSqSelector(numTopFeatures=1, featuresCol="features",
                         outputCol="selectedFeatures", labelCol="clicked")

result = selector.fit(df).transform(df)

print("ChiSqSelector output with top %d features selected" % selector.getNumTopFeatures())
result.show()
```

```
ChiSqSelector output with top 1 features selected
+---+------------------+-------+----------------+
| id|          features|clicked|selectedFeatures|
+---+------------------+-------+----------------+
|  7|[0.0,0.0,18.0,1.0]|    1.0|          [18.0]|
|  8|[0.0,1.0,12.0,0.0]|    0.0|          [12.0]|
|  9|[1.0,0.0,15.0,0.1]|    0.0|          [15.0]|
+---+------------------+-------+----------------+
```

Algoritmi de selecție a caracteristicilor

UnivariateFeatureSelector

- Operează pe etichete categoriale/continue cu caracteristici categoriale/continue
- Se specifică *featureType* și *labelType*, iar Spark va alege funcția scor pe baza acestor valori

```
featureType |  labelType |score function
------------|------------|--------------
categorical |categorical | chi-squared (chi2)
continuous  |categorical | ANOVATest (f_classif)
continuous  |continuous  | F-value (f_regression)
```

- Există 5 metode de selecție (cele 5 metode disponibile pentru *ChiSqSelector*)

Algoritmi de clasificare și regresie

Algoritmi de clasificare

Regresia logistică

- Metodă de predicție a unui răspuns categorial
- Caz particular de model liniar generalizat, ce permite predicția probabilității valorilor de ieșire
- În Spark ML poate fi aplicată pentru predicția:
  - unui rezultat binar folosind regresia logistică binomială
  - unui rezultat multi-clasă folosind regresia logistică multinomială
- Tipul de algoritm se specifică prin intermediul parametrului *family*
  - Dacă nu se specifică, *Spark* inferează varianta corectă.

Algoritmi de clasificare

Regresia logistică binomială

- Exemplu: antrenarea unui model de regresie logistică binomială și multinomială pentru clasificarea binară cu regularizare de tip elastic net.

```
from pyspark.ml.classification import LogisticRegression

# Incarcare date antrenare
training = spark.read.format("libsvm").load("Data/sample_libsvm_data.txt")

lr = LogisticRegression(=10, regParam=0.3, elasticNetParam=0.8)

# Antrenare model
lrModel = lr.fit(training)

# Afisarea coeficientilor si a interceptiei pentru regresia logistica
print("Coefficients: " + str(lrModel.coefficients))
print("Intercept: " + str(lrModel.intercept))

# Putem folosi algoritmul multinomial pentru clasificarea binara
mlr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8, family="multinomial")

# Antrenare model
mlrModel = mlr.fit(training)

# Putem folosi algoritmul multinomial pentru clasificarea binara cu algoritmul multinomial
print("Multinomial coefficients: " + str(mlrModel.coefficientMatrix))
print("Multinomial intercepts: " + str(mlrModel.interceptVector))
```

Algoritmi de clasificare

- Clasa *LogisticRegressionTrainingSummary* furnizează un sumar pentru un model de regresie logistică (*LogisticRegressionModel*)
- În cazul clasificării binare, sunt disponibile anumite metrici precum ROC

```
from pyspark.ml.classification import LogisticRegression

# Obține sumarul instantei de LogisticRegressionModel antrenate anterior
trainingSummary = lrModel.summary

# Obține funcția obiectiv pentru fiecare iterație
objectiveHistory = trainingSummary.objectiveHistory
print("objectiveHistory:")
for objective in objectiveHistory:
    print(objective)

# Obține valoarea ROC (receiver-operating characteristic) ca dataframe și valoarea areaUnderROC.
trainingSummary.roc.show()
print("areaUnderROC: " + str(trainingSummary.areaUnderROC))

# Setează pragul modelului pentru a maximiza măsura F
fMeasure = trainingSummary.fMeasureByThreshold
maxFMeasure = fMeasure.groupBy().max('F-Measure').select('max(F-Measure)').head()
bestThreshold = fMeasure.where(fMeasure['F-Measure'] == maxFMeasure['max(F-Measure)']) \
    .select('threshold').head()['threshold']
lr.setThreshold(bestThreshold)
```

Algoritmi de clasificare

Regresie logistică multinomială

- Clasificarea multi-clasă este suportată prin intermediul regresiei logistice multinomiale
- Algoritmul produce *K* seturi de coeficienți (*K* fiind numărul de clase), prin urmare o matrice (*coefficientMatrix*)
- Dacă algoritmul are o intercepție, atuci aceasta va fi un vector (*interceptVector*)
- Exemplu: vezi fișier demo

Algoritmi de clasificare

Arbori de decizie

- Arborii de decizie sunt o familie de metode de clasificare și regresie
- Exemplu (vezi demo):
  - încărcarea unui *dataset* în format libsvm, divizarea acestuia în seturi de antrenament și de test, antrenament pe *dataset*-ul corespunzător, evaluare pe *dataset*-ul de test.
  - Exemplul utilizează 2 transformatori de caracteristici pentru pregătirea datelor – aceștia indexează categoriile pentru caracteristicile categoriale și pentru coloana *label*, adăugând metadate în DataFrame, utile algoritmului Decision Tree.

Algoritmi de clasificare

Random forest

- Familie de metode de clasificare și regresie
- Exemplu (vezi demo):
  - încărcarea unui *dataset* în format libsvm, divizarea acestuia în seturi de antrenament și de test, antrenament pe *dataset*-ul corespunzător, evaluare pe *dataset*-ul de test.
  - Exemplul utilizează 2 transformatori de caracteristici pentru pregătirea datelor – aceștia indexează categoriile pentru caracteristicile categoriale și pentru coloana *label*, adăugând metadate în DataFrame, utile algoritmilor bazați pe arbori.

Algoritmi de regresie

Regresie liniară

- Interfața Spark pentru lucrul cu modelele de regresie liniară și cu rezultatele (sumarele) acestora este similar cazului regresiei logistice
- Exemplu: (vezi demo laborator 6)
  - Antrenarea unui model de regresie liniară cu regularizare elastic net și extragerea statisticilor de sumarizare.

Algoritmi de regresie

Regresie liniară generalizată

- În regresia liniară se presupune că rezultatul urmează o distribuție Gaussiană.
- Modelele liniare generalizate (GLM – *Generalized Linear Models*) sunt modele liniare în care variabila rezultat (*Y*) urmează o distribuție din familia distribuțiilor exponențiale
- În Spark, interfața *GeneralizedLinearRegression* permite specificarea flexibilă a modelelor GLM ce pot fi utilizate pentru diferite tipuri de probleme de predicție
  - Acestea includ regresia liniară, regresia Poisson, regresia logistică etc.
  - În Spark este disponibilă o submulțime a familiei de distribuții exponențiale (https://spark.apache.org/docs/latest/ml-classification-regression.html#available-families)

Algoritmi de regresie

- Exemplu (demo)
  - Antrenarea unui GLM cu răspuns Gaussian, cu funcţia identitate şi extragerea statisticilor de sumarizare

Algoritmi de regresie

Arbori de decizie

- Exemplu (vezi demo):
  - încărcarea unui dataset în format libsvm, divizarea acestuia în seturi de antrenament și de test, antrenament pe dataset-ul corespunzător, evaluare pe dataset-ul de test.
  - Exemplul utilizează un transformator de caracteristici pentru indexarea caracteristicilor categoriale, adăugând metadate în DataFrame, utile algoritmului Decision Tree.

Algoritmi de regresie

Random Forests

- Exemplu (vezi demo):
  - încărcarea unui *dataset* în format libsvm, divizarea acestuia în seturi de antrenament și de test, antrenament pe *dataset*-ul corespunzător, evaluare pe *dataset*-ul de test.
  - Exemplul utilizează un transformator de caracteristici pentru indexarea caracteristicilor categoriale, adăugând metadate în DataFrame, utile algoritmilor ce se bazează pe arbori

Bibliografie

- https://spark.apache.org/docs/latest/ml-guide.html
- https://runawayhorse001.github.io/LearningApacheSpark
- https://cbail.github.io/textasdata/word2vec/rmarkdown/word2vec.html
- https://databricks.com/session/building-custom-ml-pipelinestages-for-feature-selection