BIG DATA

CURS 13

Apache Spark MLlib

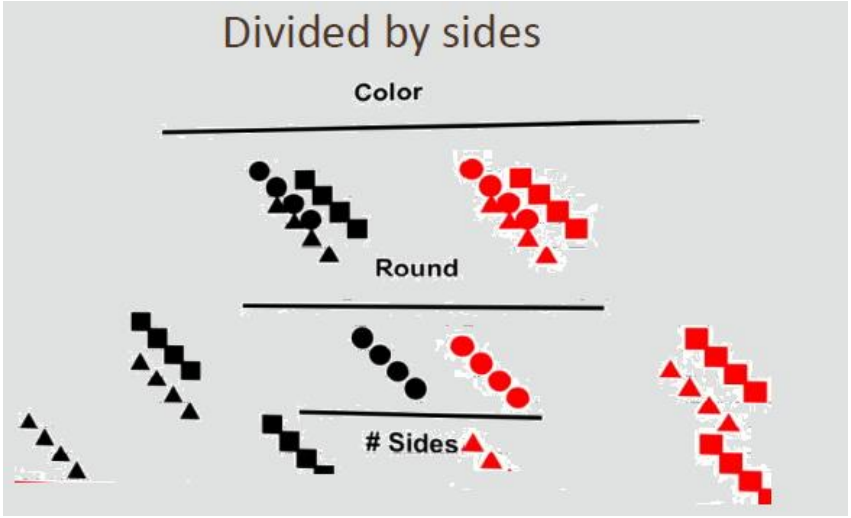
Algoritmi de clasificare și regresie (continuare)

Arbori de decizie și *Random Forests* (*review*)

- 3 metode ML importante bazate pe arbori: arbori de decizie, *random forests*, *boosting*
- Similare, cu multe caracteristici comune
- Un arbore de decizie reprezintă o diagramă simplă pentru luarea deciziilor
- Un *random forest* are un număr mare de arbori ce vor fi combinaţi la final (aplicând valori medii sau reguli ce vizează majoritatea)
- *Gradient boosting* combină, de asemenea, arborii de decizie, însă procesul de combinare are loc de la început, nu la final.

Arbori de decizie și *Random Forests* (*review*)

- **Arborele de decizie** este reprezentat de o serie de pași secvențiali al căror scop este atât să răspundă unei întrebări, cât și să furnizeze probabilități, costuri sau alte valori ce caracterizează luarea unei anumite decizii.
- Sunt foarte intuitivi, oferind o vizualizare clară a ghidării procesului de luare a deciziei.



Arbori de decizie și *Random Forests* (*review*)

- Dezavantaje:
  - *Overfitting* – poate avea mai multe cauze, ce includ zgomotul statistic* și absența unor instanțe reprezentative. Poate apărea în cazul arborilor adânci.
  - Eroarea de deplasare (*bias error*) – are loc atunci când sunt plasate prea multe restricții pe funcțiile obiectiv. Restricționarea rezultatului cu o funcție de restricționare (de exemplu, o funcție liniară) sau printr-un algoritm binar simplu (precum alegerea binară true/false din arborele de decizie) va conduce la deplasări.
  - Eroarea de varianță se referă la cât de mult se va modifica un rezultat în funcție de modificările din setul de antrenament. Arborii de decizie au varianță mare, ceea ce înseamnă că modificări minore pe setul de antrenament pot determina modificări mari în rezultatul final.

*Zgomot statistic – o iregularitate aleatoare ce poate apărea în datele reale, ce este cuantificată prin erori și valori reziduale.

Arbori de decizie și *Random Forests* (*review*)

- ***Random forest***
- Arborii de decizie au problemele menţionate anterior.
- Un arbore generat din 99 de intrări de date poate diferi în mod semnificativ de un arbore generat dintr-o singură intrare diferită.
- Ideea: dacă ar exista un mod de a genera un număr mare de arbori, facând media soluțiilor lor, atunci este probabil ca răspunsul să fie mai aproape de cel real.
- *Random forest* – o colecţie de arbori de decizie ce conduc la un singur rezultat.
- De multe ori, *random forests* conduc la mea mai bună acurateţe.

Arbori de decizie și *Random Forests* (*review*)

- Reducerea varianței observată la arborii de decizie are loc datorită:
  - Utilizării de eșantioane diferite pentru antrenare
  - Specificarea de submulțimi aleatoare de caracteristici
  - Construirea și combinarea de arbori mici de decizie
- Un singur arbore de decizie este un predictor slab, dar se construiește rapid. Mai mulți arbori conduc la generarea unui model mai robust și la prevenirea *overfitting*-ului.
- Cu cât sunt mai mulți arbori, cu atât procesul este mai lent: fiecare arbore din pădure trebuie generat, procesat și analizat.
  - În plus, procesul este mai lung cu cât sunt mai multe caracteristici.
  - Reducerea numărului de caracteristici poate îmbunătăți timpul de procesare.
- Un arbore de decizie este ușor de citit – trebuie doar urmată o cale și găsit rezultatul corespunzător. Un *random forest* este mult mai dificil de interpretat.

Gradient Boosting

- ***Random Forest* vs *Gradient Boosting***
- *Gradient Boosting* costă tot dintr-o mulțime de arbori de decizie.
- Există 2 diferențe majore față de *random forest*:
  1. Modul în care sunt construiți arborii: *random forest* construiește fiecare arbore independent, iar *gradient boosting* construiește arborii pe rând. Acest model aditiv funcționează pe etape, introducând câte un element (predictor) slab pentru a îmbunătăți neajunsurile predictorilor slabi existenți.
  2. Combinarea rezultatelor: *random forest* combină rezultatele la finalul procesului (prin medie sau majoritate), iar *gradient boosting* combină rezultatele la fiecare pas.

Gradient Boosting

- Dacă parametrii sunt configuraţi cu atenţie (*tuning*), *gradient boosting* poate avea o performanţă mai bună decât *random forest*.
- *Gradient boosting* poate să nu fie o opţiune bună atunci când există mult zgomot, fiindcă poate conduce la *overfitting*.
- Configurarea parametrilor în *gradient boosting* este mai dificilă decât în *random forest*.
- Cele 2 clase sunt adecvate în tipuri diferite de probleme
- *Random forest* are performanţe bune în detectarea obiectelor multi-clasă şi în bioinformatică (domenii în care există mult zgomot statistic)
- *Gradient boosting* are performanţe bune atunci când datele sunt neechilibrate, precum în probleme de evaluare a riscului în timp real.

Algoritmi de clasificare

Gradient-boosted trees (G-BT)

- Metodă de clasificare şi regresie ce utilizează mulţimi de arbori de decizie
- Algoritmii GBT antrenează iterativ arborii de decizie cu scopul de a optimiza o funcţie de pierdere.
- Implementarea Spark ML pentru GBT poate fi aplicată pentru clasificarea binară şi pentru regresie, utilizând atât caracteristici continue cât şi caracteristici categoriale.
- Coloanele de intrare corespund parametrilor:
  - *labelCol* – reprezintă denumirea coloanei ce va fi prezisă; valoarea sa implicită (numele implicit al coloanei) este "label", iar tipul său de date este Double
  - *featuresCol* – reprezintă denumirea vectorului de caracteristici; valoarea sa implicită este "features".
- Coloana de ieşire corespunde parametrului *predictionCol* – reprezintă numele coloanei prezise. Valoarea sa implicită este "prediction", iar tipul său de date este Double.
- În versiunea curentă, nu are coloane de ieşire pentru *rawPrediction* şi *probability* (prezente în cazul lui *RandomForest*) şi nu suportă clasificarea multi-clasă (pentru aceasta – *decision trees* şi *random forests*).

Algoritmi de clasificare

- Algoritmul de bază:
  - *Gradient boosting* antrenează iterativ o secvență de arbori de decizie.
  - La fiecare iterație, algoritmul folosește mulțimea curentă pentru a prezice eticheta (*label*-ul) fiecărei instanțe de antrenare, iar apoi compară predicția cu valoarea reală.
  - Dataset-ul este reetichetat astfel încât să pună mai mult accent pe instanțele de antrenare care au predicții slabe.
  - Astfel, în iterația următoare arborele de decizie va corecta greșelile anterioare.
  - Mecanismul specific de reetichetare a instanțelor este definit printr-o funcție de pierdere.
  - La fiecare iterație, arborii *gradient boosted* reduc această funcție de pierdere pe datele de antrenament.

Algoritmi de clasificare

- **Funcții de pierdere**
- Spark MLlib suportă câteva tipuri de funcții de pierdere, ce pot fi aplicate fie în probleme de clasificare, fie în probleme de regresie, dar nu în ambele
- *Log Loss* se aplică în probleme de clasificare
  - Formula: $2\sum_{i=1}^{N}\log(1+\exp(-2y_iF(x_i)))$
- Eroarea pătratică (*Squared error*) se aplică în probleme de regresie. Se mai numește *L2 loss* și este funcția implicită de pierdere pentru problemele de regresie.
  - Formula: $\sum_{i=1}^{N}(y_i-F(x_i))^2$
- Eroarea absolută (Absolute error) se aplică în probleme de regresie.Se mai numește *L1 loss* și este mai potrivită în cazul apariției *outlier*-ilor. Decât eroarea pătratică.
  - Formula: $\sum_{i=1}^{N}|y_i-F(x_i)|$

Algoritmi de clasificare

- Parametri specifici GBT:
  - *loss* – funcția de pierdere
  - *numIterations* – numărul de arbori din mulțime. Fiecare iterație produce un arbore. Un număr mai mare de iterații determină un model mai expresiv, îmbunătățind acuratețea datelor de antrenare. Dacă numărul este totuși prea mare, va avea un impact asupra acurateței de la momentul testării.
  - *learningRate* – se utilizează dacă algoritmul are un comportament instabil; o valoare mai mică poate îmbunătăți stabilitatea.
  - *algo* – tipul de problemă (clasificare sau regresie)

Algoritmi de clasificare

- 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: indexarea categoriilor pentru *label* și caracteristicile categoriale, adăugând metadate în DataFrame, utile algoritmilor ce se bazează pe arbori.

Algoritmi de clasificare

SVM liniar (*Linear Support Vector Machine*)

- Un SVM construiește un hiperplan sau o mulțime de hiperplanuri într-un spațiu multi sau infinit dimensional, ce poate fi utilizat în probleme de clasificare, regresie sau alte task-uri.
- Separarea bună este dată de hiperplanul care are cea mai mare distanță până la cele mai apropiate date din setul de antrenare, aparținând oricărei clase.
  - În general, cu cât marginea este mai mare, cu atât este mai mică eroarea de generalizare a clasificatorului.
  - Clasa LinearSVC din SparkML permite clasificarea binară cu SVM liniar.
  - Intern, optimizează funcția de pierdere Hinge (folosită pentru clasificarea de tip margine maximă).
- Exemplu (vezi demo)

Algoritmi de clasificare

Naive Bayes

- Clasificatorii Naive Bayes constituie o familie de clasificatori probabilişti simpli, multi-clasă, ce se bazează pe teorema lui Bayes cu presupuneri tari (naive) de independenţă între fiecare pereche de caracteristici.
- Naive Bayes pot fi antrenaţi foarte eficient.
  - Printr-o singură trecere prin datele de antrenare, calculează distribuţia de probabilitate condiţională a fiecărei caracteristici, dată fiind fiecare etichetă.
  - Pentru predicţie, aplică teorema lui Bayes pentru calculul distribuţiei de probabilitate condiţionale a fiecărei etichete, dată fiind o observaţie
- Există mai mulţi clasificatori Naive Bayes suportaţi de MLlib: multinomial, complement, Bernoulli şi Gaussian

Algoritmi de clasificare

Naive Bayes

- Modelele multinomial, complement şi Bernoulli sunt folosite de obicei pentru clasificarea documentelor
  - În acest context, fiecare observaţie este un document şi fiecare caracteristică reprezintă un termen.
  - Valoarea unei caracteristici este frecvenţa termenului (în cazul multinomial sau complement) sau valoarea 0 sau 1 indicând dacă termenul a fost găsit sau nu în document (cazul Bernoulli)
  - Valorile caracteristicilor în modelele multinomial şi Bernoulli trebuie să fie nenegative
  - Tipul modelului este selectat cu un parametru opţional ce indică una din cele 4 valori, cu valoarea implicită "multinomial".
  - Pentru clasificarea documentelor, vectorii de caracteristici de intrare trebuie să fie vectori rari
- Exemplu: (vezi demo)

Algoritmi de regresie

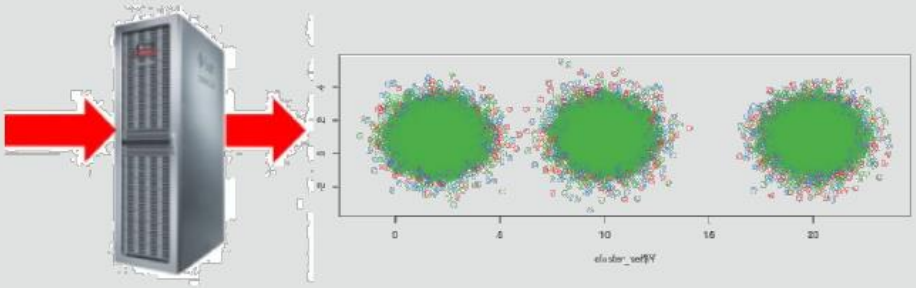
***Gradient-Boosted Tree* (GBT)**

- Exemplu (vezi demo)
  - Pentru dataset-ul exemplu, modelul GBTRegressor necesită o singură iterație, dar acest fapt nu este valabil în general.

Algoritmi de clustering

Algoritmi de *clustering*

1	8.840988	-0.7765538	5.2119346
2	10.693322	1.3662052	5.3242908
3	1.6159301	1.8064995	-0.27138004
4	1.8892474	0.84289026	1.1640315
5	19.634157	1.273606	10.025798
6	12.081877	0.5939875	4.9109597
7	20.291239	1.2454996	10.072135
8	20.333668	-0.6378313	10.290687
9	1.6389576	0.8837965	1.3142129
10	2.2812343	3.1450808	0.23675314
11	-0.69222087	1.1419488	1.1379284
12	9.514933	1.1860658	5.0218225



Algoritmi de *clustering*

K-means

- Unul dintre cei mai folosiți algoritmi de *clustering*
- Grupează datele în cadrul unui număr predefinit de *cluster*-e
- K-means (clasa *KMeans*) este implementat ca estimator și generează un model de bază *KMeansModel*.
- Coloanele de intrare sunt specificate în cadrul parametrului *featuresCol* – de tip Vector, având valoarea implicită "features"
- Coloana de ieșire este specificată în cadrul parametrului *predictionCol* – de tip Int, având valoarea implicită "prediction"
  - Reprezintă centrul *cluster*-ului ce rezultă în urma predicției

Algoritmi de *clustering*

- Evaluare cu scorul (coeficientul) Silhouette – metrică utilizată pentru evaluarea tehnicilor de *clustering*
- Scorul Silhouette ia valori între -1 şi 1, astfel:
  - Scorul 1 reprezintă *cluster*-e bine separate unele de altele, a căror distincţie este clară
  - Scorul 0 reprezintă *cluster*-e între care nu există o distanţă semnificativă
  - Scorul -1 reprezintă *cluster*-e asignate greşit
- Scorul Silhouette = $\frac{b-a}{\max(a,b)}$, unde:
  - a – distanţa medie în interiorul *cluster*-ului (distanţa medie între fiecare 2 puncte din cluster)
  - b – distanţa medie între *cluster*-e (distanţa medie dintre toate *cluster*-ele)
- Pentru calculul distanţei Spark utilizează distanţa pătratică euclidiană
- Exemplu (vezi demo)

Algoritmi de *clustering*

***Gaussian Mixture Model* (GMM)**

- Un model GMM reprezintă o distribuţie compusă în care punctele sunt obţinute dintr-una din cele *k* sub-distribuţii gaussiene, fiecare având propia ei probabilitate
- Implementarea SparkML foloseşte algoritmul expectation-maximization
- Clasa *GaussianMixture* este implementată ca estimator şi generează un model de bază *GaussianMixtureModel*.
- Coloanele de intrare alcătuiesc vectorul de caracteristici, specificat în parametrul *featureCol* (numele implicit fiind "feature")
- Coloanele de ieşire sunt:
  - *predictionCol* – de tip Int, reprezintă centrul *cluster*-ului (rezultat în urma predicţiei); numele implicit este "prediction"
  - *probabilityCol* – un vector ce reprezintă probabilitatea fiecărui *cluster*; numele implicit este "probability"
- Exemplu (vezi demo)

Câteva repere actuale

- https://www.kdnuggets.com/2020/12/kaggle-survey-2020-data-science-machine-learning.html
- https://www.kdnuggets.com/2019/04/top-data-science-machine-learning-methods-2018-2019.html
- https://www.simplilearn.com/10-algorithms-machine-learning-engineers-need-to-know-article
- https://towardsdatascience.com/5-basic-machine-learning-algorithms-you-need-to-know-in-2020-ec4f2825ce06
- **Alte dataset-uri:** https://www.kdnuggets.com/datasets/index.html
- (Exemplu ML Pipeline cu Structured Streaming (de transformat în Python): https://docs.databricks.com/applications/machine-learning/train-model/mllib/index.html#apache-spark-mllib-pipelines-and-structured-streaming-example)

Bibliografie

- https://spark.apache.org/docs/latest/ml-guide.html
- https://runawayhorse001.github.io/LearningApacheSpark
- https://www.datasciencecentral.com/profiles/blogs/decision-tree-vs-random-forest-vs-boosted-trees-explained
- https://docs.databricks.com/applications/machine-learning/train-model/mllib/index.html
- https://towardsdatascience.com/silhouette-coefficient-validating-clustering-techniques-e976bb81d10c