SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

SPARK: SPARK ML
SEMINÁRNA PRÁCA

2022 Kytošová, Stančíková, Števuliaková, Krnáčová

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

SPARK: SPARK ML
SEMINÁRNA PRÁCA

Študijný program:	Aplikovaná informatika
Predmet:	I-ASOS – Architektúra softvérových systémov
Prednášajúci:	RNDr. Igor Kossaczký, CSc.
Cvičiaci:	Ing. Stanislav Marochok

Bratislava 2022 **Kytošová, Stančíková, Števuliaková, Krnáčová**

Obsah

Zoznam obrázkov a tabuliek

Zoznam skratiek

ETL	Extract, transform, load
ML	Machine Learning
MLlib	Machine Learning Library
RDD	Resilient Distributed Dataset

Úvod

Dnes žijeme v dobe, kedy sú informačné systémy neodlúčiteľnou súčasťou nášho života. Tieto informačné systémy zachytávajú obrovské množstvo dát, ktoré musíme vo väčšine prípadov ďalej manuálne spracovávať sami. Bez informácií, ako takých, nevieme vykonávať ďalšiu prácu/rozhodnutia. Pre vykonávanie dobrých rozhodnutí pri riadení procesov, či už v oblasti podnikania alebo v osobnom živote, potrebujeme mať prístup k dátam, v čo najväčšej kvalite a kvantite. Už len na získavanie týchto dát jednotlivec vynaloží veľké množstvo času a zdrojov, no výsledok nemusí byť vždy uspokojivý.

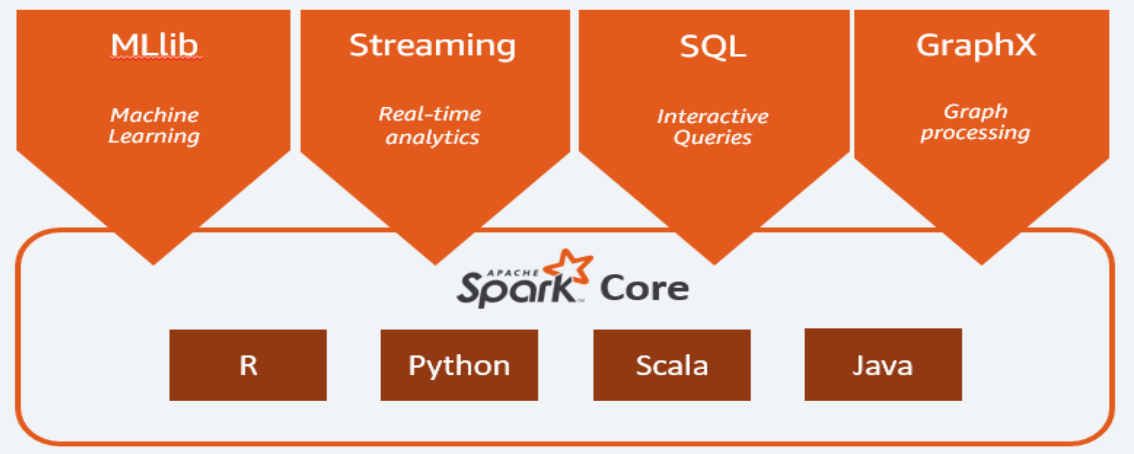
Strojové učenie rieši otázku ako vyvodzovať závery a rozhodnutia z dostupných dát na základe skúseností podobne ako to robí človek. Je to jedna z najrýchlejšie rastúcich technických oblastí súčasnosti, ktorá leží na priesečníku informatiky a štatistiky a je jadrom umelej inteligencie a vedy o údajoch. V súčasnoti existuje na trhu veľké množstvo technologických riešení, ktoré umožňujú implementáciu strojového učenia a jedným z nich je aj knižnica Spark ML, patriaca pod open-source analytický framewrok Spark, ktorej prieskum bude predmetom tejto práce.

1 Apache Spark

Apache Spark je viacjazyčný open-source analytický nástroj, ktorý slúži na spracovanie veľkých objemov dát. Nástroj využíva ukladanie do vyrovnávacej pamäte a optimalizované vykonávanie dotazov na rýchle analytické dotazy na údaje akejkoľvek veľkosti. Taktiež poskytuje rozhranie na programovanie klastrov s implicitným dátovým paralelizmom a odolnosťou voči chybám. Poskytuje vývojové API v jazykoch Java, Scala, Python a R a podporuje opätovné použitie kódu v rámci viacerých pracovných zaťažení – dávkové spracovanie, interaktívne dotazy, analýzy v reálnom čase, strojové učenie a spracovanie grafov.

1.1 Komponenty Apache Spark

Apache Spark nástroj sa skladá z viacerých komponentov, ktoré sú postavené nad jadrom celého nástroja, ktorý sa nazýva Spark Core. Túto štruktúru je možné vidieť aj na schéme nižšie.



Obr. 1: Apache Spark components

Spark Core je základom celého nástroja Spark. Je zodpovedný za správu pamäte, obnovu po poruche, plánovanie, distribúciu a monitorovanie úloh a interakciu s úložnými systémami, ktoré sú vystavené prostredníctvom aplikačného programovacieho rozhrania (pre Java, Python, Scala, .NET a R) so zameraním na abstrakciu RDD.

Spark SQL je komponent nad Spark Core, ktorý zaviedol dátovú abstrakciu nazývanú DataFrames, ktorá poskytuje podporu pre štruktúrované a pološtruktúrované dáta. Ďalej

umožňuje vykonávať interaktívne dopyty s nízkou latenciou až 100x rýchlejšie ako MapReduce[1]. Zahŕňa nákladovo orientovaný optimalizátor, stĺpcové úložisko a generovanie kódu pre rýchle dotazy pri škálovaní na tisíce uzlov.

Spark Streaming je riešenie v reálnom čase, ktoré využíva schopnosť rýchleho plánovania Spark Core na analýzu streamovania. Prijíma údaje v malých dávkach a umožňuje analýzu týchto údajov pomocou rovnakého aplikačného kódu napísaného pre analýzu dávok. To zvyšuje produktivitu vývojárov, pretože môžu používať rovnaký kód na dávkové spracovanie a na streamovanie aplikácií v reálnom čase. Spark Streaming podporuje údaje zo služieb Twitter, Kafka, Flume, HDFS a ZeroMQ a mnohých ďalších, ktoré sa nachádzajú v ekosystéme Spark Packages.

Spark ML je komponent strojového učenia nad Spark Core, poskytujúca sadu vysokoúrovňových rozhraní API, ktoré používateľom pomáhajú vytvárať a ladiť praktické problémy strojového učenia. Viac informácii o tomto komponente je možné nájsť v nasledujúcej kapitole.

Spark GraphX je distribuovaný rámec na spracovanie grafov postavený nad Spark Core. GraphX poskytuje ETL, prieskumnú analýzu a iteratívne výpočty grafov, ktoré používateľom umožňujú interaktívne zostavovať a transformovať dátovú štruktúru grafu v mierke. Dodáva sa s vysoko flexibilným API a výberom distribuovaných grafových algoritmov.

1.2 Využitie Apache Spark

Apache Spark sa v roku 2017 s 365 000 členmi stretnutia stal jedným z najpopulárnejších systémov na spracovanie veľkých dát. Používajú ho organizácie z akéhokoľvek odvetvia, ako sú banky, telekomunikačné spoločnosti, herné spoločnosti, vládne organizácie. Jeho kvalitné schopnosti využívajú aj všetky veľké technologické giganty, ako sú Apple, Facebook, IBM a Microsoft. Z takých menej známych spoločností to sú FINRA, Yelp, Zillow, DataXu, Urban Institute a CrowdStrike.

[1]MapReduce je programovací model pre spracovanie veľkých objemov dat pomoocu paralelného spracovania.

1.3 Výhody a nevýhody Apache Spark

Apache Spark zmenil svet veľkých dát. Táto distribuovaná výpočtová platforma ponúka výkonnejšie výhody ako akákoľvek iná podobná platforma. Rôzne výhody Apache Spark z neho robia veľmi atraktívny framework pre veľké dáta.

V nasledujúcej tabuľke môžeme vidieť základné výhody a nevýhode Apache Spark. Nižšie pod tabuľkou je uvedený aj krátky popis.

Apache Spark	
Výhody	**Nevýhody**
Rýchlosť	Žiadny proces automatickej optimalizácie
Jednoduchý na použitie	Systém správy súborov
Pokročilá analytika	Málo algoritmov
Dynamickosť	Problém s malými súbormi
Viacjazyčný nástroj	Nehodí sa pre prostredie s viacerými používateľmi
Lepší prístup k veľkému objemu dát	-
Open-source komunita	-

Výhody:

- **Rýchlosť** - najväčšou výhodou Spark je práve jeho rýchlosť, vďaka čomu je veľmi obľúbený. Hovorí sa, že Spark je 100-krát rýchlejší ako Hadoop[2],
- **Jednoduché použitie** - obahuje ľahko použiteľné rozhrania API na prácu s veľkými súbormi dát,
- **Pokročilá analýza** - podporuje strojové učenie, grafové algoritmy, streamovanie dát, SQL dotazy...
- **Dynamickosť** - Spark umožňuje vyvíjať paralelné aplikácie,
- **Viacjazyčný nástroj** - podporuje viac jazykov na písanie kódov - Python, Java, Scala, R,
- **Lepší prístup k veľkému objemu dát** - Apache Spark plánuje vzdelávať viac ako milión nových dátových inžinierov a vedcov.

[2]Hadoop je framework obsahujúcí sadu open-source softvérových komponentov určených na spracovávanie veľkého množstva neštruktúrovaných a distribuovaných dát.

- **Open-source komunita** - celý nástroj Apache Spark je open-source.

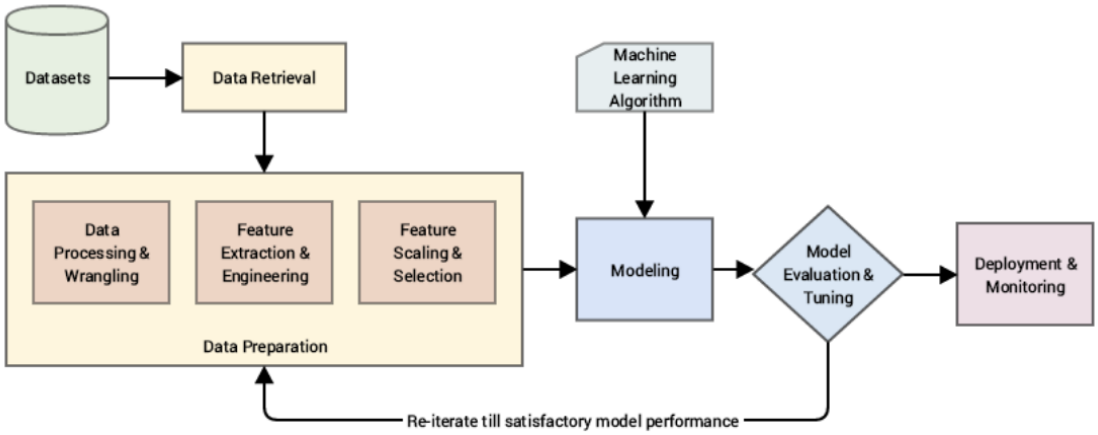
Neýhody:

- **Žiadny proces automatickej optimalizácie** - zatiaľ Spark nedisponuje funkcionalitou na automatickú optimalizáciu, preto je kód potrebné optimalizovať manuálne,
- **Systém správy súborov** - nemá vlastný systém správy súborov. Je závislý od iných platforiem, ako je napríklad Hadoop alebo iné cloudové platformy,
- **Málo algortimov** - Apache Spark MLlib obsahuje málo algoritmov oproti iným známym knižniciam,
- **Problém s malými súbormi** - Spark v spojení s Hadoop má problémy s malými súbormi, pretože Hadoop poskytuje len obmedzený počet veľkých súborov namiesto veľkého počtu malých súborov,
- **Nehodí sa pre prostredie s viacerými používateľmi** - nie je schopný zvládnuť súbežnosť viacerých používateľov.

2 Spark ML

Spark ML je knižnica strojového učenia patriaca pod framework Spark Apache, poskytujúca sadu vysokoúrovňových rozhraní API, ktoré používateľom pomáhajú vytvárať a ladiť praktické problémy strojového učenia [1].

Strojové učenie (z angl. Machine Learning, ďalej len ML) je odvetvie širšej oblasti umelej inteligencie, ktoré využíva štatistické modely na vytváranie predpovedí definovaných problémov. Často sa opisuje ako forma prediktívneho modelovania alebo prediktívnej analýzy a tradične sa definuje ako schopnosť počítača učiť sa bez toho, aby bol na to vyslovene naprogramovaný. V základných technických termínoch ML používa algoritmy, ktoré prijímajú údaje, analyzujú ich a na základe tejto analýzy generujú výstupy. V niektorých prístupoch algoritmy najskôr pracujú s takzvanými „tréningovými údajmi“ a potom sa učia, predpovedajú a nachádzajú spôsoby, ako časom zlepšiť svoj výkon.



Obr. 2: Machine Learning workflow

V súčasnosti Apache Spark disponuje dvoma knižnicami zameranými na ML. Okrem spomínanej knižnice Spark ML, podporuje aj staršiu knižnicu Spark MLlib. Táto knižnica sa zameriava na poskytovanie základnej sady algoritmov. Spark ML je nová knižnica, fungujúca od verzie Spark 1.2, ktorej cieľom je poskytnúť sadu vysokoúrovňových rozhraní API, ktoré používateľom pomáhajú vytvárať a ladiť problémy strojového učenia - od prípravy dát až po vyhodnotenie modelu. Dôležitým rozdielom týchto dvoch knižníc sú typy údajov, na ktorých pracujú. MLlib podporuje RDD, na druhej strane knižnica Spark ML podporuje dátové rámce (z angl. dataframes) a množiny údajov (z angl. datasets). Spark ML je inšpirovaná scikit-learn, a mala by časom úplne nahradiť Spark MLlib, avšak mo-

mentálne nie sú prenesené do nového rozhrania, Spark ML, všetky existujúce algoritmy a funkcie, a preto sa zatiaľ tieto knižnice používajú paralelne [2].

Spark ML štandardizuje rozhrania API pre algoritmy strojového učenia a uľahčuje tak kombinovanie viacerých algoritmov do jedného kanálu. V tejto časti predstavíme kľúčové koncepty rozhrania Spark ML API:

- **Dataset:** v Spark ML používa RDD schémy. RDD schémy sú zložené riadkové objekty popisujúce dátové typy každého stĺpca v riadku. SchemaRDD je podobná tabuľke v tradičnej relačnej databáze.
- **Transformator:** je algoritmus, ktorý dokáže transformovať jednu RDD schému na inú. Jedná sa napríklad o transformovanie vstupných dát do potrebného formátu, normalizáciu, škálovanie, transformáciu vstupných atribútov na výstupné a podobne.
- **Estimator:** je učiaci sa algoritmus, ktorý sa trénuje na dáta a vracia výsledný model.
- **Evaluator:** posudzuje výkonnosť modelu na základe definovaných metrík.
- **Pipeline:** spája viacero transformátorov a estimatorov a určuje výsledný workflow ML modelu.

2.1 Výhody a nevýhody knižnice Spark ML

Ak disponujeme obrovským množstvom dát (terabajtové, petabajtové datasety), na ktorých potrebujeme natrénovať model strojového učenia, klasické pythonovské knižnice ako napríklad scikit-learn nie sú schopné spracovať také množstvo údajov. Práve pre takéto prípdy je vhodné použiť knižnicu Spark ML. Hlavné výhody Spark ML sú [3]:

1. Ľahko sa dá integrovať s ostatnými komponentami Apache Spark.
2. Funguje ako súčasť Java alebo Scala aplikácií, tzn. je možné využiť v rámci jednej aplikácie napríklad Spring Boot a Spark ML.
3. Podporuje najpoužívanejšie a najznámejšie algoritmy strojového učenia, potrebné na väčšinu prípadov použitia.

4. Väčšina metód má rovnaké pomenovanie a použitie ako metódy v populárnej knižnici scikit-learn, a preto sa s ňou jednoducho pracuje a dá sa rýchlo pochopiť.

5. Dokáže bez problémov spracovať terabajtové až petabajtové súbory údajov.

Ako všetko na svete, ani knižnica Spark ML nie je výminkou a nemá len samé výhody a čelí viacerým nedostatkom [4]:

1. Nepodporuje všetky modely strojové učenia ako napríklad boosting a bagging a tradičné neurónové siete (CNN, RNN).

2. Má obmedzenú podporu funkcií Pandas Dafarame.

3. Vyžaduje veľa transformácií vstupných údajov na dáta typu ML.

4. Ťažko sa integruje s populárnymi frameworkami Tensorflow a Pytorch.

5. Interne používa staršiu knižnicu Mllib, pretože zatiaľ nedisponuje všetkou potrebnou funkcionalitou.

2.1.1 Porovnanie s inými ML knižnicami

Na základe vyššie uvedených faktov, ktoré vyplynuli výhod a nevýhod sme sa rozhodli porovnať Spark ML s najpopulárnejšími knižnicami, taktiež určenými na riešenie problémov pomocou strojového učenia. Základné vlastnosti jednotlivých knižníc sme pre jednoduchšie pochopenie spracovali do tabuľky nižšie.

	Spark ML	**Tensorflow**	**Scikit-learn**	**Pytorch**
Veľký objem dát	✓	x	x	x
Rýchlosť	✓	x	x	x
Pamäťovo náročné	✓	x	x	x
User-friendly API	✓	✓	✓	✓
Predtrénované modely	✓	✓	✓	✓
Python	✓	✓	✓	✓
Java	✓	✓	x	x
R	✓	✓	✓	✓

Tabuľka 1: Porovanie vybraných ML knižníc

Ako môžme vidieť, knižnice sú veľmi podobné, každá má svoje pre a proti. To, ktorá knižnica je najvhodnejším kanditátom a závisí už len od konkrétneho prípadu použitia.

2.2 Algoritmy

Algoritmus ML je metóda, pomocou ktorej systém umelej inteligencie vykonáva svoju úlohu, inak povedané, predpovedá výstupné hodnoty z daných vstupných dát [5].
Vo všeobecnosti môžme učenie v ML algoritmoch kategorizovať na:

- **učenie s učiteľom** (z angl. supervised learning) používa na trénovanie „oštítkované“ vstupné údaje, t.j. vopred definované príslušné výstupné hodnoty. Takýto typ učenia vieme ešte rozdeliť do dvoch dôležitých tried na klasifikáciu a regresiu.
- **učenie bez učiteľa** (z angl. unsupervised learning) pracuje s údajmi, ktoré nie sú nijakým spôsobom klasifikované ani „oštítkované“. Predpovede vytvára na základe zistených podobností a rozdielov.

2.2.1 Klasifikácia

Klasifikácia je proces predpovedania triedy daných vstupných údajov. Jedná sa o prediktívne modelovanie, kde aproximačná funkcia f mapuje vstupné premenné X na výstupné premenné Y. Model je trénovaný na trénovacích údajov, pričom sa jeho presnosť vyhodnocuje na testovacích údajoch. Klasifikátor využíva vstupné tréningové atribúty, aby pochopil, ako dané premenné spolu súvisia.

Logistická regresia. Logistická regresia je populárna metóda na predpovedanie kategorických problémov. Ide o špeciálny prípad zovšeobecnených lineárnych modelov, ktoré predpovedajú pravdepodobnosť výsledkov. V spark.ml môže byť logistická regresia použitá na predpovedanie binárneho výsledku pomocou binomickej logistickej regresie alebo môže byť použitá na predpovedanie viactriedneho výsledku pomocou multinomickej logistickej regresie. Pri použití binárnej klasifikácie je potrebné nastaviť parameter family="multinomial".

Rohodovacie stromy. Rozhodovacie stromy a ich súbory sú obľúbené metódy pre úlohy strojového učenia ma riešenie klasifikačných aj regresných problémov. Sú často používané, pretože sú ľahko interpretovateľné, zvládajú kategorické prvky, rozširujú sa na nastavenie klasifikácie viacerých tried, nevyžadujú škálovanie prvkov a sú schopné zachytiť nelinearity a interakcie prvkov. Medzi rozhodovacie stromy patria klasifiátory: *Náhodné lesy a Gradient-boosted klasifikátory.*

Náhodné lesy. Náhodné lesy sú jedným z najúspešnejších modelov strojového učenia na klasifikáciu aj regresiu. Kombinujú veľké množstvo rozhodovacích stromov. Trénujú množinu rozhodovacích stromov oddelene, takže trénovanie je možné vykonávať paralelne. Algoritmus vkladá do tréningového procesu náhodnosť, takže každý rozhodovací strom je trochu iný. Kombinácia predpovedí z každého stromu znižuje rozptyl predpovedí, čím sa zlepšuje výkon na testovacích údajoch. Na to, aby náhodný les vedel vytvoriť výslednú predikciu, musí agregovať predpovede zo svojej sady rozhodovacích stromov:

- pri klasifikácii vyhráva väčšina - predpoveď každého stromu sa počíta ako jeden hlas. Predpovedaný štítok, bude patriť do triedy, ktorá získa najviac hlasov.
- pri regresii sa využíva priemerovanie - výsledná predpovedaná trieda bude priemerom všetkých stromových predpovedí.

Parametre, ktoré sú často najlepšie indikátory výkonu:

- **numTrees**: počet rozhodovaích stromov v náhodnom lese. S väščím počtom stromov sa zvyšuje presnosť modelu a čas trénovania narastá približne lineárne.
- **maxDepth**: maximálna hĺbka každého stromu v náhodnom lese. Pri väčšej hĺbke sú náhodné lesy presnejšie, no na druhej strane sa dlhšie trénujú a sú náchylnejšie na pretrénovanie.

Gradient-boosted klasifikátor. Gradient-Boosted klasifikátor iteratívne trénuje rozhodovacie stromy, s cieľom minimalizovať stratovú funkciu. Pri každej iterácii používa aktuálny súbor na predpovedanie triedy a potom porovná predpoveď so skutočným výsledkom. Implementácia spark.mllib podporuje Gradient-boosted klasifikátor pre len binárnu klasifikáciu a regresiu, zatiaľ nepodporuje viactriednu klasifikáciu.
Najdôležité parametre, ktoré je vhodné pri definovaní Gradient-boosted klasifikátora definovať sú:

- **loss**: funkcia, ktorá hovorí o chybovosti modelu. Typy loss funckii, ktoré môžme použiť pri implementácii Gradient-boosted klasifikátora sú znázornené v tabuľke 2.
- **numIterations**: definuje počet stromov v súbore. Každá iterácia vytvára jeden rozhodovací strom. Zvýšením tohto parametra, by sme mali dosiahnuť lepšiu presnosť modelu na tréningových dátach, no na druhej strate pri veľmi veľkej hodnote presnoť môžme na testovacích dátach výrazne znižiť.

- **learningRate**: definuje rýchlosť učenia modelu, je potrebné ho ladiť podľa prípadu použitia.
- **algo**: algoritmus ktorý sa má použiť na riešenie úlohy (klasifikácia alebo regresia), nastavením parametra "strategy".

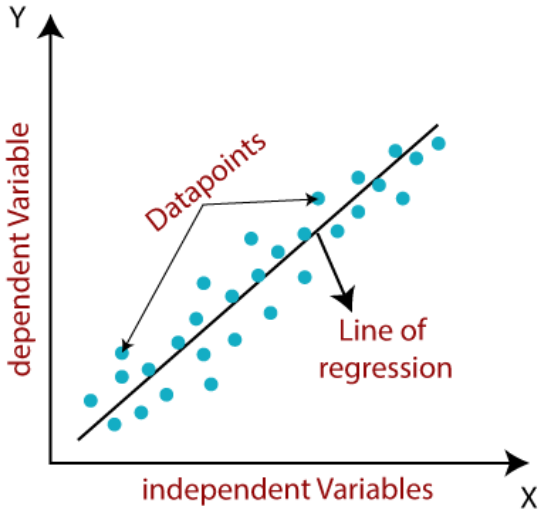
Chybová funkcia	Algoritmus	Popis
Log Loss	Klasifikácia	Čím viac sa predpovedaná pravdepodobnosť líši od skutočnej hodnoty, tým je hodnota log-straty vyššia.
Squared Error/L2	Regresia	Hovorí ako blízko je regresná čiara k množine údajov, v ideálnom prípade by sa mala hodnota priblížiť k nule.
Absolute Error/L1	Regresia	Používa sa na minimalizáciu chyby, ktorá je súčtom všetkých absolútnych rozdielov medzi skutočnou hodnotou a predpovedanou hodnotou.

Tabuľka 2: Chybové funkcie

2.2.2 Regresia

Popri klasifikácii je regresia jednou z hlavných kategórií ML učenia s učiteľom. Zatiaľ čo klasifikácia je kategorizácia objektov na základe naučených vlastností, regresia je metóda na pochopenie vzťahu medzi nezávislými premennými alebo znakmi a závislou premennou alebo výsledkom. Po odhadnutí vzťahu medzi nezávislými a závislými premennými je možné predpovedať výsledky. Vo všeobecnosti zahŕňa vykreslenie najlepšej línie cez dátové body. Vzdialenosť medzi každým bodom a čiarou sa snažime minimalizovať, aby sme dosiahli čo najlepší model.

Lineárna regresia. Lineárna regresia je jedným z najjednoduchších a najpopulárnejších algoritmov strojového učenia. Je to štatistická metóda, ktorá sa používa na predikčnú analýzu. Algoritmus lineárnej regresie ukazuje lineárny vzťah medzi závislou premennou y a jednou alebo viacerými nezávislými x premennými. Keďže lineárna regresia ukazuje lineárny vzťah, to znamená, že zisťuje, ako sa mení hodnota závislej premennej podľa hodnoty nezávislej premennej.



Obr. 3: Lineárna regresia

Implementácia lineárnej regresie v Spark ML podporuje dva typy chybových funkcií: **Squared Error** a **Huber**, čo je hybrid chybových funkcií Squared a Absolute Error, ktoré sú popísané v 2.

2.3 Dátové typy

MLlib podporuje lokálne vektory a matice uložené na jednom počítači, ako aj distribuované matice podporované jedným alebo viacerými RDD. Lokálne vektory a lokálne matice sú jednoduché dátové modely, ktoré slúžia ako verejné rozhrania. Trénovací vzor používaný pri učení pod dohľadom sa v MLlib nazýva tzv. öznačený bod"(angl. labeled sample). V MLlib rozlišujeme 4 základné dátové typy.

2.3.1 Lokálny vektor

MLlib podporuje dva typy lokálnych vektorov: husté (angl. dense) a riedke (angl. sparse).

Hustý vektor je podporovaný dvojitým poľom reprezentujúcim jeho vstupné hodnoty. Riedky vektor je podporovaný dvoma paralelnými poľami: indexy a hodnoty. Riedke vektory sa používajú, keď je väčšina čísel nulová. Pre vytvorenie riedkeho vektora je potrebné zadať dĺžku vektora - indexy nenulových hodnôt, ktoré by mali byť striktne rastúce a nenulové hodnoty.

Napríklad vektor (1.0, 0.0, 3.0) môže byť reprezentovaný v hustom formáte ako [1.0, 0.0, 3.0] alebo v riedkom formáte ako (3, [0, 2], [1.0, 3.0]), kde 3 je veľkosť vektora.

```
from pyspark.mllib.linalg import Vectors

## DENSE VECTOR - hustý vektor
print(Vectors.dense([1,2,3,4,5,6,0]))
# >> DenseVector([1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 0.0])

### SPARSE VECTOR - riedky vektor
### Vectors.sparse( length, index_of_non_zero_values, non_zero_values)
print(Vectors.sparse(10, [0,1,2,4,5], [1.0,5.0,3.0,5.0,7]))
# >> SparseVector(10, {0: 1.0, 1: 5.0, 2: 3.0, 4: 5.0, 5: 7.0})
print(Vectors.sparse(10, [0,1,2,4,5], [1.0,5.0,3.0,5.0,7]).toArray())
# >> array([1., 5., 3., 0., 5., 7., 0., 0., 0., 0.])
```

Obr. 4: Využitie lokálneho vektora v MLlib

2.3.2 Označený bod (angl. Labeled point)

Označený bod je lokálny vektor, pričom sa jedná o hustý alebo riedky vektor spojený s labelom. V MLlib sa označené body používajú najmä v algoritmoch učenia s učiteľom (učenie pod dohľadom). Na uloženie labelu používame premennú typu Double, takže označené body môžeme použiť v regresii aj v klasifikácii. Pri binárnej klasifikácii by mal byť label buď 0 (negatívny), alebo 1 (pozitívny). Pri klasifikácii viacerých tried by mali byť labelmi indexy tried začínajúce od nuly.

```
from pyspark.mllib.regression import LabeledPoint
# hustý vektor, nastavenie labelu
example = LabeledPoint(1,Vectors.dense([1,2,3,4,5]))
# label
print(example.label)
# features
print(example.features)
```

Obr. 5: Využitie označeného bodu v MLlib

2.3.3 Lokálna matica

Lokálna matica má celočíselné indexy riadkov a stĺpcov a dvojčíselné hodnoty uložené na jednom zariadení (počítači). MLlib podporuje husté matice, ktorých vstupné hodnoty sú

uložené v jednom dvojnásobnom poli v stĺpci.

V prípade využitia riedkej matice musia byť nenulové vstupné hodnoty uložené vo tzv. „Compressed Sparse Column“ formáte.

```
# import funkcii pre matice
from pyspark.mllib.linalg import Matrices

# DENSE - vytvorenie hustej matice
matrix_1 = Matrices.dense(3, 2, [1,2,3,4,5,6])
print(matrix_1)
# >> DenseMatrix(3, 2, [1.0, 2.0, 3.0, 4.0, 5.0, 6.0], False)
print(matrix_1.toArray())
"""
>> array([[1., 4.],
          [2., 5.],
          [3., 6.]])
"""

# SPARSE - vytvorenie riedkej matice
matrix_2 = Matrices.sparse(3, 3, [0, 1, 2, 3], [0, 0, 2], [9, 6, 8])
print(matrix_2)
# SparseMatrix(3, 3, [0, 1, 2, 3], [0, 0, 2], [9.0, 6.0, 8.0], False)
print(matrix_2.toArray())
"""
>> array([[9., 6., 0.],
          [0., 0., 0.],
"""
```

Obr. 6: Využitie lokálnej matice v MLlib

Napríklad nižšie uvedená matica je uložená v jednorozmernom poli [1.0, 3.0, 5.0, 2.0, 4.0, 6.0] s veľkosťou matice (3, 2).

$$\begin{pmatrix} 1.0 & 2.0 \\ 3.0 & 4.0 \\ 5.0 & 6.0 \end{pmatrix}$$

Obr. 7: Vzor lokálnej matice

2.3.4 Distribuovaná matica

Distribuovaná matica má riadkové a stĺpcové indexy typu Long a hodnoty typu Double, ktoré sú distribuovane uložené v jednom alebo viacerých RDD. Na ukladanie veľkých a distribuovaných matíc je veľmi dôležité zvoliť správny formát. Konverzia distribuovanej matice do iného formátu môže vyžadovať globálne premiešanie, ktoré je pomerne nákladné.

Pozn.: Základ RDD distribuovanej matice musí byť deterministický, pretože veľkosť matice ukladáme do vyrovnávacej pamäte. Vo všeobecnosti môže použitie nedeterministických RDD viesť k chybám.

Doposiaľ boli implementované tri typy distribuovaných matíc.

RowMatrix Typ matice RowMatrix je riadkovo orientovaná distribuovaná matica bez zmysluplných indexov riadkov. Podporuje RDD svojich riadkov, kde každý riadok predstavuje lokálny vektor. Vzhľadom na to, že každý riadok je reprezentovaný lokálnym vektorom, počet stĺpcov je obmedzený rozsahom celých čísel.

Vhodným využitím matice RowMatrix je napríklad algoritmus Random Forest, kde algoritmus rozdeľuje riadky pre vytvorenie viacerých stromov. Výsledok jedného stromu nie je závislý od ostatných stromov, preto je možné využiť distribuovanú architektúru a vykonávať paralelné spracovanie algoritmov, ako je uvedený Random Forest algoritmus pre veľké množstvo údajov.

```
# distribuovaný dátový typ - RowMatrix
from pyspark.mllib.linalg.distributed import RowMatrix
# vytvorenie RDD
rows = sc.parallelize([[1,2,3], [4,5,6], [7,8,9], [10,11,12]])
# vytvorenie distribuovanej RowMatrix
row_matrix = RowMatrix(rows)
print(row_matrix)
# >> <pyspark.mllib.linalg.distributed.RowMatrix at 0x7f425884d7f0>
print(row_matrix.numRows())
# >> 4
print(row_matrix.numCols())
# >> 3
```

Obr. 8: Využitie RowMatrix v MLlib

IndexedRowMatrix IndexedRowMatrix je podobný typ ako RowMatrix, s rozdielom, že v tomto prípade sú významné indexy riadkov (usporiadaný spôsob). Každému riadku je priradená hodnota indexu. Je podporovaná RDD indexovaných riadkov, takže každý riadok je reprezentovaný svojím indexom (typu Long) a lokálnym vektorom.

Vhodným využitím sú algoritmy, kde je poradie dôležité, ako sú napríklad údaje časových radov.

```
# importovanie IndexedRowMatrix
from pyspark.mllib.linalg.distributed import IndexedRow,
    IndexedRowMatrix
# vytvorenie RDD
indexed_rows = sc.parallelize([
    IndexedRow(0, [0,1,2]),
    IndexedRow(1, [1,2,3]),
    IndexedRow(2, [3,4,5]),
    IndexedRow(3, [4,2,3]),
    IndexedRow(4, [2,2,5]),
    IndexedRow(5, [4,5,5])
])
# vytvorenie IndexedRowMatrix
indexed_rows_matrix = IndexedRowMatrix(indexed_rows)
print(indexed_rows_matrix.numRows())
# >> 6
print(indexed_rows_matrix.numCols())
# >> 3
```

Obr. 9: Využitie IndexedRowMatrix v MLlib

CoordinateMatrix CoordinateMatrix, inak opísaná ako súradnicová matica, je distribuovaná matica podporovaná RDD jej položiek. Každá položka je typu tuple (i: Long, j: Long, value: Double), pričom i je index riadku, j je index stĺpca a value je hodnota položky. CoordinateMatrix by sa mala používať len vtedy, keď sú obe dimenzie matice veľmi veľké a matica je veľmi riedka.

```
# importovanie CoordinateMAtrix
from pyspark.mllib.linalg.distributed import CoordinateMatrix,
    MatrixEntry
# vytvorenie RDD usporiadanych vstupou pomocou triedy MatrixEntry:
matrix_entries = sc.parallelize(
[MatrixEntry(0, 5, 2),
MatrixEntry(1, 1, 1),
MatrixEntry(1, 5, 4)])
# vytvorenie CoordinateMatrix z RDD pomocou MatrixEntries.
c_matrix = CoordinateMatrix(matrix_entries)
# pocet stlpcov
print(c_matrix.numCols())
# >> 6
# pocet riadkov
print(c_matrix.numRows())
# >> 2
```

Obr. 10: Využitie CoordinateMatrix v MLlib

Pozn.:
V praxi sa využíva taktiež bloková matica, ktorá sa však nepovažuje ako samostatný typ distribuovanej matice, ale v takejto matici dokážeme ukladať rôzne čiastkové matice veľkej matice na rôznych zariadeniach (počítačoch). Potrebné je určiť rozmery bloku, ako napríklad v nasledujúcom príklade, kde je uvedený rozmer 3x3 a pre každý z blokov je možné špecifikovať maticu zadaním súradníc.

```python
# import the libraries
from pyspark.mllib.linalg import Matrices
from pyspark.mllib.linalg.distributed import BlockMatrix
# vytvorenie RDD pomocou blokov hustých sub-matic
blocks = sc.parallelize(
[((0, 0), Matrices.dense(3, 3, [1, 2, 1, 2, 1, 2, 1, 2, 1])),
 ((1, 1), Matrices.dense(3, 3, [3, 4, 5, 3, 4, 5, 3, 4, 5])),
 ((2, 0), Matrices.dense(3, 3, [1, 1, 1, 1, 1, 1, 1, 1, 1]))])
# vytvorenie blokovej matice BlockMatrix z RDD pomocou sub-matíc blokov velkosti 3x3
b_matrix = BlockMatrix(blocks, 3, 3)
# pocet stlpcov na blok
print(b_matrix.colsPerBlock)
# >> 3
# pocet riadkov na blok
print(b_matrix.rowsPerBlock)
# >> 3
# konvertovanie bloku matice na lokalnu maticu
local_mat = b_matrix.toLocalMatrix()
# vypisanie lokalnej matice
print(local_mat.toArray())
"""
>> array([[1., 2., 1., 0., 0., 0.],
          [2., 1., 2., 0., 0., 0.],
          [1., 2., 1., 0., 0., 0.],
          [0., 0., 0., 3., 3., 3.],
          [0., 0., 0., 4., 4., 4.],
          [0., 0., 0., 5., 5., 5.],
          [1., 1., 1., 0., 0., 0.],
          [1., 1., 1., 0., 0., 0.],
          [1., 1., 1., 0., 0., 0.]])
"""
```

Obr. 11: Využitie BlockMatrix v MLlib

2.4 Metriky

MLlib poskytuje množstvo algoritmov strojového učenia, ktoré je možné využiť na učenie z údajov a vytváranie predpovedí na ich základe. Keď sa tieto algoritmy použijú na vytvorenie modelov strojového učenia, je nevyhnutné vyhodnotiť správnosť modelu na základe určitých kritérií. MLlib poskytuje okrem algoritmov taktiež aj súbor metrík na účely vyhodnotenia správnosti modelov strojového učenia. Nižšie sú uvedené základné metriky využívané pre evaluáciu modelov strojového učenia.

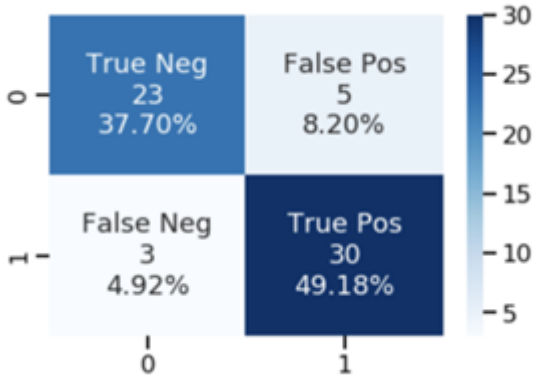
2.4.1 Konfúzna matica

Konfúzna matica slúži na meranie presnosti klasifikačného modelu. Jedná sa o maticu výkonnosti na meranie klasifikačných modelov, ktorých výstup je binárny alebo viactriedny. Má tabuľku 4 rôznych kombinácií, pričom zvažujeme hodnoty:

- **True Positive (TP)** - označenie je pozitívne a predpoveď je tiež pozitívna
- **True Negative (TN)** - označenie je negatívne a predpoveď je tiež negatívna

- **False Positive (FP)** - označenie je negatívne, ale predpoveď je pozitívna
- **False Negative (FN)** - označenie je pozitívne, ale predpoveď je negatívna

Ako môžeme vidieť aj na nižšie uvedenej ukážkovej matici, model je presnejší čím viac sa hodnoty FP a FN približujú k 0, naopak počet TN a TP sa približuje k vysokej hodnote.



Obr. 12: Príklad konfúznej matice

Miera je merným faktorom v konfúznej matici. Má tiež 4 typy TPR, FPR, TNR, FNR.

- **Pravdivá pozitívna miera (TPR)** - pravdivá pozitívna/pozitívna
- **Falošne pozitívna miera (FPR)** - falošne pozitívny/negatívny
- **Falošne negatívna miera (FNR)** - falošne negatívna/pozitívna
- **Pravá negatívna miera (TNR)** - pravdiv8 negatívna/negatívna

Pre lepší výkon by mali byť TPR, TNR vysoké a FNR, FPR nízke.

Presnosť (angl. Precission) Presnosť (nazývaná aj pozitívna prediktívna hodnota) je podiel relevantných prípadov medzi získanými prípadmi. Presnosť by mala nadobúdať hodnotu 1 (alebo sa k nej približovať), čo znamená, že počet FP je žiadny alebo nízky. Tento výsledok hovorí o spoľahlivosti predikcie modelu.

Vzorec pre výpočet presnosti je: Precision = True Positive / Predicted Positive

Spätná väzba (angl. Recall) Spätná väzba (známa aj ako citlivosť) je podiel relevantných prípadov, ktoré boli získané. Spätná väzba by mala nadobúdať hodnotu 1 (alebo sa k nej približovať), čo znamená, že počet FN je žiadny alebo nízky.

Vzorec pre výpočet spätnej väzby je: Recall = True Positive / Actual Positive

ROC krivka (angl. Receiver Operating Characteristics Curve) ROC krivka je graf, ktorý popisuje kvalitu binárneho klasifikátoru v závislosti od nastavení jeho klasifikačného prahu. Krivka ROC ukazuje pomer medzi citlivosťou (alebo TPR) a špecifickosťou (1 - FPR), pričom TPR = TP/P a FPR = FP/N.

Pri správne natrénovanej sieti, by sa mala krivka ťahať smerom k ľavému hornému rohu. Pri zle natrénovanej sieti sa krivky približujú k diagonále, prípadne klesajú pod ňu. Ak sa krivky prekrývajú na diagonále, jedná sa o vysokú stratu.

Obr. 13: Príklad ROC krivky

F-skóre (angl. F-Measure) Používa sa na meranie presnosti testovania. Je to vážený priemer presnosti a spätnej väzby. Keď nadobúda F1 skóre hodnotu 1 dosahujeme najlepší výsledok, naopak pri hodnote 0 sa jedná o zlé skóre .

Vzorec pre výpočet skóre je: F1 = 2 * (precision * recall) / (precision + recall)

3 Implementácia

Na demonštráciu práce s knižnicou Spark ML sme najprv potrebovali nájsť vhodný súbor údajov, ktorý by sme vedeli náležite analyzovať. Datasety údajov z reálneho života sú voľne dostupné na stránke www.kaggle.com. Našim cieľom bolo spoznať dáta, predpripraviť ich na trénovanie a rôznymi prístupmi nájsť najoptimálnejšie riešenie. Zamerali sme sa na odprezentovanie viacerých algoritmov na riešenie binárneho problému.

3.1 O datasete

Prvým krokom pri vytvorení úspešného modelu je oboznámenie sa s datasetom a problémom, ktorý budeme riešiť. Súbor údajov pochádza z úložiska UCI Machine Learning a súvisí s priamymi marketingovými kampaňami (telefonáty) portugalskej bankovej inštitúcie. Cieľom klasifikácie je predpovedať, či klient upíše alebo neupíše termínovaný vklad. K dispozícií máme údaje o ľuďoch v banke, ktorí dostali túto ponuku. Dataset obsahuje viac ako 40 000 respondentov, z ktorých iba 5% bolo ochotných založiť si takýto termínovaný vklad. Budeme sa snažiť vytvoriť model, ktorý na základe daných atribútov odhadne záujem o termínovaný vklad. Cieľom je teda uľahčiť prácu ľudom v banke a oslovovať len potencionálnych záujemcov. Problém budeme riešiť cez rôzne prístupy binárnej klasifikácie na demonštrovanie úspešnosti jednotlivých modelov.

	0	1	2	3	4
age	44	53	28	39	55
job	blue-collar	technician	management	services	retired
marital	married	married	single	married	married
education	basic.4y	unknown	university.degree	high.school	basic.4y
default	unknown	no	no	no	no
housing	yes	no	yes	no	yes
loan	no	no	no	no	no
contact	cellular	cellular	cellular	cellular	cellular
month	aug	nov	jun	apr	aug
day_of_week	thu	fri	thu	fri	fri
duration	210	138	339	185	137
campaign	1	1	3	2	1
pdays	999	999	6	999	3
previous	0	0	2	0	1
poutcome	nonexistent	nonexistent	success	nonexistent	success
emp_var_rate	1.4	-0.1	-1.7	-1.8	-2.9
cons_price_idx	93.444	93.2	94.055	93.075	92.201
cons_conf_idx	-36.1	-42.0	-39.8	-47.1	-31.4
euribor3m	4.963	4.021	0.729	1.405	0.869
nr_employed	5228.1	5195.8	4991.6	5099.1	5076.2
deposit	0	0	1	0	1

Obr. 14: Ukážka datasetu banking.csv zo stránky www.kaggle.com

Na obrázku č.15 vidíme prvých 5 riadkov zo súboru dát. Atribúty popisujú údaje ako napríklad, či má oslovený zákazník banky pôžičku, aký má príjem, vzdelanie a podobne. Presný opis údajov a ich vysvetlenie, čo presne znamenajú máme z oficiálnej stránky. Ďalším krokom pri práci s datasetom je spracovanie nevalidných, chybných a hraničných hodnôt (ang. outliers).

3.2 Úprava dát

V ďalšom kroku potrebujeme vedieť, či dataset obsahuje nejaké nevalidné dáta, nulové hodnoty, či ide o vyvážený alebo nevyvážený dataset. Nutná je teda kontrola dát a ich spracovanie. Analýzou sme zistili, že triedy nášho datasetu sú nevyvážené, iba 5% respondentov bolo ochotných založiť si termínovaný vklad. Našli a spracovali sme nevalidné dáta.

Napríklad atribút neplatič chýbal až v 8000 prípadoch. Okrem validácie údajov sme pracovali aj s hraničnými hodnotami. Sú to tie údajové body, ktoré sa výrazne líšia od zvyšku súboru údajov. Sú to často abnormálne pozorovania, ktoré skresľujú distribúciu údajov a vznikajú v dôsledku nekonzistentného zadávania údajov alebo chybných pozorovaní.

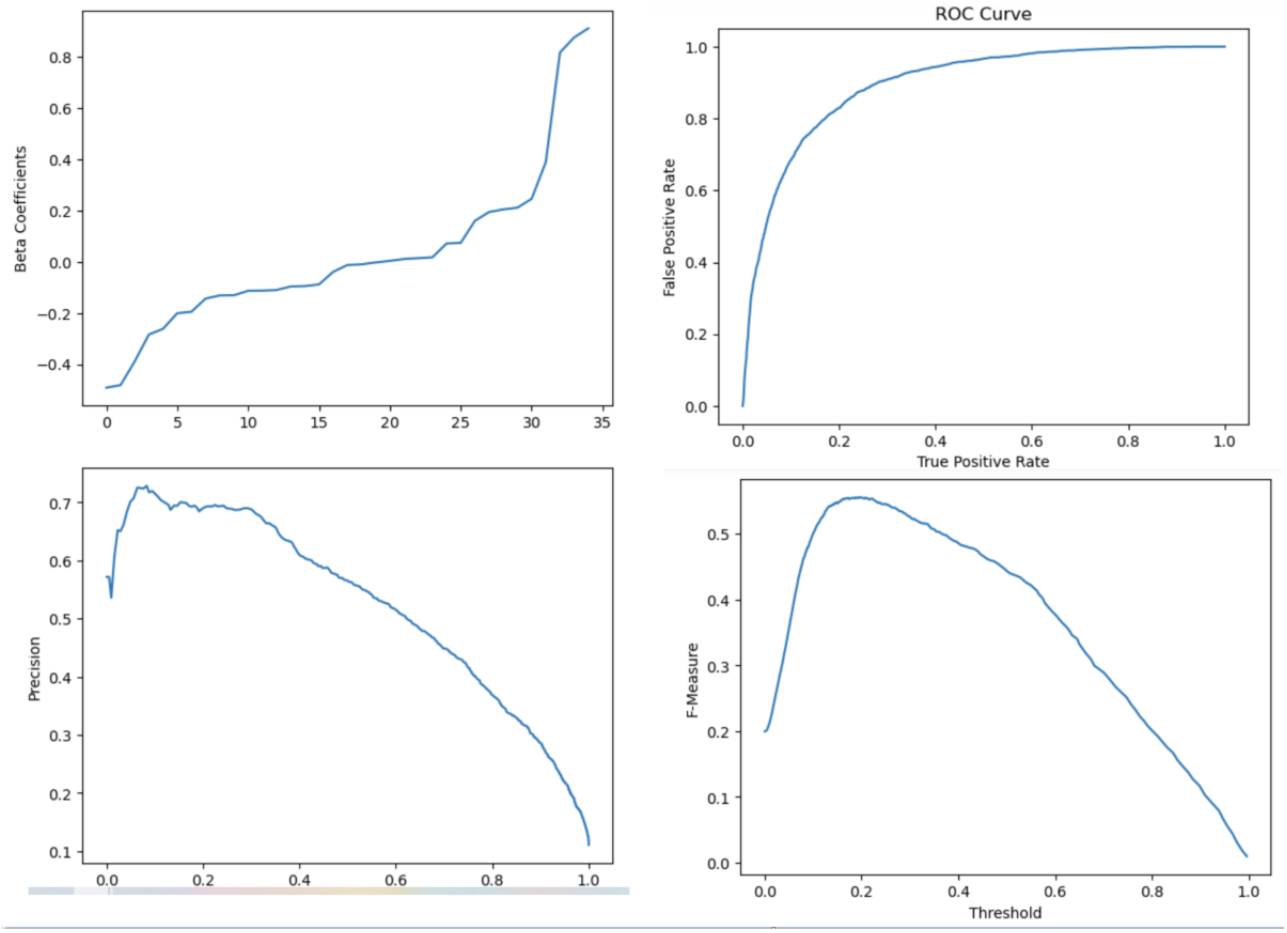
3.3 Príprava dát na trénovanie

Počítače sú navrhnuté tak, aby zvládli číslice ako 0 a 1, ale nerozumejú textu. V pysparku sme využili dve metódy na proces konverzie: String Indexer a OneHotEncoder. Ak prvok obsahuje iba dve kategórie/skupiny, v takom prípade môžeme na konverziu priamo použiť metódu StringIndexer. Najprv je potrebné použiť StringIndexer pred OneHotEncoderom, pretože OneHotEncoder potrebuje ako vstup stĺpec indexov kategórií. Následne používame OneHotEncoder, ktorý prevádza kategorické hodnoty do nového kategorického stĺpca a týmto stĺpcom priraďuje binárnu hodnotu 1 alebo 0. Nakoniec Pipeline zreťazí viaceré transformátory a odhadovače, aby sme špecifikovali náš pracovný postup strojového učenia. Etapy pipeline sú špecifikované ako usporiadané pole. Po konverzii náhodne rozdeľujeme údaje na trénovacie a testovacie dáta v pomere 7:3 pomocou generátora, ktorému nastavujeme inicializačný stav.

3.4 Použité algoritmy

3.4.1 Logistická regresia

Vďaka predpripraveným dátam sme vytvorili model logistickej regresie. Na základe rôznych metrík sme vyhodnocovali pripravený model. Aby sme zvýšili výkonnosť modelu potrebovali sme nájsť vhodné parametre. Pyspark nám umožnil použiť ParamGridBuilder a implementovať krížovú validáciu.



Obr. 15: Vyhodnotenie modelu logisticke regresia na základe metrík

3.5 Rozhodovacie stromy

Rozhodovacie stromy sú široko používané, pretože sú ľahko interpretovateľné, zvládajú kategorické vlastnosti, rozširujú sa na nastavenie klasifikácie viacerých tried, nevyžadujú škálovanie prvkov a sú schopné zachytiť nelinearity a interakcie prvkov. V rámci rozhodovacích stromov sme vyskúšali tri algoritmy. Medzi ne patria rozhodovací strom, náhodný

les a Gradient-boosted Tree klasifikátor. Rozhodovací strom, tak ako sme predpokladali dosahoval veľmi nepresné výsledky. Hodnota Roc Curve nám vyšla len 0.29, čo je oproti lineárnej regresii podstatne horší výsledok. Je to spôsobené tým, že rozhodovacie stromy sú nestabilnejšie a vyžadujjú často dlhší čas na trénovanie. Pre takéto testovacie účely sme si to časovo nemohli dovoliť a vyskúšali sme model iba na jednom rozhodovacom strome. Jeden jednoduchý rozhodovací strom nie je veľmi presný, pretože je príliš slabý vzhľadom na rozsah rôznych funkcií. Presnosť predikcie rozhodovacích stromov sa dá zlepšiť metódami Ensemble. Klasifikátor náhodný les dosahoval hodnotu Roc Curve 0.88085 a Gradient-boosted Tree obstál ešte o niečo lepšie s hodnotou Roc Curve 0.8985. Práve tento klasifikátor nám zo všetkých spomenutých dal najlepší výsledok. Znovu sme vyskúšali vyladenie tohto modelu pomocou ParamGridBuilder a CrossValidator. Predtým sme použili metódu explainParams() na zobrazenie zoznamu všetkých parametrov a ich definícií, aby sme pochopili, ktoré parametre sú k dispozícii na ladenie. Po odladení modelu sme sa dostali na hodnotu Roc Curve 0.903205.

3.6 Zhrnutie

Cieľom implementácie bolo vyskúšať si prácu s knižnicou Pyspark ML. Pripravili sme si riešenie binárneho problému na oficiálnom bankovom datasete. Dáta sme zanalyzovali, zvalidovali a predpripravili na trénovanie pomocou String Indexer a OneHotEncoder. Použili sme pipeline na reťazenie viacerých transformátorov a odhadovačov, aby sme špecifikovali náš pracovný postup strojového učenia. Po rozdelení dát na trénovacie a testovacie sme vytvorili niekoľko typov modelov. Najlepšie výsledky sme dosiahli pomocou Gradient-boosted Tree klasifikátora a jeho doladenie nám umožnil ParamGridBuilder a CrossValidator. Taktiež sme sa oboznámili s rôznymi metrikami ako sú precision, recall, treshold ale smerodajnou pre nás bola hodnota Roc Curve, ktorú sme vo výsledku dosiahli až 0.90320, v pomerne krátkom čase (približne 5 minút trénovania). Náš dataset obsahuje až 40000 údajov a v takto krátkom čase, sme dokázali dosiahnuť celkom dobré výsledky. V porovnaní s inými knižnicami ako napríklad TensorFlow, ktorú sme mali možnosť vyskúšať si na predmete Strojové učenie a neurónové siete, kladne hodnotíme hlavne časovú náročnosť trénovania vzhľadom na dosiahnuté výsledky.
[*]

Záver

V práci sme zanalyzovali knižnicu Spark ML, patriacu pod analytický nástroj Apache Spark, ktorá primárne slúži na spracovanie veľkých objemov dát a rieši definované problémy pomocou strojového učenia. Preskúmali sme jej základné koncepty a funkcionalitu, ktorú sme neskôr zúžitkovali v druhej časti práce. Dominantnou preferenciou Spark ML je rýchlosť, pretože namiesto čítania a zapisovania prechodných údajov na disk, používa RAM (Random Access Memory) vďaka čomu tu bežia algoritmy strojového učenia, v porovnaní s inými technologickými riešeniami, 100-krát rýchlejšie. O jej rýchlosti sme sa presvedčili v praktickej časti práce, pri implementácii softvéru. Keďže sme softvér implementovali v programovacom jazyku Python, využívali sme API PySpark, ktoré je jednoduché na použitie, používateľsky prívetivé a intiutívne.

Spark ML hodnotíme veľmi kladne. Je vhodným kanditátom na použitie pri riešní rôznych prípadov použitia oblasti strojového učenia. Aj napriek tomu, že má Spark niekoľko oblastí, ktoré musí vylepšiť, má pred sebou pre svoj výpočtový výkon a rýchlosť, svetlú budúcnosť a predpokladá sa, že bude v budúcnosti dominantnou platformou na spracovanie veľkých objemov údajov.

Zoznam použitej literatúry

1. *Machine Learning Library.*
   https://spark.apache.org/docs/latest/ml-guide.html.
2. *Spark ML vs MLlib.*
   https://www.oreilly.com/library/view/high-performance-spark/9781491943199/ch09.html.
3. *Good parts of the Apache Spark ML library.*
   https://zaleslaw.medium.com/i-have-been-a-spark-user-since-spark-0-8-adfb0b327338.
4. *Weakness of the Apache Spark ML library.*
   https://zaleslaw.medium.com/weakness-of-the-apache-spark-ml-library-41e674103591.
5. *Classification and regression.*
   https://spark.apache.org/docs/latest/ml-classification-regression.html.
6. *Apache Spark.*
   https://en.wikipedia.org/wiki/Apache_Spark.
7. *Oficiálna stránka Apache Spark.*
   https://spark.apache.org/.
8. *Apache Spark advantages and disadvantages.*
   https://www.knowledgehut.com/blog/big-data/apache-spark-advantages-disadvantages.
9. *Introduction to Apache Spark.*
   https://aws.amazon.com/big-data/what-is-spark/.
10. *PySpark for Beginners.*
    https://www.analyticsvidhya.com/blog/2019/10/pyspark-for-beginners-first-steps-big-data-analysis/?utm_source=av&utm_medium=feed-articles&utm_campaign=feed&fbclid=IwAR36zgbPzi-Un3Md7QYEBGapAyNy8pHbAJlUkJi9AfI
11. *Data Types in Spark MLlib.*
    https://medium.com/analytics-vidhya/data-types-in-spark-mllib-966b4800f893.

12. *MLlib - Data Types.*
https://spark.apache.org/docs/1.2.1/mllib-data-types.html?fbclid=IwAR0l8szJc5ZTv23Ofvlr5FU--7XWE63G8AF50YUDJi0oGapibKOEgvPd5gw.

13. *Evaluation Metrics - RDD-based API.*
https://spark.apache.org/docs/latest/mllib-evaluation-metrics.html?fbclid=IwAR1YejoH2kXbwaPRgDCEp1mypbYwQQK_ezsBDw7CzErkKa_Tds5jnaaGtdo.

14. *Confusion Matric(TPR,FPR,FNR,TNR), Precision, Recall, F1-Score.*
https://medium.datadriveninvestor.com/confusion-matric-tpr-fpr-fnr-tnr-precision-recall-f1-score-73efa162a25f.