

Analiza setului de date Thyroid Disease
pus la dispoziție de University of California Irvine

Proiect Big Data

CUPRINS

1. Prezentarea setului de date și enunțarea obiectivelor
2. Procesarea, pregătirea și transformarea datelor
 - 2.1. Procesarea datelor
 - 2.2. Folosirea unei funcții definite de utilizator
 - 2.3. Analiza datelor
3. Utilizarea SparkMLlib pentru clasificări și regresii
 - 3.1. Standardizarea datelor
 - 3.2. Regresie Logistică
 - 3.3. Random Forest simplu
 - 3.4. Optimizarea hiperparametrilor pentru Random Forest
4. Utilizarea unui Data Pipeline
5. Aplicarea unei metode de Deep Learning, folosind Tensorflow
6. Utilizarea SparkStreaming

1. Prezentarea setului de date și enunțarea obiectivelor

În proiectul dezvoltat s-a folosit un set de date ce cuprinde informații despre diverse simptomatologii ce sunt analizate pentru a se stabili dacă un pacient are o anumită problemă cauzată de glanda tiroidă. Acest set de date poate fi descărcat de pe site-ul Universității din California Irvine, secțiunea Machine Learning Repository și poate fi găsit la adresa următoare: <https://archive.ics.uci.edu/dataset/102/thyroid+disease>. Din arhiva descărcată se vor folosi fișierele *thyroid0387.data* și *thyroid0387.names*. Am extras informațiile din *.names* și am creat un alt fișier în care se află numele fiecărei coloane, în ordine corespunzătoare apariției în *.data*. Acest proces a facilitat crearea DataFrame-ului pe baza căruia se vor aplica operațiile de prelucrare și transformare.

În setul de date vom regăsi informații despre următoarele atribute.

Numele atributului	Valori Posibile
age	întregi
sex	M, F
on thyroxine	f, t
query on thyroxine	f, t
on antithyroid medication	f, t
sick	f, t
pregnant	f, t
thyroid surgery	f, t
I131 treatment	f, t
query hypothyroid	f, t
query hyperthyroid	f, t
lithium	f, t
goitre	f, t
tumor	f, t
hypopituitary	f, t
psych	f, t
TSH measured	f, t
TSH	întregi
T3 measured	f, t
T3	întregi
TT4 measured	f, t
TT4	întregi
T4U measured	f, t
T4U	întregi
FTI measured	f, t

FTI	întregi
TBG measured:	f, t
TBG	întregi
referral source	String
tag id	String

În continuare se vor prezenta ce operații s-au aplicat asupra datelor pentru a procesa și transforma astfel încât să fie pregătite pentru aplicarea unor algoritmi de clasificare și regresie. Se vor folosi Dataframes și Spark SQL pentru etapa de pregătire, iar pentru clasificare și regresie se utilizează SparkMLlib. Se va evidenția folosirea unui Data Pipeline, a unor funcții definite de utilizator, optimizarea hiperparametrilor, aplicarea unei metode de Deep Learning din Tensorflow, dar și folosirea SparkStreaming.

2. Procesarea, pregătirea și transformarea datelor

2.1. Procesarea datelor

Proiectul este dezvoltat în Google Collab. La început se va crea o celulă unde vom importa toate dependențele de care avem nevoie de-a lungul dezvoltării.

- Inițial se creează o sesiune de Spark și se importă setul de date într-un DataFrame.

```
spark =
SparkSession.builder.appName("Project_Thyroid_Disease").getOrCreate()
df_data =
spark.read.csv("/content/drive/MyDrive/MASTER_FMI/Colab_Notebooks/Data/
FinalProject_ThyroidDisease /thyroid0387.data", header=False)
df_names =
spark.read.csv("/content/drive/MyDrive/MASTER_FMI/Colab_Notebooks/Data/
FinalProject_ThyroidDisease /thyroid0387.names", header=False)
df_data = df_data.toDF(*df_names.head())
df_data.show()
```

age	sex	on thyroxine	query on thyroxine	on antithyroid medication	sick	pregnant	thyroid surgery	T131 treatment	query hypothyroid	query hyperthyroid	lithium	goitre	tumor	hypopituitary	psych	TSH measured	TSH	T3 measured	T3	T4 measure
29	F	f		f		f		f		t		f	f	f		f	f	t 0.3		f 2
29	F	f		f		f		f		f		f	f	f		f	f	t 1.6		f 1.9
41	F	f		f		f		f		t		f	f	f		f	f	f 2		f 2
36	F	f		f		f		f		f		f	f	f		f	f	f 2		f 2
32	F	f		f		f		f		f		f	f	f		f	f	f 2		f 2
60	F	f		f		f		f		f		f	f	f		f	f	f 2		f 2
77	F	f		f		f		f		f		f	f	f		f	f	f 2		f 2
28	F	f		f		f		f		f		f	f	f		f	f	t 0.7		t 2.6
28	F	f		f		f		f		f		f	f	f		f	f	t 1.2		t 1.8
28	F	f		f		f		f		f		f	f	f		f	f	t 1.9		t 1.7
54	F	f		f		f		f		f		f	f	f		f	f	t 1.9		t 2.3
42	F	f		f		f		f		f		f	f	f		f	f	t 1		t 1.8
51	M	t		f		f		f		f		f	f	f		f	f	t 0.5		f 2
51	F	f		f		f		f		f		f	f	f		f	f	t 0.7		t 2.4
37	F	f		f		f		f		f		f	f	t		f	f	f 2		t 2.9
16	M	f		f		f		f		f		f	f	f		f	f	t 2.6		f 2
54	F	f		f		f		f		f		f	f	f		f	f	t 1		t 2
43	M	f		f		f		f		f		f	f	f		f	f	f 2		t 2.1
63	F	t		f		f		f		f		f	f	f		f	f	t 68		f 2
36	F	f		f		f		f		t		t	f	f		f	f	t 1.5		t 2.4

- Următorul pas este procesarea datelor. Vom înlocui spațiile din numele capetelor de tabel cu "_" pentru consistență și claritate, vom înlocui "?" cu None și vom afișa pentru fiecare coloană câte valori nule conține.

```
df_data= df_data.na.replace('?', None)
```

```
df_data.select([count(when(isnan(col(c)) | col(c).isNull(),
c)).alias(c) for c in df_data.columns]).show()
```

age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_medication	sick	pregnant	thyroid_surgery	t13t1_treatment	query_hypothyroid	query_hyperthyroid	lithium	goitre	tumor	hypopituitary	psych	TSH_measured	TSH	T3_measured	T3	TT4	TT4U
0	307	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

• De asemenea, se observa ca pentru coloana TBG exista foarte multe valori nule, de aceea o vom elimina. În plus, vom elimina si coloanele "TBG_measured", "T3_measured", "TSH_measured", "TT4_measured", "T4U_measured", "FTI_measured", întrucât informația nu este relevantă dat fiind faptul ca avem incluse măsurătorile, nu este necesar și o variabilă booleană care să precizeze acest aspect. Pentru coloanele ce conțin informații despre cantitatea de hormoni rezultată la analize, ne vom asigura că valoare lipsă vor fi înlocuite cu media valorilor existente. Pentru a realiza acest proces se folosește *Imputer()*.

```
df_data =
df_data.drop(*["TBG_measured", "TBG", "T3_measured", "TSH_measured", "TT4_m
easured", "T4U_measured", "FTI_measured"])
cols = ["TSH", "T3", "TT4", "T4U", "FTI"]
imputer = Imputer(inputCols=cols, outputCols=[*cols], strategy="mean")
df_data = imputer.fit(df_data).transform(df_data)
```

• Pentru valorile de tip boolean, se observă că apar valorile 't' sau 'f', de aceea va trebui ca fiecare coloană ce conține informații de acest tip să fie transformată în coloană de tip *integer*, cu informații de 0 sau 1. Asemănător, se va proceda și pentru coloana "sex", transformând 'F' sau 'M' în 1 sau 2.

```
df_data = df_data.withColumn("sex", when(df_data.sex == "F",
2).when(df_data.sex == "M", 1))
df_data.show()
for col in df_data.columns:
    df_data = df_data.withColumn(col, when(df_data[col] == "f",
0).when(df_data[col] == "t", 1).otherwise(df_data[col]))
df_data.show()
```

age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_medication	sick	pregnant	thyroid_surgery	t13t1_treatment	query_hypothyroid	query_hyperthyroid	lithium	goitre	tumor	hypopituitary	psych	TSH	T3	TT4	TT4U	
29	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0.31	0.9706289	108.7003	0.97065574	113.6
29	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.6	1.9	128.0	0.97065574	113.6
41	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0.5.218403	1.9706289	108.7003	0.97065574	113.6
36	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5.218403	1.9706289	108.7003	0.97065574	113.6
32	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5.218403	1.9706289	108.7003	0.97065574	113.6
60	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5.218403	1.9706289	108.7003	0.97065574	113.6
77	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5.218403	1.9706289	108.7003	0.97065574	113.6
28	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.7	2.6	116.0	0.97065574	113.6
28	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.2	1.8	76.0	0.97065574	113.6
28	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.9	1.7	83.0	0.97065574	113.6
54	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.9	2.3	133.0	0.97065574	113.6
42	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	1.8	105.0	0.97065574	113.6
51	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.51.9706289	108.7003	0.97065574	113.6	
51	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.7	2.4	116.0	0.97065574	113.6
37	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0.5.218403	2.9	108.7003	0.97065574	113.6
16	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2.6	1.9706289	108.7003	0.97065574	113.6
54	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	2.0	122.0	0.97065574	113.6
43	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5.218403	2.1	116.0	0.97065574	113.6
63	2	0	1	0	0	1	0	0	0	0	0	0	0	0	0	68.0	1.9706289	48.0	1.02	1.02
36	2	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1.5	2.4	98.0	1.06	1.06

only showing top 20 rows

2.2. Folosirea unei funcții definite de utilizator

- Conform informațiilor despre dataset, se evidentiaza urmatoarele posibile *diagnostice*:

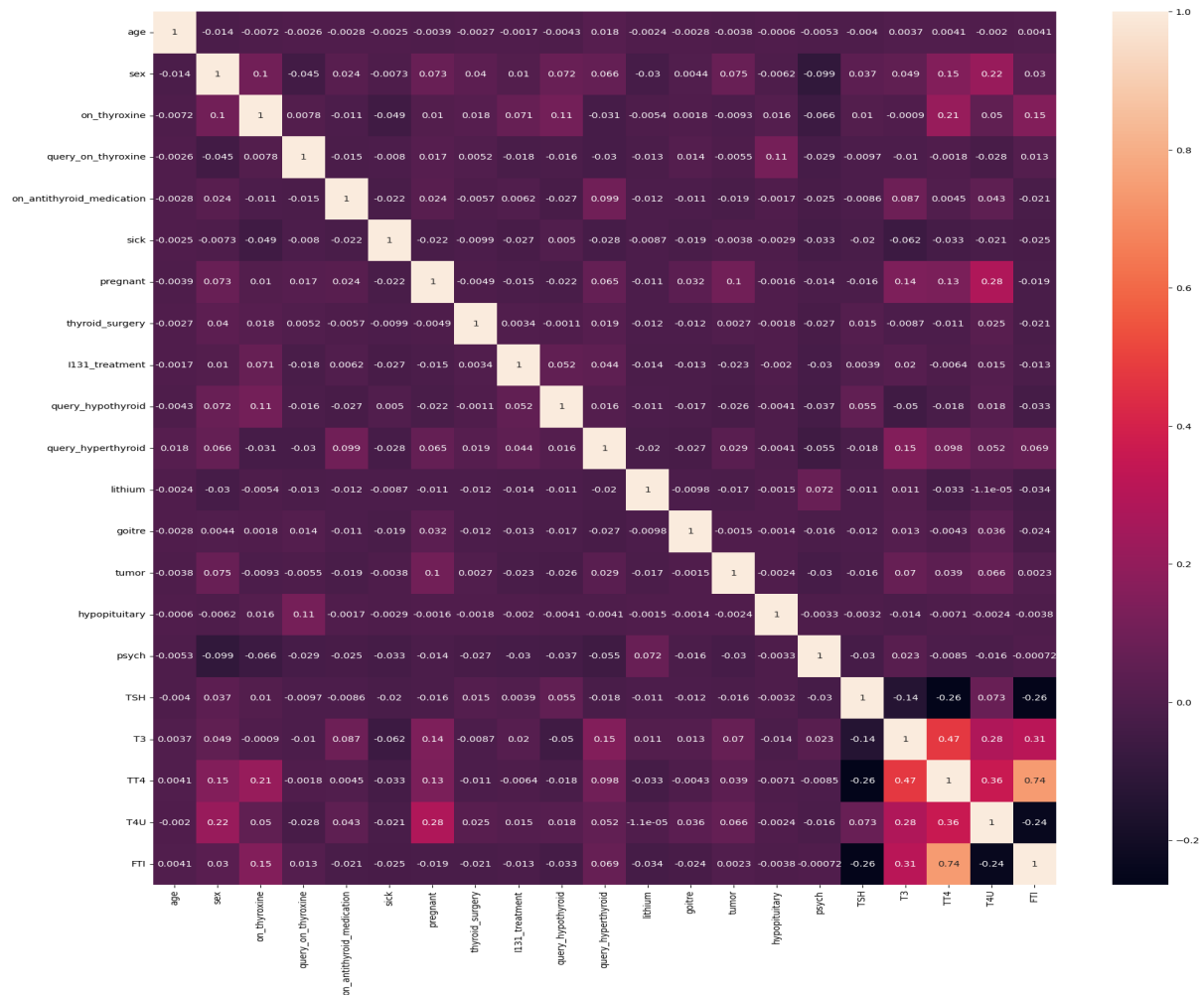
- Se va procesa coloana tag_id care este de forma S[840803047], extragându-se doar litera corespunzătoare și adăugând tag-ul corespunzător diagnosticului. Această etapă este esențială în procesul de clasificare sau regresie întrucât este corespunzătoare clasei din care fac parte fiecare înregistrare. Pentru a se realiza acest proces, am definit o funcție UFD.

[illegible]

Pentru a analiza date se evidențiază mai întâi matricea de corelație, urmând să se execute mai multe query-uri folosind SparkSql.

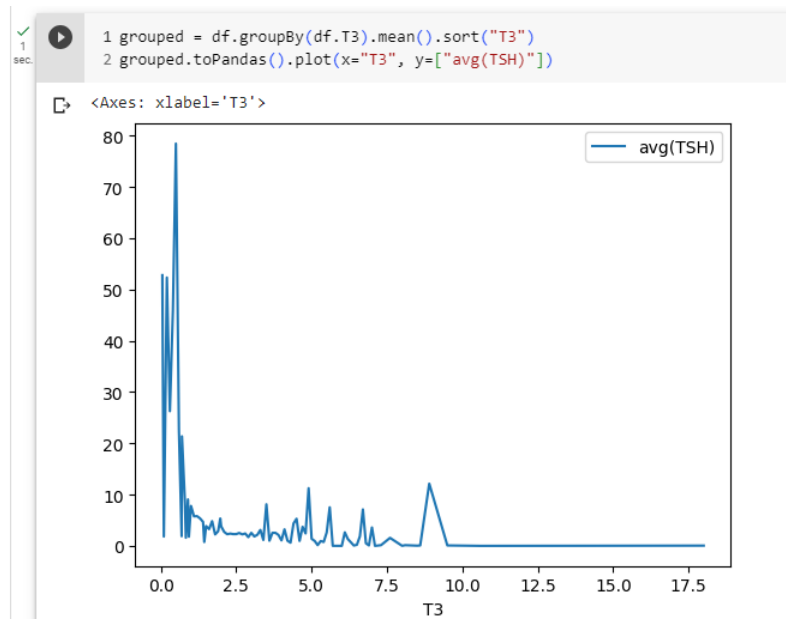
- Se va reprezenta matricei de corelație. Pentru aceasta, vom elimina coloana corespunzătoare tag-ului.

```
df = df_data.drop("tag_id")
plt.figure(figsize=(20,20))
sns.heatmap(df.toPandas().corr(),annot=True)
```



- Se reprezintă influența T3 si TSH. Se observă că există o relație inversă între TSH și T3. Acest rezultat este confirmat științific deoarece atunci când nivelul T3 este scăzut în sânge, glanda hipofiza eliberează mai mult TSH pentru a stimula glanda tiroidă să producă și să elibereze mai mult T3.

```
grouped = df.groupby(df.T3).mean().sort("T3")
grouped.toPandas().plot(x="T3", y=["avg (TSH)"])
```



- Se afișează vârsta medie a femeilor care au suferit o intervenție chirurgicală și au valoarea TSH-ului între 0.5 și 5.2 (limitele normale).

```
df_data.createOrReplaceTempView("th_disease")
queryDF = spark.sql("""
    SELECT avg(age)
    FROM th_disease
    WHERE sex = 2 and thyroid_surgery = 1 and ( TSH BETWEEN 0.5 and
5.2) """)
queryDF.show()
```

```
1 df_data.createOrReplaceTempView("th_disease")
2 queryDF = spark.sql("""
3     SELECT avg(age)
4     FROM th_disease
5     WHERE sex = 2 and thyroid_surgery = 1 and ( TSH BETWEEN 0.5 and 5.2)
6 """)
7 queryDF.show()
```

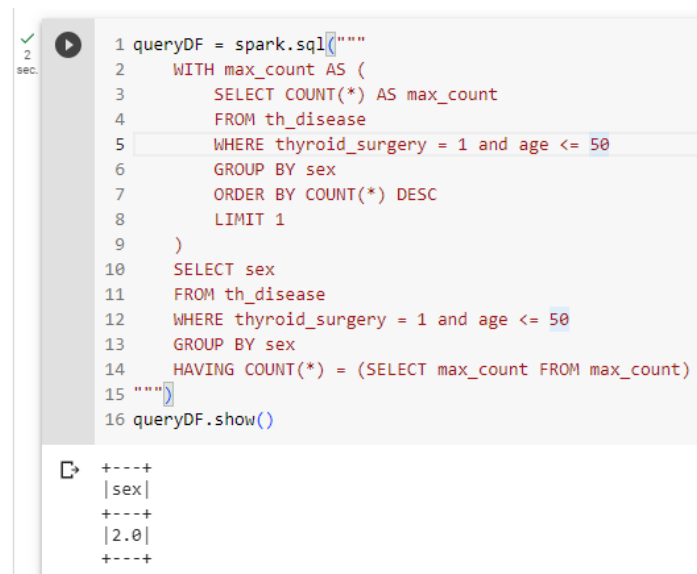
avg(age)
53.301886792452834

- Se afișează care dintre cele două categorii, bărbați sau femei, au predispoziție să sufere o intervenție chirurgicală asupra glandei tiroide, înainte de vârsta de 50 de ani.

```
queryDF = spark.sql("""
    WITH max_count AS (
        SELECT COUNT(*) AS max_count
        FROM th_disease
        WHERE thyroid_surgery = 1 and age <= 50
        GROUP BY sex
        ORDER BY COUNT(*) DESC
        LIMIT 1
    )
    )
```



```
SELECT sex
FROM th_disease
WHERE thyroid_surgery = 1 and age <= 50
GROUP BY sex
HAVING COUNT(*) = (SELECT max_count FROM max_count)
""")
queryDF.show()
```



The screenshot shows a Jupyter Notebook interface. On the left, there is a status bar indicating '2 sec.' and a play button. The main area contains a code cell with the following Spark SQL query:

```
1 queryDF = spark.sql("""
2     WITH max_count AS (
3         SELECT COUNT(*) AS max_count
4         FROM th_disease
5         WHERE thyroid_surgery = 1 and age <= 50
6         GROUP BY sex
7         ORDER BY COUNT(*) DESC
8         LIMIT 1
9     )
10    SELECT sex
11    FROM th_disease
12    WHERE thyroid_surgery = 1 and age <= 50
13    GROUP BY sex
14    HAVING COUNT(*) = (SELECT max_count FROM max_count)
15    """)
16 queryDF.show()
```

Below the code cell, the output is displayed as a table:

```
+---+
|sex|
+---+
|2.0|
+---+
```

3. Utilizarea SparkMLlib pentru clasificări și regresii

3.1. Standardizarea datelor

Înainte de a aplica modele de clasificare și regresie, datele trebuie să fie standardizate. Se folosește *VectorAssembler* pentru a combina coloanele cu caracteristicile într-un singur vector coloană, iar apoi *StandardScaler* pentru scalarea acestor caracteristici.

```
columns = df_data.columns
columns.remove('tag_id')
va = VectorAssembler(inputCols = columns, outputCol='features')
df_data = va.transform(df_data)
sc = StandardScaler(inputCol='features', outputCol='scaledFeatures',
withStd=True, withMean=False)
sc_model = sc.fit(df_data)
df_data = sc_model.transform(df_data)
```

Pentru a determina diagnosticul pe baza caracteristicilor amintite mai sus, se poate aplica atât regresie, cât și clasificare. Sunt 8 clase posibile, fiecare corespunzătoare unei anumite categorii de afecțiune, cu excepția ultimei care caracterizează rezultatele discordante.

Se folosește setul de date avut la dispoziție care se împarte astfel: 70% constituie datele de antrenare, iar 30% datele de test.

```
train, test = df_data.select('scaledFeatures',  
'tag_id').randomSplit([0.7, 0.3], seed=23)
```

Se creează un evaluator de tip *MulticlassClassificationEvaluator*, pentru a evalua modele ce vor fi folosite pentru regresie și clasificare.

```
evaluator =  
MulticlassClassificationEvaluator(predictionCol='prediction',  
labelCol='tag_id', metricName='accuracy')
```

3.2. Regresia logică

Este folosită Regresia Logistică multinomială pentru a prezice apartenența la o categorie pe baza mai multor caracteristici independente.

```
lr = LogisticRegression(featuresCol='scaledFeatures', labelCol='tag_id',  
family='multinomial')  
lr_model = lr.fit(train)  
pred = lr_model.evaluate(test).predictions  
pred.show()
```

```
[152] 1 lr = LogisticRegression(featuresCol='scaledFeatures', labelCol='tag_id', family='multinomial')  
      2 lr_model = lr.fit(train)
```

```
1  
sec. 1 pred = lr_model.evaluate(test).predictions  
      2 pred.show()
```

scaledFeatures	tag_id	rawPrediction	probability	prediction
(21,[0,1,2,3,5,16,...])	8	[-5.0221326999652...]	[4.49439056448172...]	8.0
(21,[0,1,2,3,12,1...])	8	[-4.6248189235452...]	[3.45816622323337...]	8.0
(21,[0,1,2,3,16,1...])	5	[-4.9438993901723...]	[4.01528645274340...]	5.0
(21,[0,1,2,3,16,1...])	5	[-4.9336045303585...]	[1.37538212671655...]	5.0
(21,[0,1,2,3,16,1...])	8	[-4.9433167048121...]	[3.60962981745866...]	8.0
(21,[0,1,2,3,16,1...])	8	[-4.8986734757466...]	[5.07183177500929...]	8.0
(21,[0,1,2,3,16,1...])	5	[-4.9853037859397...]	[1.37627523776565...]	5.0
(21,[0,1,2,3,16,1...])	5	[-4.9503874160666...]	[2.43853631172790...]	5.0
(21,[0,1,2,4,16,1...])	8	[-5.1032744315832...]	[1.33372380048854...]	8.0
(21,[0,1,2,4,16,1...])	8	[-5.1285390498595...]	[2.83768766842200...]	8.0
(21,[0,1,2,4,16,1...])	6	[-4.9726780035187...]	[1.87002709291198...]	5.0

După evaluarea modelului, se obține:

```
3 evaluator.evaluate(pred)
```

0.855056589996349

3.3. Random Forest

Algoritmul Random Forest este o metodă de clasificare care utilizează arbori de decizie. În cadrul Random Forest, se generează mai mulți arbori de decizie, fiecare pornind

de la o submulțime aleatorie a setului de date inițial. Fiecare arbore generat este aplicat pe instanța care trebuie clasificată, iar rezultatul este clasa atribuită de fiecare arbore. Clasa finală atribuită este determinată pe baza clasei majoritare obținute din toți arborii de decizie.

Am creat un model folosind metoda **RandomForestClassifier()**, importată din *pyspark.ml.classification*, pe care l-am antrenat pe datele de antrenare și l-am testat pe cele de test. La final am evaluat predicția obținută.

```
rf = RandomForestClassifier(featuresCol='scaledFeatures',
labelCol='tag_id', predictionCol='prediction', numTrees=10, seed=42)
rf_model = rf.fit(train)
pred = rf_model.evaluate(test).predictions
pred.show()
evaluator.evaluate(pred)
```

```
| (21,[0,1,2,5,16,1...| 1|[0.0,1.0900054030...|[0.0,0.1090005403...| 8.0|
| (21,[0,1,2,5,16,1...| 8|[0.0,0.0493698934...|[0.0,0.0049369893...| 8.0|
| (21,[0,1,2,5,16,1...| 8|[0.0,0.0507811727...|[0.0,0.0050781172...| 8.0|
| (21,[0,1,2,5,16,1...| 5|[0.0,0.0,3.223240...|[0.0,0.0,0.322324...| 5.0|
| (21,[0,1,2,5,16,1...| 2|[0.0,0.0056588520...|[0.0,5.6588520614...| 2.0|
| (21,[0,1,2,5,16,1...| 8|[0.0,0.0507811727...|[0.0,0.0050781172...| 8.0|
| (21,[0,1,2,6,10,1...| 8|[0.0,0.0451223206...|[0.0,0.0045122320...| 8.0|
| (21,[0,1,2,6,10,1...| 3|[0.0,0.2117889873...|[0.0,0.0211788987...| 8.0|
+-----+-----+-----+-----+
only showing top 20 rows
0.899963490324936
```

3.4. Optimizarea hiperparametrilor pentru Random Forest

Pentru a realiza optimizarea hiperparametrilor, am creat inițial un clasificator folosind **RandomForestClassifier()**, apoi am definit o grilă de posibili parametri pentru *numTree*. Am folosit **CrossValidator()**, validarea încrucișată, pentru a găsi cel mai bun model, apoi l-am aplicat pe datele de test și am evaluat acuratețea predicției.

```
rf = RandomForestClassifier(featuresCol='scaledFeatures',
labelCol='tag_id', predictionCol='prediction')
paramGrid = ParamGridBuilder().addGrid(rf.numTrees, [5,10]).build()
cross_validator = CrossValidator(estimator=rf,
                                estimatorParamMaps=paramGrid,
                                evaluator=evaluator,
                                numFolds=2,
                                seed=42 )
# antrenam croos_validator pe datele de antrenare
#apoi alegem cel mai bun model
cv_model = cross_validator.fit(train)
rf_best_model = cv_model.bestModel
pred = rf_best_model.transform(test)
pred.show()
evaluator.evaluate(pred)
```

scaledFeatures	tag_id	rawPrediction	probability	prediction
(21,[0,1,2,3,5,16...]	8	[0.0,0.0459975978...	[0.0,0.0091995195...	8.0
(21,[0,1,2,3,12,1...]	8	[0.0,0.0391756654...	[0.0,0.0078351330...	8.0
(21,[0,1,2,3,16,1...]	5	[0.0,0.0181708809...	[0.0,0.0036341761...	2.0
(21,[0,1,2,3,16,1...]	5	[0.0,0.0181708809...	[0.0,0.0036341761...	2.0
(21,[0,1,2,3,16,1...]	8	[0.0,0.0459975978...	[0.0,0.0091995195...	8.0
(21,[0,1,2,3,16,1...]	8	[0.0,0.0391756654...	[0.0,0.0078351330...	8.0
(21,[0,1,2,3,16,1...]	5	[0.0,0.2681272882...	[0.0,0.0536254576...	5.0
(21,[0,1,2,3,16,1...]	5	[0.0,0.0373931343...	[0.0,0.0074786268...	8.0
(21,[0,1,2,4,16,1...]	8	[0.0,0.0373931343...	[0.0,0.0074786268...	8.0
(21,[0,1,2,4,16,1...]	8	[0.0,0.0391756654...	[0.0,0.0078351330...	8.0
(21,[0,1,2,4,16,1...]	6	[0.0,0.1046983952...	[0.0,0.0209396790...	8.0
(21,[0,1,2,5,9,16...]	5	[0.0,0.2681272882...	[0.0,0.0536254576...	5.0
(21,[0,1,2,5,13,1...]	1	[0.0,1.2576342049...	[0.0,0.2515268409...	8.0
(21,[0,1,2,5,16,1...]	8	[0.0,0.0391756654...	[0.0,0.0078351330...	8.0
(21,[0,1,2,5,16,1...]	8	[0.0,0.0459975978...	[0.0,0.0091995195...	8.0
(21,[0,1,2,5,16,1...]	5	[0.0,0.0113489486...	[0.0,0.0022697897...	2.0
(21,[0,1,2,5,16,1...]	2	[0.0,0.0095664174...	[0.0,0.0019132834...	2.0
(21,[0,1,2,5,16,1...]	8	[0.0,0.0459975978...	[0.0,0.0091995195...	8.0
(21,[0,1,2,6,10,1...]	8	[0.0,0.0960171049...	[0.0,0.0192034209...	8.0
(21,[0,1,2,6,10,1...]	3	[0.0,0.9866303841...	[0.0,0.1973260768...	8.0

only showing top 20 rows

0.841913106973348

4. Utilizarea unui Data Pipeline

Un Data Pipeline este compus dintr-o serie de transformări și acțiuni aplicate asupra datelor, transformările fiind executate în mod *lazy* (leneș) doar când este necesar să se obțină rezultatele, acțiunile ocupându-se de această declanșare. Acesta facilitează crearea și gestionarea fluxurilor de lucru complexe de prelucrare a datelor în Spark.

Am adăugat în pipeline vectorizerul, scalerul și modelul Random Forest, apoi am antrenat și testat modelul, ulterior evaluând predicția.

```
# adaugarea in pipeline a va, scalerului si a modelului rf
pipeline = Pipeline(stages=[va, sc, rf])
df = df.drop('features', 'scaledFeatures')
# impartirea setului de date in antrenare si test
train, test = df.randomSplit([0.7, 0.3], seed=23)

fitted_model = pipeline.fit(train)
pred = fitted_model.transform(test)
pred.select('tag_id', 'prediction').show()
evaluator.evaluate(pred)
```

```
|      8|      8.0|
+-----+-----+
only showing top 20 rows

0.8919313618108798
```

5. *Aplicarea unei metode de Deep Learning, folosind Tensorflow*

Am importat librariile de care aveam nevoie, după care am pus într-un vector separat coloana corespunzătoare claselor. Am împărțit datele în procent de 70% pentru cele de antrenare și 30% pentru cele de test folosind metoda *train_test_split()*. Ulterior am transformat datele în date categorice.

```
X = df_data.drop('tag_id').toPandas()
y = df_data.select('tag_id').toPandas()

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, stratify=y, random_state=23)

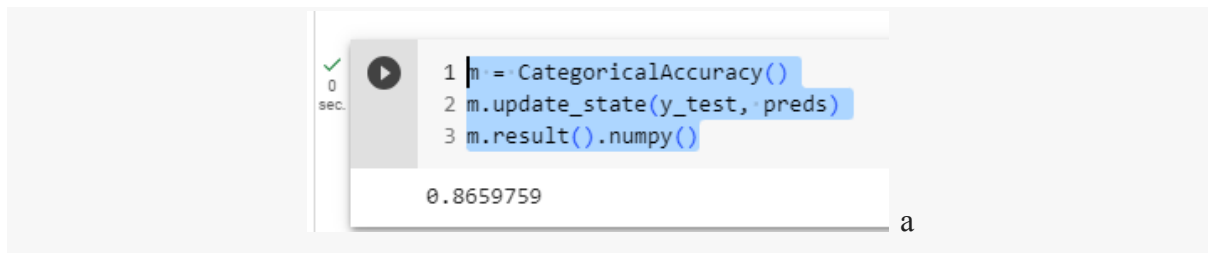
y_train = to_categorical(y_train['tag_id'].to_numpy(), num_classes=9)
y_test = to_categorical(y_test['tag_id'].to_numpy(), num_classes=9)
```

Am utilizat un model secvențial. Mai întâi este adăugat un strat de intrare, *InputLayer*, care specifică forma datelor de intrare în rețeaua neurală, în cazul de față avem 21 de caracteristici. A urmat adăugarea a patru layere cu funcția de activare *Relu*, alegând număr variabil de neuroni. La final am adăugat un layer de ieșire cu activare *softmax* pentru a obține probabilitățile de apartenență clase. În urma mai multor teste, pentru optimizare am ales SGD, iar numărul de epoci este de 100. La final am evaluat predicția obținută.

```
model = Sequential()
model.add(InputLayer(input_shape=(21,)))
model.add(Dense(100, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(28, activation='relu'))
model.add(Dense(14, activation='relu'))
model.add(Dense(9, activation="softmax"))

model.summary()
model.compile(optimizer= Adam(0.001), loss='categorical_crossentropy',
metrics=['categorical_accuracy'])

model.fit(X_train, y_train, batch_size=64, epochs=100)
preds = model.predict(X_test)
m = CategoricalAccuracy()
m.update_state(y_test, preds)
m.result().numpy()
```



6. Utilizarea SparkStreaming

Mai întâi am salvat datele de test pe care le vom citi prin intermediul SparkStreaming într-un .csv, după care am creat un obiect SparkStreaming și am apelat funcția `p_stream` ce folosește modelul antrenat la exercitiul anterior de DataPipeline.

```
test.toPandas().to_csv("/content/drive/MyDrive/MASTER_FMI/Colab_Notebooks/Data/thyroid.csv", index=False, header=False)
ssc = StreamingContext(spark.sparkContext, 3)
lines =
ssc.textFileStream('/content/drive/MyDrive/MASTER_FMI/Colab_Notebooks/Data/')
rows = lines.map(lambda line: [float(x) for x in line.split(",")])
rows.foreachRDD(p_stream)
```

Iar funcția `p_stream` creează un DataFrame bazându-se pe recordurile citite din fișier.

```
def f(x):
    d = {}
    for i in range(len(x)):
        d[columns[i]] = x[i]
    return d
def p_stream(time, record):
    print(f"==== {time} =====")
    try:
        df = record.map(lambda x: Row(**f(x))).toDF()
        pred = fitted_model.transform(df).select(["tag_id",
"prediction"])
        pred.show(pred.count(), False)
#pred.write.mode("append").csv('/content/drive/MyDrive/MASTER_FMI/Colab_Notebooks/Data/thyroid_prediction.csv')
    except Exception as e:
        print(e)
```

La final am pornit contextul.

```
ssc.start()

ssc.awaitTermination()
```