تمرین دوم درس Big Data

عرفان اشتری 401123916

1)

ابتدا برای شروع کار میبایست دیتای موجود از کدهای اسمبلی که در اختیارمان قرار گرفته را از فایل ASM_Features.rar بخوانیم و دیتا را آماده کنیم برای مراحل بعد. برای خواندن ابتدا کتابخانه زیر را نصب میکنیم

```
pip install rarfile
```

پس از آن کتابخانه های مورد نیاز را فراخوانی میکنیم

```
import pandas as pd
import matplotlib.pyplot as plt
from rarfile import RarFile, Path
from PIL import Image
import numpy as np
```

با استفاده از کد زیر ابتدا توسط کتابخانه نصب شده فایل فشرده را باز میکنیم و آدرس فایلهای csv درون آن را میخوانیم (به جز آدرس آخر که مربوط به همان فایل فشرده است). پس از آن از این آدرسها که در لیست `filelist` قرار دارند استفاده کرده و با کتابخانه `pandas` فایلهای `csv` را یک به یک خوانده و محتویات دیتافریم هایشان را در لیست `data` ذخیره میکنیم. همزمان از فایل `trainLabels.csv` نیز استفاده میکنیم و کلاسهای مربوط به هرفایل اسمبلی موجود در فایل های csv را استخراج میکنیم و در لیست `class_df` میریزیم. پس از خواندن تمام فایلهای csv در نهایت دیتافریم های موجود در لیست df را به هم `concat` کرده و یک دیتافریم واحد از تمام فایل ها ایجاد میکنیم و همینطور لیست `class_df` را نیز به دیتافریم تبدیل میکنیم.

```
#reading csv files from the zip file and concatenating them as a dat
aframe also creating related class numbers dataframe
DF_labels= pd.read_csv("trainLabels.csv")
data=[]
class_df = []
with RarFile("ASM_Features.rar") as zipfiles:

    #the last entry is the zipfile name
    #we'll skip it
    filelist = zipfiles.namelist()[:-1]

    for file_name in filelist:
        filelist = zipfiles.namelist()[:-1]

        data.append(pd.read_csv(zipfiles.open(file_name)))
        class_df.append( DF_labels[DF_labels["Id"] == file_name[13:-
4] ]["Class"].values[0] )
```

```
df = pd.concat(data)
class_df = pd.DataFrame(class_df,columns=["Class",])
```

دیتافریم class_df یک ستون دارد و 10868 سطر که تعداد فایلهای اسمبلی است و دیتا فریم df نیز 3 ستون و 507499 سطر، در این دیتافریم opcode های موجود در هر کد اسمبلی و تعداد تکرارشان به ترتیب آورده شده است.

در زیر دیتا فریم class_df را مشاهده میکنید

	Class
0	9
1	2
2	9
3	1
4	8
...	...
10863	4
10864	4
10865	4
10866	4
10867	4

10868 rows × 1 columns

دیتافریم df نیز به صورت زیر است.

	Unnamed: 0	OPcodes	count
0	0	db	1366877
1	1	mov	5295
2	2	push	3208
3	3	call	1533

	Unnamed: 0	OPcodes	count
4	4	jmp	730
...	...	...	...
64	64	movzx	1
65	65	fst	1
66	66	mul	1
67	67	jge	1
68	68	ja	1

507499 rows × 3 columns

حال باید دیتافریمی بسازیم که در آن این opcode های یکتا ستون های آن باشند و در هر سطر تعداد تکرارشان برای هر فایل اسمبلی قرار بگیرد. بدین منظور ابتدا opcode های یکتا را با استفاده از کد زیر استخراج میکنیم و در یک لیست ذخیره میکنیم.

```
#extracting all the unique OPcodes from all the files
col = df['OPcodes'].unique()
```

که به صورت زیر است

```
['db',
 'mov',
 'push',

.

.

.
 'daa',
 'vpsrlw',
 'stosd']
```

حال که نام ستون های مورد نیازمان را داریم بار دیگر فایل های csv را تک به تک از فایل فشرده میخوانیم و ضمن حذف ستون index ها به نام 'Unnamed: 0' آن را ترنسپوز میکنیم و در ادامه یک دیتافریم با مقادیر صفر به نام DATA_FRAME میسازیم که ستون های آن ها شامل تمام opcde های تمام فایلهاست که در لیست col ذخیره کرده بودیم. مقادیر به دست آمده از هر فایل csv را در سطر های این دیتافریم جای گذاری میکنیم.

```
# creating a data frame which it's columns are all the unique OPcode
s and each row contains the count of OPcodes for
```

```
# an specific file

data2=[]
with RarFile("ASM_Features.rar") as zipfiles:

    #the last entry is the zipfile name
    #we'll skip it
    filelist = zipfiles.namelist()[:-1]
    for count,file_name in enumerate(filelist):

        temp = pd.read_csv(zipfiles.open(file_name))
        tempt = temp.drop('Unnamed: 0', axis=1).transpose()
        tempt.columns= tempt[0:1].values[0]
        tempt = tempt.drop(["OPcodes"])
        DATA_FRAME= pd.DataFrame(0, index=np.arange(1), columns=col)
        DATA_FRAME[tempt.columns]=tempt.values
        data2.append(DATA_FRAME)
        print(count,end="\r")

print("out")
DATA_FRAME = pd.concat(data2)
print("out2")
DATA_FRAME.head()
```

و در نهایت نتیجه به این صورت است.

	db	mov	push	call	...	daa	vpsrlw	stosd
0	1366877	5295	3208	1533	...	0	0	0
0	63354	2190	569	333	...	0	0	0
0	1226	31	442	125	...	0	0	0
0	105	89	81	53	...	0	0	0
0	5752	2535	402	194	...	0	0	0

10868 rows × 719 columns

پس از آن با استفاده از دیتافریم `class_df` یک ستون class نیز اضافه میکنیم و ایندکس ها را نیز مرتب میکنیم.

```
DATA_FRAME.index = np.arange(DATA_FRAME.shape[0])

# adding labels as last column of the main dataframe
DF = pd.concat([DF_temp, class_df], axis=1).drop('Unnamed: 0',
axis=1)
```

```
filepath = "DATA_FRAME_Final.csv"
DF.to_csv(filepath)
```

نتیجه نهاییمان به صورت زیر میشود که آن را به عنوان یک فایل csv ذخیره کردیم.

	db	mov	push	...	daa	vpsrlw	stosd	Class
0	1366877	5295	3208	...	0	0	0	9
1	63354	2190	569	...	0	0	0	2
2	1226	31	442	...	0	0	0	9
...	...	...	...	...	...	...	...	...
10866	75982	512	879	...	0	0	0	4
10867	67654	474	826	...	0	0	0	4

10868 rows × 720 columns

حالا دیتای مناسب آموزش را داریم و می توانیم به سراغ اسپارک و استفاده از آن برای انجام عملیات کلاس بندی برویم. پس در ابتدا کتابخانه های مورد نیازمان را فرامیخوانیم و اسپارک را راه اندازی کرده و مطابق کدهای زیر دیتافریممان را از فایل csv خوانده و به یک دیتافریم اسپارک تبدیل میکنیم.

```
#initializing and starting a pyspark session and transform pandas
dataframe to spark dataframe

from pyspark.sql import SparkSession #Import the pyspark
from pyspark.conf import SparkConf #Import the SparkConf
from pyspark.context import SparkContext #Import the SparkContext
import time
from pyspark.sql import SparkSession

DF = pd.read_csv("DATA_FRAME_Final.csv")
DF = DF.drop('Unnamed: 0', axis=1)
conf = SparkConf()
conf.setMaster("local").setAppName("HW2_1")
sc = SparkContext.getOrCreate(conf=conf)

#Create PySpark SparkSession
spark= SparkSession.builder.master("local").getOrCreate()
#Create PySpark DataFrame from Pandas
sparkDF=spark.createDataFrame(DF)
```

از روی دیتافریم pandas میتوانیم نشان دهیم که توزیع داده ها در کلاس ها به چه صورت است.

```
#count of each class in the dataframe
for i in range(9):
```

```
    print("num Total class ",i+1," : ",sum(sparkDF_pd["Class"]==i+1)
)
```

که نتیجه زیر برای 9 کلاسی که داریم بدست می آید

```
num Total class  1  :  1541
num Total class  2  :  2478
num Total class  3  :  2942
num Total class  4  :  475
num Total class  5  :  42
num Total class  6  :  751
num Total class  7  :  398
num Total class  8  :  1228
num Total class  9  :  1013
```

که مطابق آن مشاهده میکنید که در هر کلاس چند داده وجود دارد و کلاس 5 با 42 کمترین و کلاس 3 با 2942 بیشترین تعداد داده را دارند.

حال که داده ها آمده شده و در حافظه هستند داده را به سه قسمت train, test, validation تقسیم میکنیم.

```
#spliting the dataframe to TRAIN,TEST,VALID
TRAIN,TEST,VALID = sparkDF.randomSplit([0.85,0.1,0.05])
```

از کدی که در مرحله قبل استفاده کردیم مجدد استفاده میکنیم و تعداد داده های هر کلاس برای train, test, validation را به دست می آوریم

داده های train

```
num TRAIN class  1  :  1327
num TRAIN class  2  :  2126
num TRAIN class  3  :  2491
num TRAIN class  4  :  414
num TRAIN class  5  :  39
num TRAIN class  6  :  645
num TRAIN class  7  :  330
num TRAIN class  8  :  1032
num TRAIN class  9  :  858
```

داده های test

```
num TEST class  1  :  151
num TEST class  2  :  230
num TEST class  3  :  293
num TEST class  4  :  45
num TEST class  5  :  3
num TEST class  6  :  72
num TEST class  7  :  39
num TEST class  8  :  119
num TEST class  9  :  113
```

داده های validation

```
num VALID class  1  :  63
num VALID class  2  :  122
num VALID class  3  :  158
num VALID class  4  :  16
num VALID class  5  :  0
num VALID class  6  :  34
num VALID class  7  :  29
num VALID class  8  :  77
num VALID class  9  :  42
```

در مجموع تعداد داده ها به این صورت است که 9262 داده آموزش 1065 داده تست و 541 داده ارزیابی وجود دارد که در ادامه از آن ها استفاده خواهیم کرد.

الف)

برای استفاده از داده هایی که در مرحله قبل به دست آوردیم ابتدا آن ها را shuffle میکنیم تا مشکلی در زمان آموزش پیش نیاید.

```
from pyspark.sql.functions import rand
# TRAIN,TEST,VALID shuffling
TRAIN = TRAIN.orderBy(rand())
TEST = TEST.orderBy(rand())
VALID = VALID.orderBy(rand())
```

حال باید دیتافریم ها را به صورتی دربیاوریم که بتوانیم آن را به عنوان ورودی به مدل خود بدهیم. برای این کار باید یک ستون به عنوان features بسازیم که شامل تمام ویژگی ها یا همان ستون های دیتافرم ها هست به جز ستون مربوط به لیبل ها. برای این کار از کد زیر استفاده میکنیم.

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

input_cols=VALID_pd.columns[:-1]
output_cols="features"

#adding a features column which contains all the columns as a single
list, except labels column, to all dataframes
assembler = VectorAssembler(inputCols=input_cols.to_list(), outputCo
l=output_cols)
TRAIN = assembler.transform(TRAIN)
TEST = assembler.transform(TEST)
VALID = assembler.transform(VALID)
```

برای مثال دیتافریم داده های ارزیابی به صورت زیر در می آید.

	db	mov	push	...	daa	vpsrlw	stosd	Class	features
0	986	158	...	0	0	0	3	(986.0, 158.0, 367.0, 172.0, 1.0, 6.0, 1.0, 51...
1	331	66	...	0	0	0	6	(331.0, 66.0, 152.0, 64.0, 19.0, 14.0, 41.0, 5...
2	3335549	1138	...	0	0	0	2	(3335549.0, 1138.0, 451.0, 209.0, 83.0, 64.0, ...
...	...	...	...	...	...	...	...	...
539	16246	2454	...	0	0	0	8	(16246.0, 2454.0, 428.0, 136.0, 68.0, 2.0, 209...
540	4324	2532	...	0	0	0	8	(4324.0, 2532.0, 393.0, 204.0, 71.0, 3.0, 178....

541 rows × 721 columns

حال دیگر داده ها به صورت کامل آماده ورودی داده شدن به شبکه هستند.

حالا باید مدل های random forest خود را براساس خواست سوال آماده کنیم، برای این کار از `RandomForestClassifier` موجود در اسپارک استفاده میکنیم و پارامتر `maxDepth` را برابر 10 قرار میدهیم و ستون ویژگی ها و لیبل ها را نیز مشخص میکنیم و مطابق خواست سوال طی چند آزمایش مقدار `numTrees` را بین 10-20-30-40-50 تغییر میدهیم. تکه کد زیر به عنوان نمونه آمده که میتواند برای هر تعداد درخت تکرار شود و یا حتی در یک حلقه for این تعداد درختها را تغییر دهیم، که البته ما این کار را مجزا برای تعداد درخت ها انجام دادیم. همینطور لیستی به عنوان TIME هم وجود دارد که مدت زمان اجرای آموزش با هر تعداد درخت در آن قرار میگیرد.

```
# from pyspark.ml.classification import RandomForestClassifier

rf = RandomForestClassifier(numTrees=10, maxDepth=10,featuresCol = "
features", labelCol = "Class")
s1 = time.time()
rfModel = rf.fit(TRAIN)
predictions = rfModel.transform(TEST)
s2 = time.time()
TIME.append(s2-s1)
```

همانطور که مشاهده میکنید پس از آموزش، مقدار پیشبینی شده برای داده های تست در دیتافریم اسپارک `predictions` ذخیره میگردد. در اینجا ستون های لیبل و پیشبینی را برای ده سطر ابتدایی آموزش با تعداد ده درخت را برای نمونه نشان میدهیم که به صورت زیر هستند.

```
predictions.select("Class", "prediction").show(10)
```

```
+-----+----------+
|Class|prediction|
+-----+----------+
|    6|       6.0|
|    1|       1.0|
|    4|       4.0|
|    2|       2.0|
|    9|       9.0|
|    3|       3.0|
|    3|       3.0|
|    1|       1.0|
|    9|       9.0|
|    2|       2.0|
+-----+----------+
only showing top 10 rows
```

در همین جدول قابل ملاحظه هست که به نظر می آید شبکه برای این تعداد درخت به خوبی آموزش دیده. برای بررسی دقیق تر میزان دقت یا accuracy را برای همه حالتها با استفاده از کد زیر بدست آورده و در یک لیست ذخیره میکنیم.

```
#from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(labelCol="Class", pred
ictionCol="prediction")
accuracy = evaluator.evaluate(predictions)
print("Accuracy = %s" % (accuracy))
print("Test Error = %s" % (1.0 - accuracy))
ACC.append(accuracy)
```

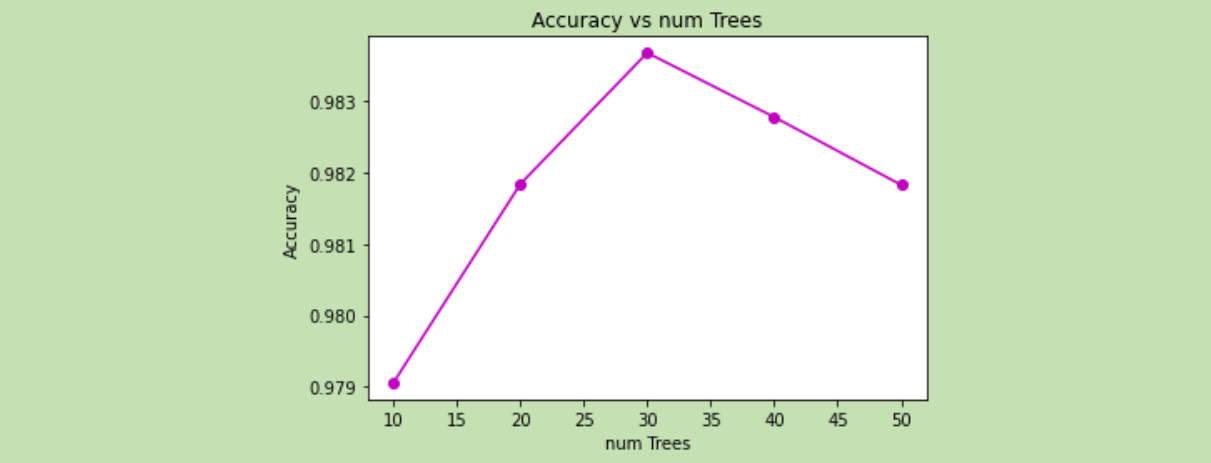
در نهایت با استفاده از لیست های ACC و TIME که بدست آوردیم و کدهای زیر نمودار مربوط به دقت و زمان اجرا را برای تعداد درخت های مختلف رسم میکنیم.

```
TREES = [10, 20, 30, 40, 50]
#plotting the duration of training for each num_trees in experiments
import matplotlib.pyplot as plt
plt.plot(TREES, TIME,"go-")
plt.title("Time vs num Trees")
plt.xlabel("num Trees")
plt.ylabel("Time")
plt.savefig("Time vs num_Trees.png")

# plotting the the accuracy of models for each num_trees in experime
nts
plt.plot(TREES, ACC,"mo-")
plt.title("Accuracy vs num Trees")
plt.xlabel("num Trees")
plt.ylabel("Accuracy")
plt.savefig("Accuracy vs num_Trees.png")
```

در نتیجه در نهایت نمودارهای زیر بدست می آیند.

نمودار زمان برحسب تعداد درخت

نمودار دقت برحسب تعداد درخت



همانطور که در نمودارها مشخص است نمودار زمان صعودی بوده و با افزایش تعداد درخت محاسبات افزایش یافته و زمان آن طولانی تر شده. اما برای تعداد درخت مشاهده میشود که نمودار در مقدار 30 به ماکزیمم خود در 0.9837 رسیده و بیشترین دقت را دارد و در نتیجه میتوان آن را به عنوان تعداد بهینه درخت در نظر گرفت.

ب)

در این بخش خواسته شده تا یک سری متریک ها را برای ارزیابی بهترین مدل که همان مدل با تعداد درخت 30 است را بدست آوریم. البته در نوت بوک این مقادیر برای همه حالات بدست آمده و موجود است. اما در این جا فقط نتایج بهترین مدل را نشان میدهیم. ابتدا به سراغ ماتریس سردرگمی می رویم و با استفاده از کد زیر آن را بدست می آوریم.

```
preds_and_labels = predictions.select(['prediction','Class']).withCo
lumn('Class',F.col('Class').cast(FloatType())).orderBy('prediction')
preds_and_labels = preds_and_labels.select(['prediction','Class'])
```

```
metrics = MulticlassMetrics(preds_and_labels.rdd.map(tuple))
print(metrics.confusionMatrix().toArray())
```

در این کد از کتابخانه ها و توابع اسپارک استفاده کردیم و با استفاده از ستون های پیشبینی و کلاس موجود در دیتافریم predictions ماتریس سردرگمی به صورت زیر برای تعداد درخت 30 بدست آمد.

```
[[151.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  1. 229.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0. 292.   1.   0.   0.   0.   0.   0.]
 [  0.   0.   0.  44.   0.   1.   0.   0.   0.]
 [  1.   0.   0.   0.   1.   0.   0.   1.   0.]
 [  2.   0.   0.   0.   0.  69.   0.   1.   0.]
 [  0.   0.   0.   1.   0.   0.  38.   0.   0.]
 [  5.   0.   0.   1.   0.   0.   0. 113.   0.]
 [  2.   0.   0.   0.   0.   0.   0.   0. 111.]]
```

هر سطر نشان دهنده کلاس واقعی داده ها و هر ستون نشان دهنده کلاس پیشبینی شده میباشد، بنابراین هرچه قطر اصلی ماتریس سنگین تر باشد نشان دهنده این است که عملکرد مدل بهتر بوده.همانطور که میبینند در برخی کلاس ها مثل کلاس یک (سطر اول) همه داده ها درست پیشبینی شده و روی قطر اصلی هستند. و به نظر می آید بدترین عملکرد برای کلاس 5 است که از 3 داده تست، فقط 1 داده روی قطر اصلی دارد و یک داده آن به اشتباه کلاس 1 تشخیص داده شده و داده دیگر آن کلاس 8 تشخیص داده شده. البته این اشتباه خیلی دور از انتظار هم نبود و بخاطر تعداد بسیار کم داده های این کلاس بود که در کل 42 داده از این کلاس داشتیم و به نظر می آید عملکرد آموزش روی این کلاس ضعیف تر بوده که برای حل این مسئله میتوان از روش هایی مثل over sampling استفاده کرد.

برای بدست آوردن سایر متریک ها از کد زیر استفاده میکنیم.

```
# Overall statistics
precision = []
recall = []
f1Score = []
tpr = []
fpr = []
for i in range(9):
    precision.append(metrics.precision(i+1.) )
    recall.append(metrics.recall(i+1.) )
    f1Score.append(metrics.fMeasure(i+1.) )
    tpr.append(metrics.truePositiveRate(i+1.) )
    fpr.append(metrics.falsePositiveRate(i+1.) )
print("Summary Stats")
print("\nPrecision = %s" % precision)
print("\nRecall = %s" % recall)
print("\nF1 Score = %s" % f1Score)
print("\nTrue Positive Rate = %s" % tpr)
print("\nFalse Positive Rate = %s" % fpr)
```

در نتیجه کدهای بالا مقدار متریک های مورد نظر برای هرکلاس بدست آمده و در لیست هایی نمایش داده میشود

```
Precision = [0.9320987654320988, 1.0, 1.0, 0.9361702127659575, 1.0,
0.9857142857142858, 1.0, 0.9826086956521739, 1.0]

Recall = [1.0, 0.9956521739130435, 0.9965870307167235, 0.97777777777
77777, 0.3333333333333333, 0.9583333333333334, 0.9743589743589743, 0
.9495798319327731, 0.9823008849557522]

F1 Score = [0.9648562300319489, 0.9978213507625272, 0.99829059829059
83, 0.9565217391304347, 0.5, 0.971830985915493, 0.9870129870129869,
0.9658119658119659, 0.9910714285714286]

True Positive Rate = [1.0, 0.9956521739130435, 0.9965870307167235, 0
.9777777777777777, 0.3333333333333333, 0.9583333333333334, 0.9743589
743589743, 0.9495798319327731, 0.9823008849557522]

False Positive Rate = [0.012035010940919038, 0.0, 0.0, 0.00294117647
05882353, 0.0, 0.0010070493454179255, 0.0, 0.0021141649048625794, 0.
0]
```

هر درایه از این لیستها مقدار آن متریک برای کلاس مربوط به آن است و در صورت میانگین گیری از این متریک ها عملکرد کلی شبکه بدست می آید.

```
# calculaing the means for all the above metrics
precision_m = np.mean(precision)
recall_m = np.mean(recall)
f1Score_m = np.mean(f1Score)
tpr_m = np.mean(tpr)
fpr_m = np.mean(fpr)

print("Summary mean Stats")
print("\nPrecision = %s" % precision_m)
print("\nRecall = %s" % recall_m)
print("\nF1 Score = %s" % f1Score_m)
print("\nTrue Positive Rate = %s" % tpr_m)
print("\nFalse Positive Rate = %s" % fpr_m)
```

نتایج کلی به صورت زیر است.

```
Precision = 0.9818435510627239

Recall = 0.9075470378135234

F1 Score = 0.9259130317252647

True Positive Rate = 0.9075470378135234
```

```
False Positive Rate = 0.0020108224068653086
```

همانطور که مشاهده میشود عملکرد کلی شبکه بسیار خوب بوده و متریکهایی که نشان دهنده دقت شبکه اند همگی مقادیری نزدیک به یک دارند و False Positive Rate که معیاری از میزان خطای شبکه است نیز مقداری نزدیک به صفر دارد.

در توضیح مربوط به این متریک ها و بدست آوردن آن ها درحقیقت از فرمول های زیر استفاده میگردد

$$\mathbf{Precision} = \frac{tp}{tp + fp}$$

$$\mathbf{Recall} = \frac{tp}{tp + fn}$$

$$F_1 = \frac{2}{\mathbf{recall}^{-1} + \mathbf{precision}^{-1}} = 2\frac{\mathbf{precision} \cdot \mathbf{recall}}{\mathbf{precision} + \mathbf{recall}} = \frac{\mathbf{2tp}}{\mathbf{2tp} + \mathbf{fp} + \mathbf{fn}}$$

$$\mathbf{FPR} = \frac{\mathbf{FP}}{\mathbf{N}} = \frac{\mathbf{FP}}{\mathbf{FP} + \mathbf{TN}} = 1 - \mathbf{TNR}$$

$$\mathbf{TPR} = \frac{\mathbf{TP}}{\mathbf{P}} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FN}} = 1 - \mathbf{FNR}$$

تمام این متریک ها با استفاده از ماتریس سردرگمی قابل دستیابی هستند فقط باید به این نکته دقت کرد که TP یعنی مقادیری که پیشبینی شده و درست بوده یعنی درایه روی قطر اصلی، FP نشان دهنده داده هایی است که برای کلاس های دیگر بوده اما به عنوان کلاس مورد نظر ما پیشبینی شده اند یعنی همان ستون کلاس به جز درایه قطر اصلی میباشد و FN نشان دهنده داده هایی است که در کلاس مورد نظر ما هستند اما به اشتباه در کلاسهای دیگر پیشبینی شده اند یعنی درایه های همان سطر به جز درایه قطر اصلی.

درواقع Precision نسبت تعداد پیشبینیهای درست از میان کل پیشبینی هایی با عنوان هرکلاس را نشان میدهد، Recall یا همان True Positive Rate نسبت تعداد پیشبینی درست به کل داده های کلاس است، F1 score ترکیبی از دو روش قبل است و در نهایت False Positive Rate نشان دهنده نسبت پیشبینی های غلط یک کلاس به کل داده های دیگر کلاس هاست.

2)

در این سوال همانند سوال قبل از فایل فشرده ای که در اختیارمان قرار گرفته استفاده میکنیم که شامل داده های باینری مربوط به فایلهاست که به عکس هایی با ابعاد 32×32 تبدیل شده اند. برای این کار ابتدا کتابخانه های مورد نیاز را فراخوانی میکنیم.

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
from rarfile import RarFile, Path
from PIL import Image
import numpy as np
```

با استفاده از کد زیر ابتدا توسط کتابخانه `rarfile` فایل فشرده را باز میکنیم و آدرس فایلهای عکس png درون آن را میخوانیم (به جز آدرس آخر که مربوط به همان فایل فشرده است). پس از آن از این آدرسها که در لیست `filelist` قرار دارند استفاده کرده و با کتابخانه `PIL` فایلهای تصاویر را یک به یک خوانده و محتویات هر عکس را ابتدا `flat` کرده و به یک بردار `1024` درایه ای دست میابیم و کلاس مربوط به عکس مورد نظر را هم که از `csv` مربوط به آن خوانده ایم به عنوان درایه `1025` به بردار اضافه کرده و آن راتبدیل به دیتا فریم میکنیم. برای تمام عکسها این دیتافریم ها را در لیست `images_list` ذخیره میکنیم و در پایان همه را به یک دیتافریم واحد `df` تبدیل میکنیم.

```
#reading Images from the zip file and concatenating them as a
dataframe also adding related class numbers as a column to it

DF_labels= pd.read_csv("trainLabels.csv")
with RarFile("ByteToImage.rar") as zipfiles2:

    #the first entry is the zipfile name
    #we'll skip it
    filelist2 = zipfiles2.namelist()[:-1]
    images_list = []
    for file_name in filelist2:
        m= Image.open(zipfiles2.open(file_name))
        images_list.append(pd.DataFrame(
np.append((np.asarray(m).reshape(-1,)),DF_labels[DF_labels["Id"] ==
file_name[12:-4] ]["Class"].values) ).transpose())
df = pd.concat(images_list)
# at first I did a normalization on the data to make them be between
0 and 1 , But with some experiments I realized that
# it's not nessecary and the model works well without any
normalization so I commented it out.

# df[df.columns[:-1]]= df[df.columns[:-1]]/256.
```

در این مرحله میشد داده های تصویری را با تقسیم کردن بر 256 نرمال کرد اما با انجام آزمایشات مشاهده شد که تاثیر خاصی ندارد و به همین دلیل کامنت شد.

در ادامه اسم ستون آخر را به `Class` تغییر میدهیم.

```
#changing the name of the last column
new_col = np.arange(df.columns[-1]).tolist()+["Class"]
df.columns = new_col
```

در نهایت دیتافریم به شکل زیر بدست می آید

	0	1	2	3	...	1015	1021	1022	1023	Class
0	92	92	95	...	56	57	58	55	9
0	124	123	121	...	117	116	121	121	2
...	...	...	...	...	...	...	...	...	...
0	124	127	126	...	79	83	88	85	4
0	125	122	101	...	73	77	78	75	4

10860 rows × 1025 columns

که 10860 سطر به تعداد عکس ها دارد و 1025 ستون شامل 1024 ستون مربوط به پیکسل های عکس و 1 ستون لیبل دارد.

درضمن عکسهای 32×32 موجود عکس های باینری و سیاه و سفیدی به صورت زیراند.

داده ها از نظر کلاس بندی به صورت زیر هستند.

```
#count of each class in the dataframe

for i in range(9):
    print("num Total class ",i+1," : ",sum(df["Class"]==i+1))
```

```
num Total class  1  :  1533
num Total class  2  :  2478
num Total class  3  :  2942
num Total class  4  :  475
num Total class  5  :  42
num Total class  6  :  751
num Total class  7  :  398
num Total class  8  :  1228
num Total class  9  :  1013
```

در نهایت میتوان این دیتافریم را در یک فایل csv ذخیره کرد.

```
#saving the mentioned Dataframe
filepath = "Binary_DATA_FRAME.csv"
df.to_csv(filepath)
```

حالا دیتای مناسب آموزش را داریم و می توانیم به سراغ اسپارک و استفاده از آن برای انجام عملیات کلاس بندی برویم. پس در ابتدا کتابخانه های مورد نیازمان را فرامیخوانیم و اسپارک را راه اندازی کرده و مطابق کدهای زیر دیتافریممان را از فایل csv خوانده و به یک دیتافریم اسپارک تبدیل میکنیم.

```
#initializing and starting a pyspark session and transform pandas
dataframe to spark dataframe

from pyspark.sql import SparkSession #Import the pyspark
from pyspark.conf import SparkConf #Import the SparkConf
from pyspark.context import SparkContext #Import the SparkContext
import time
from pyspark.sql import SparkSession

DF = pd.read_csv("Binary_DATA_FRAME.csv")
DF = DF.drop('Unnamed: 0', axis=1)
df = DF
conf = SparkConf()
conf.setMaster("local").setAppName("HW2_q2")

sc = SparkContext.getOrCreate(conf=conf)

#Create PySpark SparkSession
spark= SparkSession.builder.master("local").getOrCreate()

sparkDF=spark.createDataFrame(df)
```

حال که داده ها آمده شده و در حافظه هستند داده را به سه قسمت train, test, validation تقسیم میکنیم.

```
#spliting the dataframe to TRAIN,TEST,VALID
TRAIN,TEST,VALID = sparkDF.randomSplit([0.85,0.1,0.05])
```

از کدی که در مرحله قبل استفاده کردیم مجدد استفاده میکنیم و تعداد داده های هر کلاس برای train, test, validation را به دست می آوریم

داده های train

```
num TRAIN class  1  :  1315
num TRAIN class  2  :  2097
num TRAIN class  3  :  2539
num TRAIN class  4  :  395
num TRAIN class  5  :  39
```

```
num TRAIN class  6  :  653
num TRAIN class  7  :  335
num TRAIN class  8  :  1041
num TRAIN class  9  :  867
```

داده های test

```
num TEST class  1  :  139
num TEST class  2  :  248
num TEST class  3  :  278
num TEST class  4  :  42
num TEST class  5  :  2
num TEST class  6  :  72
num TEST class  7  :  40
num TEST class  8  :  127
num TEST class  9  :  89
```

داده های validation

```
num VALID class  1  :  79
num VALID class  2  :  133
num VALID class  3  :  125
num VALID class  4  :  38
num VALID class  5  :  1
num VALID class  6  :  26
num VALID class  7  :  23
num VALID class  8  :  60
num VALID class  9  :  57
```

در مجموع تعداد داده ها به این صورت است که 9281 داده آموزش 1037 داده تست و 542 داده ارزیابی وجود دارد که در ادامه از آن ها استفاده خواهیم کرد.

الف)

برای استفاده از داده هایی که در مرحله قبل به دست آوردیم ابتدا آن ها را shuffle میکنیم تا مشکلی در زمان آموزش پیش نیاید.

```
from pyspark.sql.functions import rand
# TRAIN,TEST,VALID shuffling
TRAIN = TRAIN.orderBy(rand())
TEST = TEST.orderBy(rand())
VALID = VALID.orderBy(rand())
```

حال باید دیتافریم ها را به صورتی دربیاوریم که بتوانیم آن را به عنوان ورودی به مدل خود بدهیم. برای این کار باید یک ستون به عنوان features بسازیم که شامل تمام ویژگی ها یا همان ستون های دیتافرم ها هست به جز ستون مربوط به لیبل ها. برای این کار از کد زیر استفاده میکنیم.

```
#adding a features column which contains all the columns as a single
list, except labels column, to all dataframes

from pyspark.ml.feature import StringIndexer, VectorAssembler


input_cols=VALID_pd.columns[:-1]
output_cols="features"

assembler = VectorAssembler(inputCols=input_cols.to_list(),
outputCol=output_cols)
TRAIN = assembler.transform(TRAIN)
TEST = assembler.transform(TEST)
VALID = assembler.transform(VALID)
```

در نهایت برای مثال داده های ارزیابی به صورت زیر در می آیند.

	0	1	2	...	1022	1023	Class	features
0	118	122	...	122	120	2	[118.0, 122.0, 125.0, 124.0, 117.0, 120.0, 121...
1	131	131	...	103	109	2	[131.0, 131.0, 131.0, 132.0, 132.0, 131.0, 133...
2	101	93	...	43	41	1	[101.0, 93.0, 97.0, 93.0, 88.0, 99.0, 102.0, 1...
...	...	...	...	...	...	...	...
540	133	119	...	117	116	3	[133.0, 119.0, 115.0, 126.0, 130.0, 132.0, 124...
541	117	112	...	96	92	6	[117.0, 112.0, 115.0, 110.0, 122.0, 112.0, 106...

542 rows × 1026 columns

حال دیگر داده ها به صورت کامل آماده ورودی داده شدن به مدل هستند.

حالا باید مدل های random forest خود را براساس خواست سوال آماده کنیم، برای این کار از RandomForestClassifier موجود در اسپارک استفاده میکنیم و ستون ویژگی ها و لیبل ها را مشخص میکنیم و مطابق خواست سوال طی چند آزمایش مقدار و پارامتر maxDepth را بین 3 تا 8 تغییر میدهیم و همچنین تعداد درخت را نیز به دلخواه عدد 10 در نظر میگیریم. لیستی به عنوان predictions در نظر میگیریم که مقادیر پیشبینی شده با هر عمق درخت بر روی داده های تست در آن قرار میگیرد.

```
#building and training a random forest classifier with numTrees=10,
maxDepth=3,4,5,6,7,8
from pyspark.ml.classification import RandomForestClassifier
predictions = []
for dep in range(3,9):
    rf = RandomForestClassifier(numTrees=10, maxDepth= dep
,featuresCol = "features", labelCol = "Class")
    rfModel = rf.fit(TRAIN)
    predictions.append( rfModel.transform(TEST) )
```

برای نمونه مقدار لیبل و پیشبینی برای ده داده اول توسط مدلی با عمق 3 در زیر آمده است.

```
predictions[0].select("Class", "prediction").show(10)
```

```
+-----+----------+
|Class|prediction|
+-----+----------+
|    2|       3.0|
|    2|       2.0|
|    1|       1.0|
|    1|       1.0|
|    2|       2.0|
|    1|       4.0|
|    7|       1.0|
|    3|       3.0|
|    8|       8.0|
|    2|       2.0|
+-----+----------+
only showing top 10 rows
```

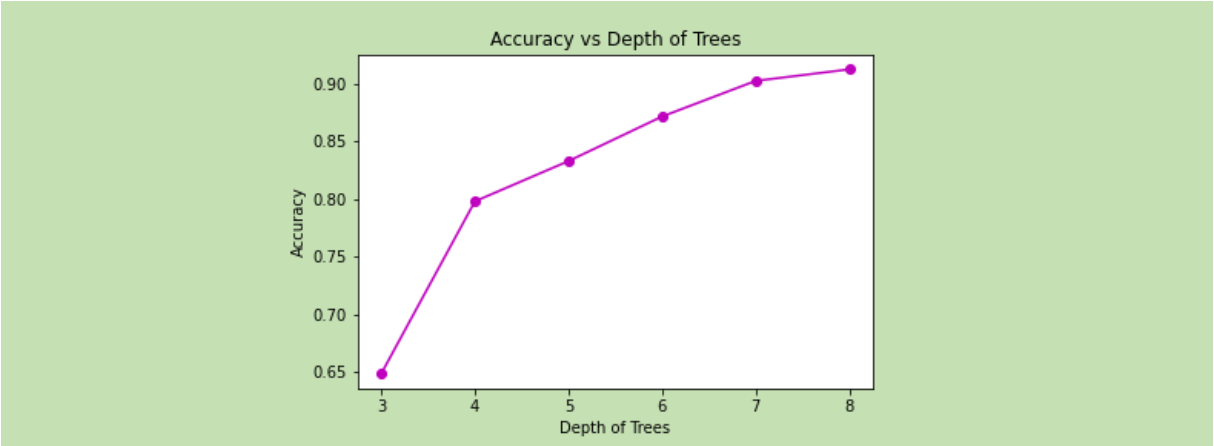
همانطور که مشاهده میکنید برای تعدادی از داده ها پیشبینی درست بوده و برای تعدادی غلط بوده است. برای بررسی دقیق تر میزان دقت یا accuracy را برای همه حالتها با استفاده از کد زیر بدست آورده و در یک لیست ذخیره میکنیم.

```
# calculating the accuracy
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
accuracy=[]
for p in predictions:
    evaluator = MulticlassClassificationEvaluator(labelCol="Class",
predictionCol="prediction")
    accuracy.append(evaluator.evaluate(p))

print("Accuracy = %s" % (accuracy))
```

در نهایت با استفاده از لیست accuracy که بدست آوردیم و کدهای زیر نمودار مربوط به دقت آموزش را برای عمق درخت های مختلف رسم میکنیم.

```
DEPTH_TREES = [3, 4, 5, 6, 7, 8]

# plotting the the accuracy of models for each DEPTH_trees in experi
ments

plt.plot(DEPTH_TREES, accuracy,"mo-")
plt.title("Accuracy vs Depth of Trees")
plt.xlabel("Depth of Trees")
plt.ylabel("Accuracy")
plt.savefig("Accuracy vs DEPTH_trees.png")

# so according to this result we choose DEPTH_trees = 8
```

نمودار دقت برحسب عمق درخت



همانطور که در نمودار مشخص است برای عمق درخت های متفاوت مشاهده میشود که نمودار صعودی بوده و در عمق 8 به بیشینه خود با دقت 0.9125 رسیده است و تقریبا ثابت شده و در نتیجه میتوان آن را به عنوان عمق بهینه درخت در نظر گرفت.

ب)

در این بخش آزمایش دیگری انجام میدهیم و با عمق بهینه ای که پیدا کردیم عملکرد مدل را با تغییر پارامتر MaxBins مورد ارزیابی قرار میدهیم. برای این کار همانند قبل مدل را میسازیم و آموزش میدهیم و فقط مقدار پارامتر عمق را برابر 8 قرار داده و مقدار پارامتر MaxBins را بین مقادیر 4 و 8 و 16 و 32 تغییر میدهیم.

```
# choosing the best_depth
best_depth = DEPTH_TREES[np.argmax(accuracy)]

#building and training a random forest classifier with numTrees=10,
maxDepth=best_depth, Max bins=4, 8, 16, 32

from pyspark.ml.classification import RandomForestClassifier
predictions = []
BINS = [4,8,16,32]
```

```
for bins in BINS:
    rf = RandomForestClassifier(numTrees=10, maxDepth= best_depth ,
maxBins=bins ,featuresCol = "features", labelCol = "Class")
    rfModel = rf.fit(TRAIN)
    predictions.append( rfModel.transform(TEST) )

predictions[0].select("Class", "prediction").show(10)
```

همانطور که مشاهده میکنید پس از آموزش، مقدار پیشبینی شده برای داده های تست در دیتافریم اسپارک predictions ذخیره میگردد. در اینجا ستون های لیبل و پیشبینی را برای ده سطر ابتدایی آموزش با maxBins=4 را برای نمونه نشان میدهیم که به صورت زیر هستند.

```
predictions[0].select("Class", "prediction").show(10)
```

```
+-----+----------+
|Class|prediction|
+-----+----------+
|    2|       2.0|
|    2|       2.0|
|    1|       1.0|
|    1|       1.0|
|    2|       2.0|
|    1|       1.0|
|    7|       7.0|
|    3|       3.0|
|    8|       8.0|
|    2|       2.0|
+-----+----------+
only showing top 10 rows
```

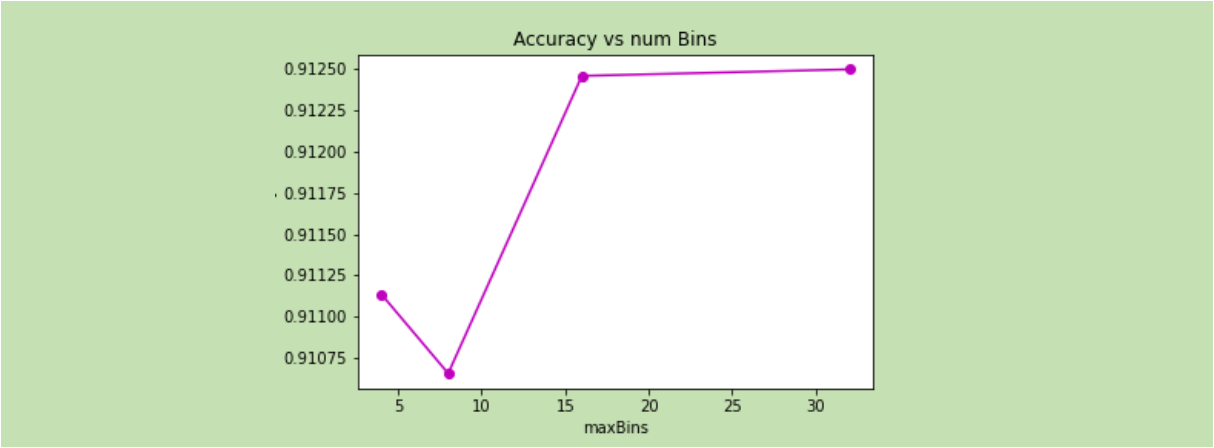
همانطور که مشاهده میکنید به نظر دقت خوبی در پیشبینی دارد. برای بررسی دقیق تر میزان دقت یا accuracy را برای همه حالتها با استفاده از کد زیر بدست آورده و در یک لیست ذخیره میکنیم.

```
# calculating the accuracy
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
accuracy=[]
for p in predictions:
    evaluator = MulticlassClassificationEvaluator(labelCol="Class",
predictionCol="prediction")
    accuracy.append(evaluator.evaluate(p))

print("Accuracy = %s" % (accuracy))
```

و در نهایت با استفاده از کد زیر نمودار تغییرات دقت براساس تغییر maxBins را رسم میکنیم.

```
# plotting the the accuracy of models for each maxBins in experiments

plt.plot(BINS, accuracy,"mo-")
plt.title("Accuracy vs num Bins")
plt.xlabel("maxBins")
plt.ylabel("Accuracy")
plt.savefig("Accuracy vs maxBins.png")
```

که نتیجه زیر بدست می آید.



با توجه به نمودار و میزان تغییرات در دقت به نظر می آید این پارامتر تاثیر چندانی نگذاشته و مقدار تغییرات کم است اما به هرحال در کل نمودار صعودی بوده و به نظر میرسد در انتها به مقدار ثابتی که بیشینه نمودار هست ختم شده که در آن برای `maxBins=32` مقدار دقت برابر با 0.9125 است که همان مقدار دقت بهینه بخش قبل است. این به این خاطر است که مقدار پیشفرض و بدون تعیین این پارامتر همین 32 است و درواقع در بخش قبل این پارامتر روی 32 تنظیم بوده و به نظر می آید همین مقدار 32 مقدار بهینه و مناسب برای این پارامتر است و کمتر از این باعث عملکرد ضعیفتر میشود و بیشتر کردن آن هم تاثیر چندانی ندارد.