me19b190-ass9-ml-final

April 9, 2023

Installing the required packages

[ ]:
```
from pyspark.ml.feature import StringIndexer,VectorAssembler,Normalizer
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.classification import RandomForestClassifier,LinearSVC
from pyspark.ml.classification import GBTClassifier
from pyspark.ml.feature import PCA
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml import Pipeline
from pyspark.sql import SparkSession
from pyspark.ml.feature import Imputer
from pyspark.sql.functions import col, explode, array,␣
 ↪lit,countDistinct,when,regexp_replace

spark = SparkSession.builder.appName('me19b190-ass9').getOrCreate()
```

Downloading the e Pima Indians Diabetes Database and uploading to a google cloud bucket

[ ]:
```
file_location ="gs://dataproc-staging-us-central1-775468331472-mpvg9j7b/
 ↪diabetes.csv"



training = spark.read.format("csv").option("header","True").load(file_location).
 ↪toDF("Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin","BMI","DiabetesPedig
training = training.withColumn("Pregnancies",
training["Pregnancies"].cast("float")).withColumn("Glucose",
training["Glucose"].cast("float")).withColumn("BloodPressure",
training["BloodPressure"].cast("float")).withColumn("SkinThickness",
training["SkinThickness"].cast("float")).withColumn("Insulin",
training["Insulin"].cast("float")).withColumn("BMI",
training["BMI"].cast("float")).withColumn("DiabetesPedigreeFunction",
training["DiabetesPedigreeFunction"].cast("float")).withColumn("Age",
training["Age"].cast("float")).withColumn("Outcome",
training["Outcome"].cast("float"))
training = training.withColumnRenamed("Outcome","label")
```

[ ]:
```python
#replacing the value zero for columns␣
↪Glucose|BloodPressure|SkinThickness|Insulin| BMI|DiabetesPedigreeFunction|␣
↪Age with NULL

training = training.replace(0.0, value=None, subset=training.columns[1:-1])
training.show()
```

```
+-----------+-------+-------------+-------------+-------+----+------------------
------+----+-----+
|Pregnancies|Glucose|BloodPressure|SkinThickness|Insulin|
BMI|DiabetesPedigreeFunction| Age|label|
+-----------+-------+-------------+-------------+-------+----+------------------
------+----+-----+
|        6.0|  148.0|         72.0|         35.0|   null|33.6|
0.627|50.0|  1.0|
|        1.0|   85.0|         66.0|         29.0|   null|26.6|
0.351|31.0|  0.0|
|        8.0|  183.0|         64.0|         null|   null|23.3|
0.672|32.0|  1.0|
|        1.0|   89.0|         66.0|         23.0|   94.0|28.1|
0.167|21.0|  0.0|
|        0.0|  137.0|         40.0|         35.0|  168.0|43.1|
2.288|33.0|  1.0|
|        5.0|  116.0|         74.0|         null|   null|25.6|
0.201|30.0|  0.0|
|        3.0|   78.0|         50.0|         32.0|   88.0|31.0|
0.248|26.0|  1.0|
|       10.0|  115.0|         null|         null|   null|35.3|
0.134|29.0|  0.0|
|        2.0|  197.0|         70.0|         45.0|  543.0|30.5|
0.158|53.0|  1.0|
|        8.0|  125.0|         96.0|         null|   null|null|
0.232|54.0|  1.0|
|        4.0|  110.0|         92.0|         null|   null|37.6|
0.191|30.0|  0.0|
|       10.0|  168.0|         74.0|         null|   null|38.0|
0.537|34.0|  1.0|
|       10.0|  139.0|         80.0|         null|   null|27.1|
1.441|57.0|  0.0|
|        1.0|  189.0|         60.0|         23.0|  846.0|30.1|
0.398|59.0|  1.0|
|        5.0|  166.0|         72.0|         19.0|  175.0|25.8|
0.587|51.0|  1.0|
|        7.0|  100.0|         null|         null|   null|30.0|
0.484|32.0|  1.0|
|        0.0|  118.0|         84.0|         47.0|  230.0|45.8|
0.551|31.0|  1.0|
```

```
|          7.0|  107.0|          74.0|          null|   null|29.6|
0.254|31.0|  1.0|
|          1.0|  103.0|          30.0|          38.0|   83.0|43.3|
0.183|33.0|  0.0|
|          1.0|  115.0|          70.0|          30.0|   96.0|34.6|
0.529|32.0|  1.0|
+-----------+-------+-------------+-------------+-------+----+------------------
------+----+-----+
only showing top 20 rows
```

[ ]:
```
training.columns
```

[ ]:
```
['Pregnancies',
 'Glucose',
 'BloodPressure',
 'SkinThickness',
 'Insulin',
 'BMI',
 'DiabetesPedigreeFunction',
 'Age',
 'label']
```

[ ]:
```
#split train and test data
train,test = training.randomSplit([0.8,0.2],seed = 1)
train.show(5)
```

```
+-----------+-------+-------------+-------------+-------+----+------------------
------+----+-----+
|Pregnancies|Glucose|BloodPressure|SkinThickness|Insulin|
BMI|DiabetesPedigreeFunction| Age|label|
+-----------+-------+-------------+-------------+-------+----+------------------
------+----+-----+
|          0.0|   57.0|          60.0|          null|   null|21.7|
0.735|67.0|  0.0|
|          0.0|   67.0|          76.0|          null|   null|45.3|
0.194|46.0|  0.0|
|          0.0|   73.0|          null|          null|   null|21.1|
0.342|25.0|  0.0|
|          0.0|   74.0|          52.0|          10.0|   36.0|27.8|
0.269|22.0|  0.0|
|          0.0|   84.0|          64.0|          22.0|   66.0|35.8|
0.545|21.0|  0.0|
+-----------+-------+-------------+-------------+-------+----+------------------
------+----+-----+
only showing top 5 rows
```

Null Value imputation

```
[ ]: imputer = Imputer(
         inputCols = train.columns,
         outputCols = train.columns,
     ).fit(train)

     train = imputer.transform(train)
     test = imputer.transform(test)
     train.show(5)
```

```
+-----------+-------+-------------+-------------+---------+----+----------------
--------+----+-----+
|Pregnancies|Glucose|BloodPressure|SkinThickness|  Insulin|
BMI|DiabetesPedigreeFunction| Age|label|
+-----------+-------+-------------+-------------+---------+----+----------------
--------+----+-----+
|        0.0|   57.0|         60.0|    29.363432|154.67284|21.7|
0.735|67.0|  0.0|
|        0.0|   67.0|         76.0|    29.363432|154.67284|45.3|
0.194|46.0|  0.0|
|        0.0|   73.0|      72.5231|    29.363432|154.67284|21.1|
0.342|25.0|  0.0|
|        0.0|   74.0|         52.0|         10.0|     36.0|27.8|
0.269|22.0|  0.0|
|        0.0|   84.0|         64.0|         22.0|     66.0|35.8|
0.545|21.0|  0.0|
+-----------+-------+-------------+-------------+---------+----+----------------
--------+----+-----+
only showing top 5 rows
```

Because it is a classification problem it is important to see if there is any class imbalance

```
[ ]: major_df = train.filter(train.label == 0.0)
     minor_df = train.filter(train.label == 1.0)
     print("Majority class is 0 with count: ",major_df.count())
     print("Minority class is 1 with count: ",minor_df.count())
     ratio = round(major_df.count()/minor_df.count())
     print("ratio of majority and minority class: {}".format(ratio))
```

```
Majority class is 0 with count:  408
Minority class is 1 with count:  221
ratio of majority and minority class: 2
```

Thus, there is class imbalance, Let us explore undersampling and oversampling techniques

```
[ ]: #undersampling of the majority class
     sampled_majority_df = major_df.sample(False, 1/ratio)
```

```
train_undersampling = sampled_majority_df.unionAll(minor_df)
print("Train data with Under Sampling")
train_undersampling.show(5)
print('------------------------------------------------------------------------')
#oversampling
a = range(ratio)
print("Train data with Over Sampling")
# duplicate the minority rows
train_oversampling = minor_df.withColumn("temp_col", explode(array([lit(x) for␣
↪x in a]))).drop('temp_col')
train_oversampling.show(5)
```

```
Train data with Under Sampling
+-----------+-------+-------------+-------------+---------+----+----------------
--------+----+-----+
|Pregnancies|Glucose|BloodPressure|SkinThickness|  Insulin|
BMI|DiabetesPedigreeFunction| Age|label|
+-----------+-------+-------------+-------------+---------+----+----------------
--------+----+-----+
|        0.0|   67.0|         76.0|    29.363432|154.67284|45.3|
0.194|46.0|  0.0|
|        0.0|   73.0|      72.5231|    29.363432|154.67284|21.1|
0.342|25.0|  0.0|
|        0.0|   74.0|         52.0|         10.0|     36.0|27.8|
0.269|22.0|  0.0|
|        0.0|   86.0|         68.0|         32.0|154.67284|35.8|
0.238|25.0|  0.0|
|        0.0|   91.0|         68.0|         32.0|    210.0|39.9|
0.381|25.0|  0.0|
+-----------+-------+-------------+-------------+---------+----+----------------
--------+----+-----+
only showing top 5 rows

------------------------------------------------------------------------
Train data with Over Sampling
+-----------+-------+-------------+-------------+---------+----+----------------
--------+----+-----+
|Pregnancies|Glucose|BloodPressure|SkinThickness|  Insulin|
BMI|DiabetesPedigreeFunction| Age|label|
+-----------+-------+-------------+-------------+---------+----+----------------
--------+----+-----+
|        0.0|  105.0|         84.0|    29.363432|154.67284|27.9|
0.741|62.0|  1.0|
|        0.0|  105.0|         84.0|    29.363432|154.67284|27.9|
0.741|62.0|  1.0|
|        0.0|  107.0|         62.0|         30.0|     74.0|36.6|
0.757|25.0|  1.0|
```

```
|          0.0|   107.0|           62.0|           30.0|      74.0|36.6|
0.757|25.0|  1.0|
|          0.0|   113.0|           76.0|     29.363432|154.67284|33.3|
0.278|23.0|  1.0|
+-----------+-------+-------------+-------------+---------+----+----------------
--------+----+-----+
only showing top 5 rows
```

```
[ ]: #concatenating  the feature columns to make a feature vector
assembler = VectorAssembler(inputCols  =␣
 ↪["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin","BMI","DiabetesPedigreeF
                           outputCol = "temp_features")
```

```
[ ]: df = train
df = assembler.transform(df)
df.show(5, False)
```

```
+-----------+-------+-------------+-------------+---------+----+----------------
--------+----+-----+------------------------------------------------------------
--------------------------------------------------+
|Pregnancies|Glucose|BloodPressure|SkinThickness|Insulin  |BMI
|DiabetesPedigreeFunction|Age |label|temp_features
|
+-----------+-------+-------------+-------------+---------+----+----------------
--------+----+-----+------------------------------------------------------------
--------------------------------------------------+
|0.0        |57.0   |60.0         |29.363432    |154.67284|21.7|0.735
|67.0|0.0  |[0.0,57.0,60.0,29.363431930541992,154.67283630371094,21.700000762939
453,0.7350000143051147,67.0]                 |
|0.0        |67.0   |76.0         |29.363432    |154.67284|45.3|0.194
|46.0|0.0  |[0.0,67.0,76.0,29.363431930541992,154.67283630371094,45.299999237060
55,0.1940000057220459,46.0]                  |
|0.0        |73.0   |72.5231      |29.363432    |154.67284|21.1|0.342
|25.0|0.0  |[0.0,73.0,72.52310180664062,29.363431930541992,154.67283630371094,21
.100000381469727,0.34200000762939453,25.0]|
|0.0        |74.0   |52.0         |10.0         |36.0     |27.8|0.269
|22.0|0.0
|[0.0,74.0,52.0,10.0,36.0,27.799999237060547,0.26899999380111694,22.0]
|
|0.0        |84.0   |64.0         |22.0         |66.0     |35.8|0.545
|21.0|0.0  |[0.0,84.0,64.0,22.0,66.0,35.79999923706055,0.545000016689300 5,21.0]
|
+-----------+-------+-------------+-------------+---------+----+----------------
--------+----+-----+------------------------------------------------------------
--------------------------------------------------+
only showing top 5 rows
```

```
[ ]: normalizer = Normalizer(inputCol  = "temp_features",outputCol="norm_features",p␣
     ↪= 1.0)
```

```
[ ]: df = normalizer.transform(df)
```

```
[ ]: df.select("norm_features").show(5)
```

```
+--------------------+
|       norm_features|
+--------------------+
|[0.0,0.1459774496…|
|[0.0,0.1600840015…|
|[0.0,0.1941482285…|
|[0.0,0.3332297630…|
|[0.0,0.2863522480…|
+--------------------+
only showing top 5 rows
```

Feature Engineering:

For any Machine learning problem, feature extraction is very important. One of the most popular Machine Learning techniques for feature extraction is Principal component analysis("PCA"). Thus, PCA has been performed to extract the 5 most prominent features in a dataset, which will be used for classification

```
[ ]: pca  = PCA(k = 5, inputCol = 'norm_features')
     pca.setOutputCol('features')

     model_pca = pca.fit(df)
     df = model_pca.transform(df)
     df.select("features").show(5)
```

```
+--------------------+
|            features|
+--------------------+
|[0.23471773600388…|
|[0.19963916199366…|
|[0.23166956014258…|
|[-0.0548345104460…|
|[0.02144591420669…|
+--------------------+
only showing top 5 rows
```

```
[ ]: lr = LogisticRegression(maxIter = 10)
     pipeline =  Pipeline(stages = [assembler,normalizer,pca,lr])

     model = pipeline.fit(train)
```

```
pred = model.transform(test)
print("Accuracy - Logistic regression: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='accuracy').
 ↪evaluate(pred)))
print("F1 score - Logistic regression: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
Accuracy - Logistic regression: 0.7410071942446043
F1 score - Logistic regression: 0.7248035417819592
```

[ ]:
```
dt = DecisionTreeClassifier(maxDepth = 5)
pipeline =  Pipeline(stages = [assembler,normalizer,model_pca,dt])
model = pipeline.fit(train)
pred = model.transform(test)
print("Accuracy - Decision Tree Classifier: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='accuracy').
 ↪evaluate(pred)))
print("F1 score - Decision Tree Classifier: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
Accuracy - Decision Tree Classifier: 0.6690647482014388
F1 score - Decision Tree Classifier: 0.6773739449998443
```

[ ]:
```
rf = RandomForestClassifier(maxDepth = 6,numTrees = 100,featuresCol =␣
 ↪'features',featureSubsetStrategy = 'sqrt')
pipeline =  Pipeline(stages = [assembler,normalizer,model_pca,rf])
model = pipeline.fit(train)
pred = model.transform(test)
print("Accuracy - Random Forest Classifier: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='accuracy').
 ↪evaluate(pred)))
print("F1 score - Random Forest Classifier: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
Accuracy - Random Forest Classifier: 0.7769784172661871
F1 score - Random Forest Classifier: 0.7684902309848136
```

[ ]:
```
gbt = GBTClassifier(labelCol="label", featuresCol="features", maxIter=10)
pipeline =  Pipeline(stages = [assembler,normalizer,model_pca,gbt])
model = pipeline.fit(train)
pred = model.transform(test)
print("Accuracy - GBTClassifier: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='accuracy').
 ↪evaluate(pred)))
print("F1 score - GBTClassifier: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
Accuracy - GBTClassifier: 0.6834532374100719
F1 score - GBTClassifier: 0.6863961891711121
```

[ ]:
```
svc = LinearSVC(labelCol="label", featuresCol="features", maxIter=10,regParam=0.
 ↪1)
pipeline =  Pipeline(stages = [assembler,normalizer,model_pca,svc])
model = pipeline.fit(train)
pred = model.transform(test)
print("Accuracy - GBTClassifier: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='accuracy').
 ↪evaluate(pred)))
print("F1 score - GBTClassifier: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
Accuracy - GBTClassifier: 0.6618705035971223
F1 score - GBTClassifier: 0.527204210657448
```

Let us see, if undersampling works better

[ ]:
```
lr = LogisticRegression(maxIter = 10)
pipeline =  Pipeline(stages = [assembler,normalizer,pca,lr])

model = pipeline.fit(train_undersampling)
pred = model.transform(test)
print("Accuracy - Logistic regression: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='accuracy').
 ↪evaluate(pred)))
print("F1 score - Logistic regression: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
Accuracy - Logistic regression: 0.6402877697841727
F1 score - Logistic regression: 0.6498902572856968
```

[ ]:
```
dt = DecisionTreeClassifier(maxDepth = 5)
pipeline =  Pipeline(stages = [assembler,normalizer,model_pca,dt])
model = pipeline.fit(train_undersampling)
pred = model.transform(test)
print("Accuracy - Decision Tree Classifier: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='accuracy').
 ↪evaluate(pred)))
print("F1 score - Decision Tree Classifier: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
Accuracy - Decision Tree Classifier: 0.5971223021582733
F1 score - Decision Tree Classifier: 0.5997508538322613
```

[ ]:
```
rf = RandomForestClassifier(maxDepth = 6,numTrees = 100,featuresCol =␣
 ↪'features',featureSubsetStrategy = 'sqrt')
```

```
pipeline =  Pipeline(stages = [assembler,normalizer,model_pca,rf])
model = pipeline.fit(train_undersampling)
pred = model.transform(test)
print("Accuracy - Random Forest Classifier: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='accuracy').
 ↪evaluate(pred)))
print("F1 score - Random Forest Classifier: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
Accuracy - Random Forest Classifier: 0.7050359712230215
F1 score - Random Forest Classifier: 0.7128305917181585
```

[ ]:
```
gbt = GBTClassifier(labelCol="label", featuresCol="features", maxIter=10)
pipeline =  Pipeline(stages = [assembler,normalizer,model_pca,gbt])
model = pipeline.fit(train_undersampling)
pred = model.transform(test)
print("Accuracy - GBTClassifier: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='accuracy').
 ↪evaluate(pred)))
print("F1 score - GBTClassifier: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
Accuracy - GBTClassifier: 0.6402877697841727
F1 score - GBTClassifier: 0.6481010680177662
```

[ ]:
```
svc = LinearSVC(labelCol="label", featuresCol="features", maxIter=10,regParam=0.
 ↪1)
pipeline =  Pipeline(stages = [assembler,normalizer,model_pca,svc])
model = pipeline.fit(train_undersampling)
pred = model.transform(test)
print("Accuracy - GBTClassifier: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='accuracy').
 ↪evaluate(pred)))
print("F1 score - GBTClassifier: {} ".
 ↪format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
Accuracy - GBTClassifier: 0.6402877697841727
F1 score - GBTClassifier: 0.6499078016149341
```

Let us see if oversampling techniques works better

[ ]:
```
lr = LogisticRegression(maxIter = 10)
pipeline =  Pipeline(stages = [assembler,normalizer,pca,lr])

model = pipeline.fit(train_oversampling)
pred = model.transform(test)
```

```
print("Accuracy - Logistic regression: {} ".
↪format(MulticlassClassificationEvaluator(metricName='accuracy').
↪evaluate(pred)))
print("F1 score - Logistic regression: {} ".
↪format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
23/04/07 12:08:47 WARN org.apache.spark.ml.util.Instrumentation: [1e5e6976] All
labels are the same value and fitIntercept=true, so the coefficients will be
zeros. Training is not needed.
```

```
Accuracy - Logistic regression: 0.3381294964028777
F1 score - Logistic regression: 0.1708826487197339
```

[ ]:
```
dt = DecisionTreeClassifier(maxDepth = 5)
pipeline =  Pipeline(stages = [assembler,normalizer,model_pca,dt])
model = pipeline.fit(train_oversampling)
pred = model.transform(test)
print("Accuracy - Decision Tree Classifier: {} ".
↪format(MulticlassClassificationEvaluator(metricName='accuracy').
↪evaluate(pred)))
print("F1 score - Decision Tree Classifier: {} ".
↪format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
Accuracy - Decision Tree Classifier: 0.3381294964028777
F1 score - Decision Tree Classifier: 0.1708826487197339
```

[ ]:
```
rf = RandomForestClassifier(maxDepth = 6,numTrees = 100,featuresCol =␣
↪'features',featureSubsetStrategy = 'sqrt')
pipeline =  Pipeline(stages = [assembler,normalizer,model_pca,rf])
model = pipeline.fit(train_oversampling)
pred = model.transform(test)
print("Accuracy - Random Forest Classifier: {} ".
↪format(MulticlassClassificationEvaluator(metricName='accuracy').
↪evaluate(pred)))
print("F1 score - Random Forest Classifier: {} ".
↪format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
Accuracy - Random Forest Classifier: 0.3381294964028777
F1 score - Random Forest Classifier: 0.1708826487197339
```

[ ]:
```
gbt = GBTClassifier(labelCol="label", featuresCol="features", maxIter=10)
pipeline =  Pipeline(stages = [assembler,normalizer,model_pca,gbt])
model = pipeline.fit(train_oversampling)
pred = model.transform(test)
print("Accuracy - GBTClassifier: {} ".
↪format(MulticlassClassificationEvaluator(metricName='accuracy').
↪evaluate(pred)))
```

```
print("F1 score - GBTClassifier: {} ".
↪format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
Accuracy - GBTClassifier: 0.3381294964028777
F1 score - GBTClassifier: 0.1708826487197339
```

```
[ ]: svc = LinearSVC(labelCol="label", featuresCol="features", maxIter=10,regParam=0.
     ↪1)
     pipeline =  Pipeline(stages = [assembler,normalizer,model_pca,svc])
     model = pipeline.fit(train_oversampling)
     pred = model.transform(test)
     print("Accuracy - GBTClassifier: {} ".
     ↪format(MulticlassClassificationEvaluator(metricName='accuracy').
     ↪evaluate(pred)))
     print("F1 score - GBTClassifier: {} ".
     ↪format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
Accuracy - GBTClassifier: 0.3381294964028777
F1 score - GBTClassifier: 0.1708826487197339
```

Conclusion:

Random Forest classifier without under sampling or oversampling is giving best results followed by logistic regression (without undersampling or oversampling)

Accuracy - Random Forest Classifier: 0.7769784172661871

F1 score - Random Forest Classifier: 0.7684902309848136

Accuracy - Logistic regression: 0.7410071942446043

F1 score - Logistic regression: 0.7248035417819592