PROJECT REPORT

CMPE-256 - Large Scale Analytics

Submitted By: Group 7

Team Members:
Qiao Liu - 013802893
Ching-Min Hu – 013726154
Dandan Zhao – 013795392
Fernanda Bordin - 013800638
Megha Rajam Rao - 013709488
Rajasree Rajendran - 013774358

Best ML Algorithm: Random Forest classifier

Google Colab link :
https://colab.research.google.com/drive/1zzd4R4r3TFaM2OUQJ6nAhfdgWPZOR8IB

Referred Online Links: See Appendix B

TABLE OF CONTENTS

1. Task Assignment

Task	Description	Main Contributor	Other Contributor
Data Preparation	Load, pre-process, validate and visualize data	Dandan Zhao, Ching-Min Hu	Rajasree Rajendran, Megha Rajam Rao, Fernanda Bordin, Qiao Liu
ML methods	Logistic classification Naïve Bayes Classifier Random Forest Classifier	Rajasree Rajendran Megha Rajam Rao	Dandan Zhao, Ching-Min Hu, Fernanda Bordin, Qiao Liu
PowerPoint presentation	Overall Input on Data preparation Input on lessons learned	Fernanda Bordin	Rajasree Rajendran, Megha Rajam Rao, Dandan Zhao, Ching-Min Hu, Qiao Liu
Project Report	Overall Input on Data Preparation Input on graphics	Qiao Liu	Rajasree Rajendran, Megha Rajam Rao, Dandan Zhao, Ching-Min Hu, Fernanda Bordin

Table 1. Table of Task Assignment

2. Introduction

In this study, we compare the performance between PySpark MLib library and Scikit-learn through running naive machine learning algorithms for image recognition. By training our models to recognize 2 classes of mammals: medium-sized mammals and small mammals, we learn the difference between doing machine learning with PySpark MLib library and Scikit-learn.

The CIFAR datasets are labeled subsets of the 80 million tiny images dataset collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The images are of size 32x32 pixels with 3 color channels (RGB). It comprises of 100 classes containing 600 images each (500 training and 100 testing). The classes (fine labels) are grouped into 20 super classes (coarse labels) and corresponding classes. In this report we filtered the CIFAR-100 dataset to select images from the super classes we chose, which are medium-sized mammals and small mammals. The medium-sized mammals superclass includes the following classes: fox, porcupine**,** possum, raccoon, skunk. Small mammal's superclass includes the following classes: Hamster, mouse, rabbit, shrew, squirrel. The models should be able to differentiate between the 2 classes. We will compare the 2 approaches based on accuracy and speed. We will also discuss our personal experience on working with the 2 different approaches.

2.a. Libraries

1. NumPy
2. Scikit-learn
3. Matplotlib
4. Math
5. Seaborn
6. Spark MLib library

2.b. Software & Tools

1. Google Colaboratory
2. Python (language)
3. PowerPoint (presentation)
4. Word (report)
5. Google drive (document sharing)

3. Comparison between PySpark MLib and Scikit-learn

3.a. Qualitative Comparison

We first conducted studies on industrial opinions of the differences between the 2 approaches: Machine learning with PySpark MLib and Scikit-learn. Based on opinions of Villu Ruusmann, distributed systems such as Spark works best with running simple models for large datasets.

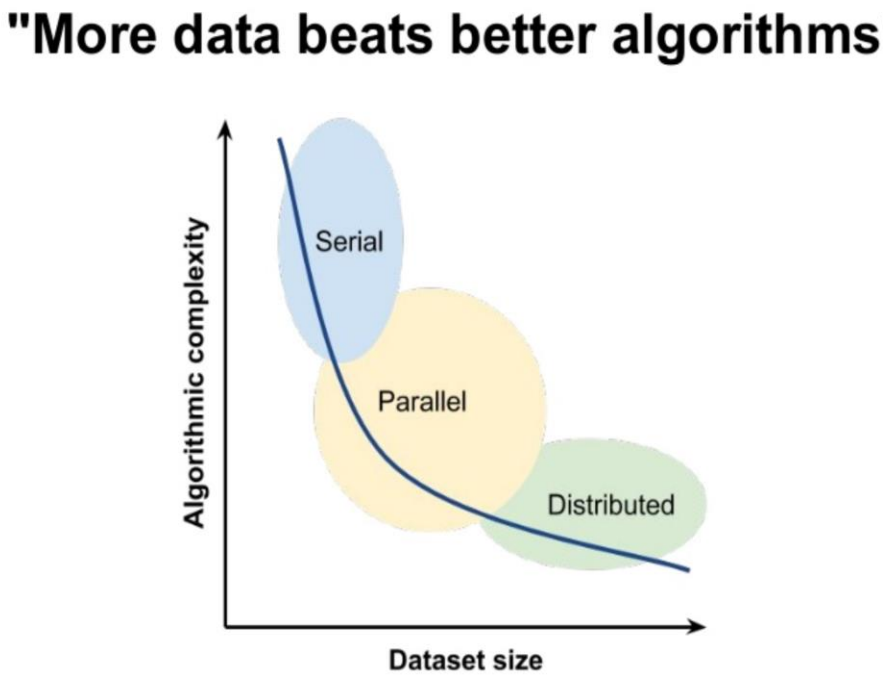


Fig 3.1. Ruusmann, V. (2017). Openscouring.IO.

Yet Scikit-learn, as one of the most popular machine learning libraries in current industries, also has the advantage of better visualization tools and rich ecosystem. We made 2 tables to compare the pros and cons of the 2 approaches:

Table of Comparison between the advantages of Machine Learning (ML) with Scikit-learn and ML with Spark in Python environment

ML with Sckit-learn	ML with Spark (Python)
Easy to use	Uses caching to reuse data
Great visualization tools (Pandas and Matplotlib)	Has accumulators (keep state across iterations)
Rich ecosystem (many libraries)	Fault tolerance
ML models run smoothly and are easy to optimize	Popular algorithms supported

Table 3.1 Table of Comparison between the advantages of Machine Learning (ML) with Scikit-learn and ML with Spark in Python environment

Table of Comparison between the disadvantages of Machine Learning (ML) with Scikit-learn and ML with Spark in Python environment

ML with Sckit-learn	ML with Spark (Python)
Limited to one machine	Takes a long time to aggregate dataframe (ML Lib not efficient)
	Memory expensive
	High latency
	Spark doesn't provide handy functions to print score, confusion matrix and classification report in one go.

Table 3.2 Table of Comparison between the disadvantages of Machine Learning (ML) with Scikit-learn and ML with Spark in Python environment

3.b. Quantitative comparison

To compare the performance between Spark MLib and Scikit-learn, we ran the same models from both libraries on the same dataset for a binary classification problem. All models were run in Google Colaboratory with TPU. By comparing the results, we see that PySpark has faster CPU time than Scikit-learn but longer wall time. This is because we must wait for server response from Spark but can almost instantly run on Python.

For model accuracy, we compared the same models that gave us the best results for Milestone 1 from the previous project. The results did not show significantly better accuracy for one package over the other. We also tested other models such as Extra Decision Tree and SVM, but they have poor accuracy for PySpark.

Model results overview:

Table of Models Results Comparison

	Scikit-Learn (Last semester)		**PySpark**	
	Accuracy	Time	Accuracy	Time
Logistic Regression	59.60%	CPU: 33.8 s Wall: 33.8 s	62.17%	CPU: 64.5 ms Wall: 2 min 40s
Naive Bayes	61.0%	CPU: 168 ms Wall: 143 ms	60.15%	CPU: 49.9 ms Wall: 3 min 19s
Random Forest	61.2%	CPU: 1.9 s Wall: 1.91 s	66.64%	CPU: 91.8 ms Wall: 8 min

Table 3.3 Table of Models Results Comparison

3.c. Code comparison

We compared codes that have similar functionality for the 2 approaches and listed them in the below table.

Table of Code Comparison between Scikit-Learn & Spark

Steps		**Scikit-Learn**	**Spark**
Data download and preparation	Resource	from keras.datasets import cifar100	Download cifar100 python version from https://www.cs.toronto.edu/~kriz/cifar.html
	Read the data	(x_train, y_train), (x_test, y_test) = cifar100.load_data()	First, unzip the file and upload the cifar100 to colab; Second, unpickle the train data and test data; Third, convert the train and test file to RDD respectively
	Combine train and test data	np.concatenate	By using union to combine two RDD, Transform the RDD to DataFrame by using spark.createDataFrame
	Filter out the assigned superclasses	Generate the target indexes for two superclasses. By using enumerate function slice the target data index and label	Generate the target indexes for two superclasses(same). By using filter with isin to filter out the target data filter(col('fine_labels').isin(target_index))
	Display the first 5 rows	df.head()	target_df.show(5)
	Take a random sample without replacement	df.sample(frac=0.5, replace=True, random_state=1)	Seed is used to save the state of the random function in subsequent executions.

			combine_rdd.takeSample(withReplacement=False, num=5, seed=123) We can also use orderBy function with rand to randomly order the dataframe. Further, limit is used to choose the number of rows. df.select([col1, col2]).orderBy(rand()).limit(36).rdd.collect()
Visualize and validate the data	Validate the data	Normalization and reshape the data x_train /= 255.0 x_test /= 255.0 x_train.reshape(x_train.shape[0],3*32*32)	Normalization the data/255.0,change the data type to DenseVector()and reshape the data using reshape(3,32,32)(similar)
		By using np.array and enumerate function to get the data and the label to do the validation	By using sampleBy function to get the part of the data in a ratio, then pick 4 samples for each class sampleBy('fine', fractions)
	Visualize the data	Matplotlib plt.imshow(img) Define grid function add the fine label and coarse label	matplotlib(same) plt.imshow(img) Add two columns coarse and fine label to the dataFrame, generate the picture with the label easily.
Modify the format for model building step	Convert coarse label 'small mammals' and 'medium mammals' into a binary label '0.0' And '1.0'.	np.array([[int(y[0] in medium_sized_mammals_index)] for y in y_train]	stringindexer = StringIndexer(inputCol='coarse_labels', outputCol='binary_index') target_name_df = stringindexer.fit(target_name_df).transform(target_name_df)

	Rename column ‘data’ as ‘features’ and re-order columns	df.rename(columns={"Data": "features"}) df = df[['features', 'binary_index', 'coarse_labels', 'coarse', 'fine_labels',’fine]]	target_name_df = target_name_df.withColumnRenamed("data","features").select("features","binary_index", "coarse_labels","coarse","fine_labels","fine")
	Check for data type of each column	df.dtypes	target_name_df.dtypes Similar utility was used in spark and pandas.
	Check for null values in each column	Use isna(), isnull() or isnull().sum().	for c in target_name_df.columns: print ("Column",c, "- no.of null values:", target_name_df.where(col(c).isNull()).count()) The absence of a straightforward null value check was highly inconvenient. We wanted to check in greater granularity and decided to find the null values in each column. It took more than 9 minutes to execute.
Generate random Train-test split	Randomly select 80% of data as training data and remaining 20% data as testing data. Seed is used to save the state of the random function in subsequent executions.	from sklearn.model_selection import train_test_split X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)	train_df, test_df= target_name_df.randomSplit([0.80,0.20], seed=1369)
	Verify the number of entries in training and testing data	X_train.shape x_test.shape	train_df.count() test_df.count() We can use spark sql functions too.

	Verify the distinct labels	np.unique()	train_df.select('coarse_labels','binary_index','coarse').distinct().collect() train_df.select('coarse_labels','binary_index','coarse').distinct().collect()
Verify the input data	Check the type of file	type(df)	type(train_df) Similar utility was used in spark and pandas.
	Check the schema	df.info()	train_df.printSchema()
	Check the statistical features of the dataset	df.describe()	train_df.describe().show()

Define a function for model prediction visualization	Function to print out randomly chosen images and their labels from predictions	Matplotlib
		plt.imshow(img)
		
		Define grid function add the fine label and coarse label
			samples = predictions.select(['coarse', 'prediction','fine','binary_index','features']).orderBy(rand()).limit(36).rdd.collect()
			
			matplotlib
			plt.imshow(img)
			
			Sample of 36 images, number restricted using limit function, is randomly generated using rand() function. The user-defined function for image with the label is defined using matplotlib (same as in previous milestone). Incorrect predictions were labeled in red using a parameter color as shown below.
			
			if label == pred:
			plt.title("Predicted: {}\n Actual: {}".format(pred,label))
			else:
			plt.title("Predicted: {}\n Actual: {}".format(pred,label), color = "red")

	Import package	From sklearn.linear_model	from pyspark.ml.classification import LogisticRegression

Model 1 – Logistic Regression		import LogisticRegression	
	Instantiate the model	lr = LogisticRegression()	lr = LogisticRegression(labelCol="binary_index",featuresCol="features",maxIter=10)
	Fit the model	lr.fit(x_train_1, y_train_bin)	model=lr.fit(train_df)
	Predict using the model and print the first 10 rows from the resultant dataframe.	lr_pred=lr.predict(x_test_1) cifar_grid(x_test_1, y_test_bin,indices,4,lr_pred)	predict_lr=model.transform(test_df) predict_lr.select("coarse","binary_index","prediction","probability").show(10) Total execution time for this model was more than 2 minutes.
	Print the prediction score / accuracy	print ("Logistic Regression Accuracy: {}%".format(lr.score(x_test_1, y_test_bin)*100))	BinaryClassificationEvaluator' or multiclassclassificationevaluator can be used for finding the accuracy of the 3 models. eval=BinaryClassificationEvaluator(labelCol="binary_index", rawPredictionCol= "prediction") accuracy = (eval.evaluate(predict_lr))*100 print("Model Accuracy: %.3f%%" % accuracy)
Model 2 – Naïve Bayes Classifier	Import package	from sklearn.naive_bayes import MultinomialNB	from pyspark.ml.classification import NaiveBayes
	Instantiate the model	naive = MultinomialNB()	naive_bayes = NaiveBayes(featuresCol="features", labelCol="binary_index",smoothing=1.0, modelType="multinomial")

	Fit the model	naive.fit(x_train_1, y_train_bin)	naive_bayes = naive_bayes.fit(train_df)
	Predict using the model and print the first 10 rows from the resultant dataframe.	naive_predict= naive.predict(x_test_1) cifar_grid(x_test_1, y_test_bin,indices,4,naive_predict)	predict_nb = naive_bayes.transform(test_df) predict_nb.select("coarse","binary_index","prediction","probability").show(10) Total execution time for this model was more than 3 minutes.
Model 3 – Random Forest Classifier	Import package	from sklearn.ensemble import RandomForestClassifier	from pyspark.ml.classification import RandomForestClassifier
	Instantiate the model	logit = RandomForestClassifier()	rfc=RandomForestClassifier(featuresCol="features", labelCol="binary_index",numTrees=100)
	Fit the model	logit.fit(x_train_1, y_train_bin)	rfc_model=rfc.fit(train_df)
	Predict using the model and print the first 10 rows from the resultant dataframe.	logit_pred=logit.predict(x_test_1) cifar_grid(x_test_1, y_test_bin,indices,4,logit_pred)	predict_rfc=rfc_model.transform(test_df) predict_rfc.select("coarse","binary_index","prediction","probability").show(10) Total execution time for this model was more than 8 minutes.
Confusion matrix and classification report	Confusion matrix	confusion_matrix(y_test_bin, logit_pred, labels=None, sample_weight=None)	**Method 1** -Simple version using below code by converting the dataframe to RDD and using zipWithIndex and countByKey functions. conf_mat1 = predict_lr.select("binary_index","prediction") print (conf_mat1.rdd.zipWithIndex().countByKey()) **Method 2** - Formatted version using below code using multiclassmetrics.

			predictionRDD_1 = predict_lr.select(['binary_index', 'prediction']) \ .rdd.map(lambda line: (line[1], line[0])) metrics1 = MulticlassMetrics(predictionRDD_1) cm1 = metrics1.confusionMatrix().toArray() Thereafter, it is printed using for loop. The absence of a ready-made function made this step inconvenient as it took nearly 6 minutes for both methods to execute in pyspark.
	Classification report	classification_report(y_test_bin, logit_pred)	Below function was created using 'Multiclassmetrics' to print the classification report with precision, recall, f1-score and support def cr1(label_in): precision = metrics1.precision(label=label_in) recall = metrics1.recall(label=label_in) F1_Measure = metrics1.fMeasure(label=label_in) support = test_df.filter(test_df.binary_index==label_in).count() print("%10s %12.2f %12.2f %12.2f %12d" % \ (label_in,precision, recall, F1_Measure, support)) Thereafter, a for loop was created to format and print the report. Similar, to confusion matrix, lack of function made this step inconvenient. It took nearly 6-7 minutes to execute.

Table 3.4 Table of Code Comparison between PySpark MLib and Scikit-learn

4. EDA

4.a Data Preparation

- For preparation, we installed OpenJDK and findspark packages to set up the environment and build Spark session. Then, we imported all the necessary packages and modules such as Spark SQL and MLib.
- Reading in the data with PySpark is not as straightforward as importing from Keras datasets with ***cifar100.load_data ()*** function. With much effort and experimentation, we downloaded the python version of CIFAR100 dataset from online source: https://www.cs.toronto.edu/~kriz/cifar.html.
- The downloaded dataset includes three files: training data, test data and metadata. We converted the training and test data into Spark RDD.
- Then we applied the ***unpickle*** function provided by https://www.cs.toronto.edu/~kriz/cifar.html to convert the dataset into dictionaries. We c hose four key-value pairs: {filename: name of the image}, {coarse label: index of superclass}, {fine label: index of fine class}, and {image data: list of pixels of the image} to generate the RDD for later operation. The values of filename consisted of the name of the image, the type was byte, and we had to convert the byte to string type.
- With ***zip*** function, we transformed values with different keys into one tuple for each record, then used ***parallelize*** function to generate the RDD for both train and test data.

```
# generate the train data RDD
cifar_RDD_train = sc.parallelize(list(zip( \
    [name.decode('utf-8') for name in datatrain.get(b'filenames')],\
    datatrain.get(b'fine_labels'), datatrain.get(b'coarse_labels'), \
    datatrain.get(b'data'))))
cifar_RDD_train.count()
```

```
50000
```

- With ***union*** function, we combined test RDD and train RDD. The total count of records is 60000.

```
[ ]  # combine the train RDD and test RDD into one RDD
     combine_rdd = cifar_RDD_train.union(cifar_RDD_test)
     combine_rdd.count()
```

```
60000
```

- We transformed the RDD into Spark DataFrame by with ***createDataFrame()*** and ***map()*** function. During this step, we converted the data type to ***DenseVector*** so Spark can recognize them as arrays of value. We normalized the image data by dividing it with the highest pixel value, which is 255.

```
[ ]  # Convert the combined RDD into spark dataframe and normalize
     %%time
     combine_df = spark.createDataFrame(combine_rdd.map(lambda x: Row(filenames=x[0],\
                  fine_labels=int(x[1]), coarse_labels=int(x[2]), data=DenseVector(x[3])/255.0)) )
     print (combine_df.count())
```

```
60000
CPU times: user 28.9 ms, sys: 5.94 ms, total: 34.8 ms
Wall time: 1min 52s
```

- In our DataFrame, there are four columns, ***filename, coarse label, fine label and data***.

```
combine_df.show()
```

```
+-------------+--------------------+--------------------+-----------+
|coarse_labels|                data|           filenames|fine_labels|
+-------------+--------------------+--------------------+-----------+
|           11|[255.0,255.0,255....|bos_taurus_s_0005...|         19|
|           15|[255.0,253.0,253....|stegosaurus_s_000...|         29|
|            4|[250.0,248.0,247....|mcintosh_s_000643...|          0|
|           14|[124.0,131.0,135....|altar_boy_s_00143...|         11|
|            1|[43.0,32.0,87.0,1...|cichlid_s_000031.png|          1|
|            5|[190.0,191.0,194....|  phone_s_002161.png|         86|
|           18|[50.0,56.0,52.0,5...|car_train_s_00004...|         90|
|            3|[178.0,175.0,175....| beaker_s_000604.png|         28|
|           10|[122.0,127.0,134....|    fog_s_000397.png|         23|
|           11|[255.0,255.0,255....|rogue_elephant_s_...|         31|
|            5|[83.0,67.0,61.0,5...|computer_keyboard...|         39|
|           17|[254.0,255.0,255....|willow_tree_s_000...|         96|
|            2|[227.0,233.0,231....|sunflower_s_00054...|         82|
|            9|[253.0,251.0,252....| palace_s_000759.png|         17|
|           10|[183.0,185.0,189....|adriatic_s_001782...|         71|
|            5|[44.0,64.0,131.0,...|computer_keyboard...|         39|
```

- We generated indexes of 2 superclasses as a list. The assigned superclasses (small mammals and medium-sized mammals) were filtered by using given metadata.

```
# get the index of ten classes of animals
medium_sized_mammals_index = [ fine_labels.index(x) for x in ['fox', 'porcupine', 'possum', 'raccoon', 'skunk'] ]
small_mammals_index = [ fine_labels.index(x) for x in ['hamster', 'mouse', 'rabbit', 'shrew', 'squirrel'] ]

target_index = medium_sized_mammals_index + small_mammals_index
print (medium_sized_mammals_index, small_mammals_index)
print(target_index)
```

```
[34, 63, 64, 66, 75] [36, 50, 65, 74, 80]
[34, 63, 64, 66, 75, 36, 50, 65, 74, 80]
```

- We filtered out the assigned classes with ***filter...isin ()*** function.

```
from pyspark.sql.functions import col
#filter out the assigned ten fine classes
target_df = combine_df.filter(col('fine_labels').isin(target_index))
target_df.count()
```

```
6000
```

4.b. Data Validation

- We validated our data on the generated DataFrame. With ***.withColumn()*** function, we added two columns ***coarse*** and ***fine*** which contain the verbal labels for each record.

```
# Add two columns fine label as fine and coarse label as coarse
%%time
target_name_df = target_df.withColumn("coarse", udf(lambda coarse_label: coarse_labels[coarse_label], \
                                                    StringType())(target_df['coarse_labels'])) \
    .withColumn("fine", udf(lambda fine_label: fine_labels[fine_label], StringType())(target_df['fine_labels']))
```

```
CPU times: user 11 ms, sys: 1.57 ms, total: 12.6 ms
Wall time: 52 ms
```

- Let's check if the columns were created correctl

```
[ ] # Print the first 3 rows
    %%time
    target_name_df.show(3)
```

```
+--------------------+-------------+-----------+--------------------+--------------+--------+
|                data|coarse_labels|fine_labels|           filenames|        coarse|    fine|
+--------------------+-------------+-----------+--------------------+--------------+--------+
|[0.27843137254901...|           16|         80|squirrel_s_002467...| small_mammals|squirrel|
|[0.61176470588235...|           16|         74|  shrew_s_002233.png| small_mammals|   shrew|
|[0.96078431372549...|           12|         64|opossum_s_001237.png|medium_mammals|  possum|
+--------------------+-------------+-----------+--------------------+--------------+--------+
only showing top 3 rows
```

- Check statistical features of the data.

```
[ ] # Display the statistical features of the data
    %%time
    target_name_df.describe().show()
```

```
+-------+-----------------+------------------+--------------------+--------------+--------+
|summary|    coarse_labels|       fine_labels|           filenames|        coarse|    fine|
+-------+-----------------+------------------+--------------------+--------------+--------+
|  count|             6000|              6000|                6000|          6000|    6000|
|   mean|             14.0|              60.7|                null|          null|    null|
| stddev|2.000166687502895|15.014911741191348|                null|          null|    null|
|    min|               12|                34|american_water_sh...|medium_mammals|     fox|
|    max|               16|                80|water_shrew_s_000...| small_mammals|squirrel|
+-------+-----------------+------------------+--------------------+--------------+--------+
```

- We used ***sampleBy()*** function to generate a sample DataFrame for validation purpose and transformed it back to RDD with ***.collect().*** We got the list containing all records including data and labels, randomly picked 4 samples for each class to perform visual inspection and validation.

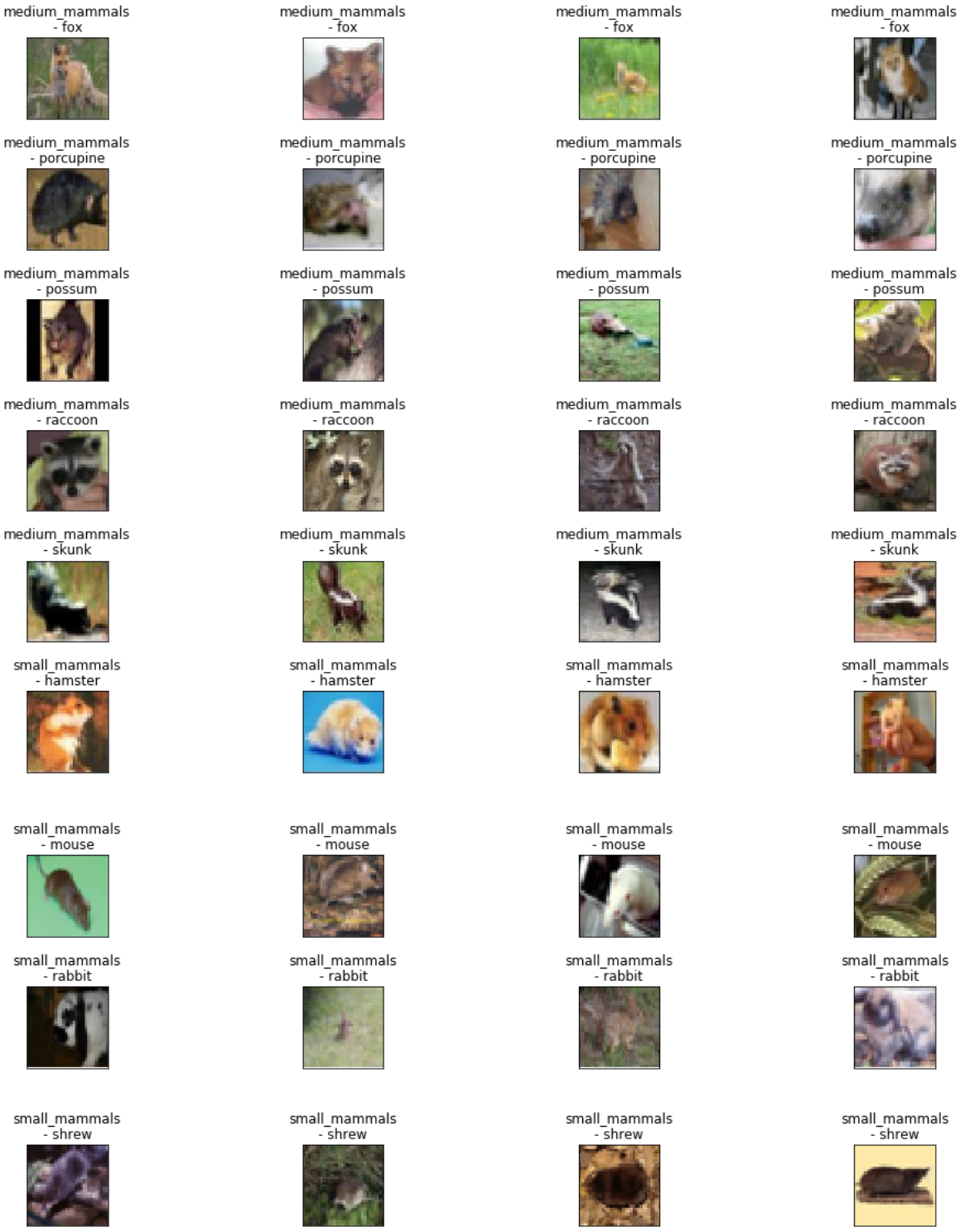
```
# generate sample data randomly
fractions = dict()
for fine in ['fox', 'porcupine', 'possum', 'raccoon', 'skunk', 'hamster', 'mouse', 'rabbit', 'shrew', 'squirrel']:
  fractions[fine] = 0.04      #24pictures for each class

sample_df = target_name_df.sampleBy('fine', fractions)
sample_df.count()
```

```
256
```

```
# generate the sample list in order to use later
samples = sample_df.select('coarse', 'fine', 'ndata').orderBy('coarse', 'fine').rdd.collect()
```

- Visualization was an essential step towards validation of the filtered data. We generated the 4 random images from each subclass using ***matplotlib***. As instructed, the title included the verbal superclass name (coarse label) and subclass name (fine label). The following were the validation images, we manually check the images, they all matched their fine labels and coarse labels.



- Once the dataset was validated and ready, we proceeded to model training.

5. Data Modeling

Because wall time for Spark is a bit long, we took several methods to improve our speed:

1. Optimize memory usage in system. Empty recycle bin.
2. Optimize browser usage. Remove unwanted plugins, pop-ups, ads, tabs and clear history.
3. Using TensorFlow processing unit (TPU) as runtime type.
4. Increase memory in Google Colaboratory using below code:

```
#spark = SparkSession.builder.master("local[*]").getOrCreate()
Memory_limit = "12g"
spark      =      SparkSession.builder.appName("Foo").config("spark.executor.memory",
    Memory_limit).config("spark.driver.memory", Memory_limit).getOrCreate()
```

- Because we have defined our issue as a binary classification machine learning problem.

We first combined the coarse labels of each superclass into a binary label.

```
[ ]  # Change the coarse label into a binary label
     %%time
     stringindexer = StringIndexer(inputCol='coarse_labels', outputCol='binary_index')
     target_name_df = stringindexer.fit(target_name_df).transform(target_name_df)

     # Select the distinct binary indices
     print ("Distinct binary indeices:", target_name_df.select('binary_index').distinct().collect() )

     Distinct binary indeices: [Row(binary_index=0.0), Row(binary_index=1.0)]
     CPU times: user 61.2 ms, sys: 17.2 ms, total: 78.4 ms
     Wall time: 3min 41s
```

- Then we renamed the data as features and re-ordered the columns.

```
[ ]  # Rename data as features and re-order columns
     %%time
     target_name_df = target_name_df.withColumnRenamed("data","features").select("features", \
                                    "binary_index", "coarse_labels","coarse","fine_labels","fine")

     CPU times: user 4.58 ms, sys: 1.39 ms, total: 5.97 ms
     Wall time: 40.9 ms
```

- We verified that we have the correct data type for training.

```
# Verify the data type
%%time
target_name_df.dtypes
```

```
CPU times: user 360 µs, sys: 1.86 ms, total: 2.22 ms
Wall time: 18.1 ms
[('features', 'vector'),
 ('binary_index', 'double'),
 ('coarse_labels', 'bigint'),
 ('coarse', 'string'),
 ('fine_labels', 'bigint'),
 ('fine', 'string')]
```

- Checking for any missing values in the column, since null values would affect our training.

```
[ ] # Check for null values in each column
    %%time
    for c in target_name_df.columns:
        print ("Column",c, "- no.of null values:", target_name_df.where(col(c).isNull()).count())
```

```
Column features - no.of null values: 0
Column binary_index - no.of null values: 0
Column coarse_labels - no.of null values: 0
Column coarse - no.of null values: 0
Column fine_labels - no.of null values: 0
Column fine - no.of null values: 0
CPU times: user 75 ms, sys: 20.3 ms, total: 95.3 ms
Wall time: 9min 5s
```

- We randomly split the whole data into 80% training data and 20% test data.

```
[ ] # Split randomly into training set and testing set
    %%time
    train_df, test_df = target_name_df.randomSplit([0.80, 0.20], seed=1369)
```

- Check that we have the correct number of entries for train and test set.

```
[ ]  # Count the number of entries in training and testing data
     %%time
     print ("Training set count:", train_df.count(), \
                    "\nTesting set count:", test_df.count())
```

```
Training set count: 4857
Testing set count: 1143
CPU times: user 26.3 ms, sys: 6.57 ms, total: 32.9 ms
Wall time: 3min 38s
```

- We verified that we have the correct training and test data by examining their datatype, schema and statistical features. Below are the screenshots for training data. Same validation procedure is applied to this test data as well. We can see the first 5 rows of training data below.

```
[ ]  ## Print the first 5 rows
     %%time
     train_df.show(5)
```

```
+--------------------+------------+-------------+--------------+-----------+-------+
|            features|binary_index|coarse_labels|        coarse|fine_labels|   fine|
+--------------------+------------+-------------+--------------+-----------+-------+
|[0.0,0.0,0.0,0.0,...|         1.0|           12|medium_mammals|         34|    fox|
|[0.0,0.0,0.0,0.0,...|         1.0|           12|medium_mammals|         75|  skunk|
|[0.0,0.0078431372...|         0.0|           16| small_mammals|         36|hamster|
|[0.0,0.0470588235...|         0.0|           16| small_mammals|         36|hamster|
|[0.00392156862745...|         0.0|           16| small_mammals|         36|hamster|
+--------------------+------------+-------------+--------------+-----------+-------+
only showing top 5 rows

CPU times: user 5.3 ms, sys: 2.59 ms, total: 7.9 ms
Wall time: 44.3 s
```

Verifying that the datatype is what we needed for training:

```
[ ]  # Verify the training set file type
     %%time
     print ("Training data - Type", type(train_df))

     Training data - Type <class 'pyspark.sql.dataframe.DataFrame'>
     CPU times: user 0 ns, sys: 999 µs, total: 999 µs
     Wall time: 1.15 ms
```

Verifying that we have the correct schema:

```
[ ]  # Verify the Schema
     %%time
     train_df.printSchema()

     root
      |-- features: vector (nullable = true)
      |-- binary_index: double (nullable = false)
      |-- coarse_labels: long (nullable = true)
      |-- coarse: string (nullable = true)
      |-- fine_labels: long (nullable = true)
      |-- fine: string (nullable = true)
```

Checking the statistics to see if there's any obvious outliers or irregularity:

```
[ ]  # Display the statistical features
     %%time
     train_df.describe().show()

     +-------+------------------+------------------+-------------+-----------------+--------+
     |summary|      binary_index|     coarse_labels|       coarse|      fine_labels|    fine|
     +-------+------------------+------------------+-------------+-----------------+--------+
     |  count|              4857|              4857|         4857|             4857|    4857|
     |   mean|0.4968087296685197|14.012765081325922|         null| 60.6438130533251|    null|
     | stddev|0.5000412946929902|2.0001651787719608|         null|15.04927242806568|    null|
     |    min|               0.0|                12|medium_mammals|               34|     fox|
     |    max|               1.0|                16| small_mammals|               80|squirrel|
     +-------+------------------+------------------+-------------+-----------------+--------+

     CPU times: user 14.8 ms, sys: 3 ms, total: 17.8 ms
     Wall time: 1min 48s
```

5.a. Logistic Regression

- We first created a Logistic Regression model to classify our data.

```
[ ] # Create a Logistic Regression model, fit, predict and print relevant columns from the first 10 rows
    %%time
    lr = LogisticRegression(labelCol="binary_index", featuresCol="features",maxIter=10)
    model=lr.fit(train_df)
    predict_lr=model.transform(test_df)
    predict_lr.select("coarse","binary_index","prediction","probability").show(10)
```

```
+--------------+------------+----------+--------------------+
|        coarse|binary_index|prediction|         probability|
+--------------+------------+----------+--------------------+
| small_mammals|         0.0|       1.0|[0.47472145551301...|
|medium_mammals|         1.0|       1.0|[0.43350257031928...|
| small_mammals|         0.0|       0.0|[0.66759395480564...|
| small_mammals|         0.0|       0.0|[0.65729166310509...|
|medium_mammals|         1.0|       1.0|[0.43337212473262...|
|medium_mammals|         1.0|       0.0|[0.67365162847310...|
|medium_mammals|         1.0|       1.0|[0.40707934151075...|
|medium_mammals|         1.0|       0.0|[0.50087052162286...|
|medium_mammals|         1.0|       1.0|[0.14650473985964...|
|medium_mammals|         1.0|       1.0|[0.43103612241028...|
+--------------+------------+----------+--------------------+
only showing top 10 rows
```

- The accuracy score for Logistic Regression is 62.17%.

```
# Print the prediction score / accuracy
%%time
eval=BinaryClassificationEvaluator(labelCol="binary_index", rawPredictionCol= "prediction")
accuracy = (eval.evaluate(predict_lr))*100
print("Model Accuracy: %.3f%%" % accuracy)
```

```
Model Accuracy: 62.173%
CPU times: user 16.6 ms, sys: 4.09 ms, total: 20.7 ms
Wall time: 1min 47s
```

- We randomly printed out 36 images with original and predicted labels to visually inspect our results. The labels printed in red are the wrong predictions and black ones are right predictions.

Predicted: medium mammals
Actual: medium mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: small mammals
Actual: medium mammals

Predicted: small mammals
Actual: small mammals

Predicted: medium mammals
Actual: small mammals

Predicted: small mammals
Actual: medium mammals

Predicted: small mammals
Actual: small mammals

Predicted: small mammals
Actual: medium mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: small mammals
Actual: medium mammals

Predicted: medium mammals
Actual: small mammals

Predicted: small mammals
Actual: medium mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: small mammals
Actual: small mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: small mammals
Actual: small mammals

Predicted: small mammals
Actual: small mammals

Predicted: small mammals
Actual: small mammals

Predicted: small mammals
Actual: medium mammals

Predicted: medium mammals
Actual: small mammals

Predicted: small mammals
Actual: small mammals

Predicted: medium mammals
Actual: small mammals

Predicted: small mammals
Actual: medium mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: small mammals
Actual: medium mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: small mammals
Actual: small mammals

Predicted: small mammals
Actual: medium mammals

Predicted: medium mammals
Actual: small mammals

Predicted: medium mammals
Actual: small mammals

Predicted: small mammals
Actual: small mammals

Predicted: small mammals
Actual: small mammals

Predicted: medium mammals
Actual: small mammals

Predicted: small mammals
Actual: medium mammals

Predicted: small mammals
Actual: small mammals

- The Confusion Matrix, with comparison with Scikit-Learn Logistic Regression Confusion Matrix from last semester, is shown in the below table:

Logistic Regression Confusion Matrix Comparison

Scikit-Learn (Last semester)			**PySpark**		
	predict 0	predict 1		predict 0	predict 1
real 0	293	207	real 0	411	145
real 1	197	303	real 1	291	296

Table 5.a.1 Table of Logistic Regression Confusion Matrix Comparison

- Classification report for both Scikit-Learn and PySpark is shown below:

Scikit-Learn

	precision	recall	f1-score	support
0	0.60	0.59	0.59	500
1	0.59	0.61	0.60	500
accuracy			0.60	1000
macro avg	0.60	0.60	0.60	1000
weighted avg	0.60	0.60	0.60	1000

PySpark

Classification Report

label	precision	recall	f1-score	support
0.0	0.59	0.74	0.65	556
1.0	0.67	0.50	0.58	587

times: user 58.8 ms, sys: 12.5 ms, total: 71.3 ms
. time: 7min 15s

- We can see that PySpark model performs slightly better at eliminating False Negatives than Scikit-Learn. But both models have around 60% accuracy.

5.b. Naïve Bayes

- We created a Naïve Bayes classifier-based model with binary index.

```
[ ] # Create a Naive bayes classifier-based model, fit, predict and print relevant columns from the first 10 rows
    %%time
    naive_bayes = NaiveBayes(featuresCol="features", labelCol="binary_index",smoothing=1.0, modelType="multinomial")
    naive_bayes = naive_bayes.fit(train_df)
    predict_nb = naive_bayes.transform(test_df)
    predict_nb.select("coarse","binary_index","prediction","probability").show(10)

+--------------+------------+----------+--------------------+
|        coarse|binary_index|prediction|         probability|
+--------------+------------+----------+--------------------+
| small_mammals|         0.0|       1.0|[0.04699485125869...|
|medium_mammals|         1.0|       1.0|[0.37349437754014...|
| small_mammals|         0.0|       1.0|[0.49742404147437...|
| small_mammals|         0.0|       0.0|[0.60447148053640...|
|medium_mammals|         1.0|       1.0|[0.08935989872676...|
|medium_mammals|         1.0|       0.0|[0.99281403497744...|
|medium_mammals|         1.0|       1.0|[0.00413106018955...|
|medium_mammals|         1.0|       0.0|[0.98257673622296...|
|medium_mammals|         1.0|       1.0|[1.38216751282667...|
|medium_mammals|         1.0|       0.0|[0.65504292132605...|
+--------------+------------+----------+--------------------+
only showing top 10 rows

CPU times: user 36.4 ms, sys: 13.5 ms, total: 49.9 ms
Wall time: 3min 19s
```

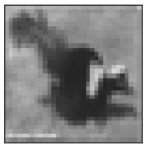
- The accuracy score for Naïve Bayes is **60.15%.**

```
[ ] # Print the prediction score / accuracy
    %%time
    accuracy2 = (eval.evaluate(predict_nb))*100
    print("Model Accuracy: %.3f%%" % accuracy2)

Model Accuracy: 60.152%
CPU times: user 13.5 ms, sys: 5.09 ms, total: 18.6 ms
Wall time: 1min 47s
```

- We randomly printed out 36 images with original and predicted labels to visually inspect our results. The labels printed in red are the wrong predictions and the black ones are right predictions.

Predicted: medium mammals
Actual: medium mammals



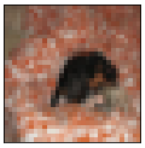
Predicted: medium mammals
Actual: medium mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: medium mammals
Actual: small mammals

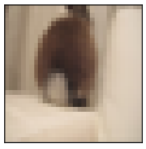
Predicted: medium mammals
Actual: medium mammals



Predicted: small mammals
Actual: small mammals

Predicted: medium mammals
Actual: small mammals

Predicted: small mammals
Actual: small mammals



Predicted: medium mammals
Actual: medium mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: small mammals
Actual: medium mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: medium mammals
Actual: small mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: small mammals
Actual: small mammals

Predicted: small mammals
Actual: small mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: medium mammals
Actual: small mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: small mammals
Actual: small mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: small mammals
Actual: small mammals

Predicted: small mammals
Actual: small mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: medium mammals
Actual: small mammals

Predicted: medium mammals
Actual: small mammals

Predicted: medium mammals
Actual: small mammals

Predicted: small mammals
Actual: small mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: medium mammals
Actual: small mammals

Predicted: small mammals
Actual: medium mammals

Predicted: small mammals
Actual: small mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: medium mammals
Actual: medium mammals

Predicted: medium mammals
Actual: small mammals

- The Confusion Matrix, with comparison with Scikit-Learn Naïve Bayes Classifier Confusion Matrix from last semester, is shown in the below table:

Naïve Bayes Classifier Confusion Matrix Comparison

Scikit-Learn (Last semester)			**PySpark**		
	predict 0	predict 1		predict 0	predict 1
real 0	311	189	real 0	344	212
real 1	201	299	real 1	244	343

Table 5.b.1 Table of Naïve Bayes Classifier Confusion Matrix Comparison

- Classification report for both Scikit-Learn and PySpark is shown below:

Scikit-Learn

	precision	recall	f1-score	support
0	0.61	0.62	0.61	500
1	0.61	0.60	0.61	500
accuracy			0.61	1000
macro avg	0.61	0.61	0.61	1000
weighted avg	0.61	0.61	0.61	1000

PySpark

Classification Report

label	precision	recall	f1-score	support
0.0	0.59	0.62	0.60	556
1.0	0.62	0.58	0.60	587

- We can see that Scikit-Learn Naïve Bayes have more balance in predicting the binary labels, but PySpark predicts 1 label better than 0 labels.

5.c. Random Forest

- We created a Random Forest classifier-based model with binary index.

```
[ ] # Create a Random Forest classifer-based model, fit, predict and print relevant columns from the first 10 rows
    %%time
    rfc=RandomForestClassifier(featuresCol="features",  labelCol="binary_index",numTrees=100)
    rfc_model=rfc.fit(train_df)
    predict_rfc=rfc_model.transform(test_df)
    predict_rfc.select("coarse","binary_index","prediction","probability").show(10)
```

```
+--------------+------------+----------+--------------------+
|        coarse|binary_index|prediction|         probability|
+--------------+------------+----------+--------------------+
| small_mammals|         0.0|       1.0|[0.40035757638691...|
|medium_mammals|         1.0|       1.0|[0.38387068492749...|
| small_mammals|         0.0|       1.0|[0.43530567957958...|
| small_mammals|         0.0|       0.0|[0.65135556347857...|
|medium_mammals|         1.0|       1.0|[0.38914469976349...|
|medium_mammals|         1.0|       1.0|[0.46753362003885...|
|medium_mammals|         1.0|       1.0|[0.37085833903610...|
|medium_mammals|         1.0|       0.0|[0.52357669190181...|
|medium_mammals|         1.0|       1.0|[0.27146339848296...|
|medium_mammals|         1.0|       1.0|[0.35838241330998...|
+--------------+------------+----------+--------------------+
```

- The accuracy score for Random Forest is **66.64%.**

```
[ ] # Print the prediction score / accuracy
    %%time
    accuracy3 = (eval.evaluate(predict_rfc))*100
    print("Model Accuracy: %.3f%%" % accuracy3)
```

```
Model Accuracy: 66.640%
CPU times: user 13.7 ms, sys: 4.93 ms, total: 18.6 ms
Wall time: 1min 48s
```

- We randomly printed out 36 images with original and predicted labels to visually inspect our results. The labels printed in red are the wrong predictions.

Predicted: small mammals
Actual: medium mammals


Predicted: small mammals
Actual: medium mammals

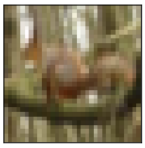

Predicted: medium mammals
Actual: medium mammals


Predicted: medium mammals
Actual: medium mammals


Predicted: medium mammals
Actual: small mammals


Predicted: medium mammals
Actual: medium mammals


Predicted: medium mammals
Actual: small mammals

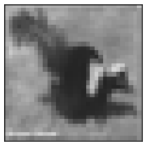

Predicted: medium mammals
Actual: small mammals


Predicted: small mammals
Actual: medium mammals


Predicted: small mammals
Actual: small mammals


Predicted: medium mammals
Actual: medium mammals


Predicted: small mammals
Actual: small mammals


Predicted: medium mammals
Actual: medium mammals


Predicted: medium mammals
Actual: small mammals


Predicted: medium mammals
Actual: medium mammals

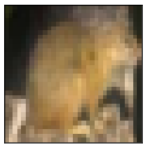

Predicted: small mammals
Actual: small mammals


Predicted: small mammals
Actual: small mammals


Predicted: small mammals
Actual: medium mammals


Predicted: medium mammals
Actual: medium mammals


Predicted: medium mammals
Actual: medium mammals


Predicted: small mammals
Actual: small mammals


Predicted: medium mammals
Actual: medium mammals


Predicted: small mammals
Actual: small mammals


Predicted: small mammals
Actual: medium mammals


Predicted: small mammals
Actual: small mammals


Predicted: small mammals
Actual: medium mammals


Predicted: small mammals
Actual: small mammals


Predicted: small mammals
Actual: medium mammals


Predicted: medium mammals
Actual: small mammals


Predicted: medium mammals
Actual: small mammals


Predicted: medium mammals
Actual: medium mammals


Predicted: small mammals
Actual: small mammals


Predicted: medium mammals
Actual: medium mammals


Predicted: small mammals
Actual: medium mammals


Predicted: medium mammals
Actual: small mammals


Predicted: small mammals
Actual: small mammals


- The Confusion Matrix, with comparison with Scikit-Learn Random Forest Classifier Confusion Matrix from last semester, is shown in the below table:

Naïve Bayes Classifier Confusion Matrix Comparison

Scikit-Learn (Last semester)			**PySpark**		
	predict 0	predict 1		predict 0	predict 1
real 0	361	139	real 0	383	173
real 1	244	256	real 1	209	378

Table 5.c.1 Table of Random Forest Classifier Confusion Matrix Comparison

- Classification report for both Scikit-Learn and PySpark is shown below:

Scikit-Learn

	precision	recall	f1-score	support
0	0.60	0.72	0.65	500
1	0.65	0.51	0.57	500
accuracy			0.62	1000
macro avg	0.62	0.62	0.61	1000
weighted avg	0.62	0.62	0.61	1000

PySpark

Classification Report

label	precision	recall	f1-score	support
0.0	0.65	0.69	0.67	556
1.0	0.69	0.64	0.66	587

- We can see that PySpark have more balance in predicting the binary labels, but Scikit-Learn predicts 1 label better than 0 labels. During training, accuracy score for Random Forest can vary from 62% to 70%, showing the model is not very robust.

6. Conclusion

In this milestone, we used multiple algorithms to train models to recognize small sized mammals and medium sized mammals with PySpark MLib and compared it with results from last semester which were predicted with Scikit-Learn. During the project, we experimented and learned the difference between PySpark MLib an Scikit-Learn. We pick **Random Forest** as our best model because it has the best accuracy. In general, doing Machine Learning with PySpark MLib has been an educative experience. Through the practice, we learned the difference in design philosophy can reflect on the products of different systems. For instance, Spark is designed as an Analytics Engine for Big Data, so it performs better on large scale data. Scikit-Learn is designed as a python machine-learning library, so it provides a better machine learning workflow.

Lessons Learned:

1. PySpark MLib runs faster in CPU time than Scikit-Learn. But in practice, Spark takes longer wall time since our large dataset had longer queue time from server;

2. Scikit-Learn fits better with our working habit. Scikit-Learn is a well-developed machine learning package with consistent parameter formats and rich ecosystem while MLib requires extra steps to perform some of the basic operations in Scikit-Learn. Input data parameters are also inconsistent for MLib

3. Some simple operations are time consuming in Spark, e.g. combining datasets, ***count(), describe()*** and ***union()*** equivalents. A faster workaround for count is to use Spark SQL.

4. Spark is ideal for running simple models on large datasets with Scala as its programming language on a RAM-rich hardware. Our Laptop computer experienced some memory issues while

running Spark. When your RAM can handle the amount of data, then Spark would be a good choice to perform machine learning;

5. A good workflow we concluded for machine learning on large scale data is: first train and optimize your model on a small data sample with Scikit learn, then run the large dataset using Spark.

Future Improvement:

For future work, we plan on improving the performance of our Spark models in 2 ways: 1. Use pipelining; 2. Optimize our models with parameter tuning if time allows.

Appendix A. References

1. Opala, M. (2018). *Top Machine Learning Frameworks Compared: Scikit-Learn, Dlib, MLib, Tensorflow, and More*. Retrieved from netguru.com: https://www.netguru.com/blog/top-machine-learning-frameworks-compared

2. Quesada, J., & Anderson, D. (2016). *Data Science Retreat*. Retrieved from https://www.slideshare.net/JoseQuesada5/a-full-machine-learning-pipeline-in-scikitlearn-vs-in-scalaspark-pros-and-con

3. Ruusmann, V. (2017). *Openscouring.IO*. Retrieved from https://www.slideshare.net/VilluRuusmann/r-scikitlearn-and-apache-spark-ml-what-difference-does-it-make

Appendix B. Referred online links

1) https://spark.apache.org/docs/2.3.0/ml-classification-regression.html
2) https://spark.apache.org/docs/2.3.0/ml-tuning.html
3) https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-mllib/spark-mllib-BinaryClassificationEvaluator.html

4) https://stackoverflow.com/questions/41714698/how-to-get-accuracy-precision-recall-and-roc-from-cross-validation-in-spark-ml

5) https://drive.google.com/file/u/0/d/1ZJwp9Oyp5adE7J57XEj_OOdYLDPO_q3B/edit

6) https://drive.google.com/file/u/0/d/1kennf0tB893aLbIRY8CD4tmo-nbErLQx/edit

7) https://stackoverflow.com/questions/48202900/what-does-these-parameters-mean-in-jupyter-notebook-when-i-input-time

8) http://spark.apache.org/docs/latest/api/python/pyspark.ml.html

9) http://spark.apache.org/docs/latest/api/python/pyspark.mllib.html

10) https://machinelearningmastery.com/how-to-manually-scale-image-pixel-data-for-deep-learning/

11) https://www.programcreek.com/python/example/93276/pyspark.ml.classification.LogisticRegression

12) http://benalexkeen.com/multiclass-text-classification-with-pyspark/

13) https://jarrettmeyer.com/2017/05/04/random-forests-with-pyspark

14) https://medium.com/@dhiraj.p.rai/logistic-regression-in-spark-ml-8a95b5f5434c

15) https://docs.databricks.com/applications/machine-learning/mllib/binary-classification-mllib-pipelines.html

16) https://runawayhorse001.github.io/LearningApacheSpark/classification.html

17) https://towardsdatascience.com/multi-class-text-classification-with-pyspark-7d78d022ed35

18) https://towardsdatascience.com/machine-learning-with-pyspark-and-mllib-solving-a-binary-classification-problem-96396065d2aa