# **Assignment 3: Apache Spark**

Emily Strong

CSYE 7245 Spring 2018

## Introduction

For this assignment, I used three of the Apache Spark modules to analyze the Labeled Faced in the Wild data set. While Apache Spark is not ideal for image analysis, I was able to find useful applications for each of the modules with my data. Structured Streaming and GraphFrames were both useful in identifying the individuals with the largest number of images in the set, a metric I used in selecting individuals for use with MLlib. In my project, I use a SVM to classify face embeddings generated by the OpenFace pre-trained model. Here I tested out the MLlib classification algorithms that currently support multinomial classification, and also compared those results to training the TensorFlow Inception model designed for image classification on the raw images. My results are consistent with the fact that facial recognition requires different models than standard image classification, however MLlib's Random Forest classifier would be a good choice in combination with a facial embedding CNN for setting up a streaming facial recognition pipeline.

## Methodology

### Data Set

My data set is the Labeled Faces in the Wild (LFW) [1] data set, the data set from my term project. The LFW contains over 13,000 labeled pictures of famous people, with 1680 of the subjects having multiple photos in the set. I modified the data set in various ways described in the relevant sections below.

### Part A: Set Up Apache Spark

I set up PySpark on my computer using a conda install.

### Part B: Structured Streaming

Structured Streaming does not support image formats so I wrote the label for each image in my data set to a csv, and then randomly split them into a batch and streaming set. I then followed the model from the Databricks Introduction to Structured Streaming [2] for setting up the data stream. As more data came in I ran successively more complex queries, starting with a select \* and progressing to the labels with more than 50 occurrences in the data set, the same query I used on the batch data.

### Part C: MLlib

I took the five people identified in Part B as having more than 100 photos in the LFW (at least 50 in batch and at least 50 in stream), generated the embeddings for their images using the OpenFace CNN [3] and used that data with the Decision Tree Classifier, Random Forest Classifier, and Multinomial Logistic Regression Classifier libraries in Spark MLlib. In setting these up I followed the Binary Classification Example [4] and Decision Trees Example [5] from Databricks. I did not use the other classification models available as Gradient Boosted Trees, Support Vector Machines, and One-Vs-Rest only work with binomial data and the Naïve Bayes implementation doesn't support negative values. For each model I ran, I timed the training and obtained precision, recall and confusion matrices. For comparison, I ran the raw images for the five people through TensorFlow Inception using the retraining tutorial [6]. For this I used the script that comes with the model. Due to its poor accuracy (discussed in the results section), I did not attempt to set it up with Spark for additional analysis.

### Part D: GraphFrames

I encountered difficulty getting GraphFrames to run in Jupyter Notebook or as a Python script. Both attempts threw errors related to the GraphFrames classes not being loaded. Multiple sources indicated that to fix this error the jar file for GraphFrames needed to be added to the Python path or the PySpark path however neither of these resolved the issue. I thus performed my graph analysis using the PySpark shell. If I had used Databricks for the assignment I likely would not have encountered this problem. A full copy of my work in the PySpark shell is in pyspark\_commands.rtf.

For my data, I converted the image files to two csvs: one listing all people and images, the other listing the images (src) and people pictured (dst). I added to the data set a second set of relationships, the errata from the LFW [7] of images that are mislabeled. I structured the errata as src: image file, dst: correct person and added a label column of image/errata so that it could be combined with the images to create network connections. I then followed the tutorial on using GraphFrames with Python [8] for my analysis, as well as performing centrality analysis to confirm that the results are consistent with the structured streaming results.

## Results

***Structured Streaming***

In my structured streaming analysis, I monitored the data as it came in, initially rerunning a select \* query until there was enough data to start filtering the results. I then selected labels that had a count > 1 indicating multiple images of that person in the set. I also checked the counts of the number of labels and the number of images processed. After 20 minutes there were enough people with >1 image that the results began to be truncated. After allowing the process to run for several hours, it completed processing the data in the streaming set, indicating 6664 images of 3499 people. At that point I compared this set to the static set on the number of people with more than 50 images in each set and found that both had the same list of five, indicating there are only five people in the LFW with more than 100 images.

*Table 1: People with >100 images*

|  |  |  |
| --- | --- | --- |
| **Person** | **Batch Count** | **Streaming Count** |
| George\_W\_Bush | 253 | 277 |
| Colin\_Powell | 117 | 119 |
| Tony\_Blair | 62 | 82 |
| Donald\_Rumsfeld | 64 | 57 |
| Gerhard\_Schroeder | 53 | 56 |

***Machine Learning***

Of the three Spark MLlib algorithms I tested, random forest performed the best. It was the longest to train, however the fact that the difference in training time for a single decision tree versus a random forest (0.237 seconds) demonstrates the power of Spark's parallelization in speeding up a typically time consuming process. The precision (96%) and recall (95%) for random forest were comparable to the accuracy of the OpenFace pipeline which uses an SVM.

Logistic regression was the fastest to train however it had poor precision and recall, and the confusion matrix (table 3) suggests that although the documentation indicates it supports multinomial classification, it only used two of the five labels when classifying the test data.

Since this assignment included an option of using TensorFlow, I decided to try the Inception image classification model to compare the results to classification based on embeddings as the face recognition pipeline in my project uses. I used what should have been two true positives (an image of Colin Powell and an image of George W Bush) and a false positive (an image of Tiger Woods), but I found that even trained on a limited data set Inception was incapable of correctly identifying any faces (table 4). The correct labels for the two images that should have been true positives were both predicted as the second best fit out of five. Because of this poor performance, I did not attempt to run the model with Spark as though it might have sped up the training time (the script took 17 minutes to run) it would not have improved the results.

*Table 2: Training Times and Accuracy Measures*

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Training time (seconds)** | **Precision** | **Recall** |
| Decision Tree (Spark ML) | 2.174 | 0.909685 | 0.903885 |
| Random Forest (Spark ML) | 2.411 | 0.95955 | 0.9591 |
| Logistic Regression (Spark ML) | 0.651 | 0.39937 | 0.593047 |

*Table 3: Confusion Matrices*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Actual** | **Predicted** | | | | | | |
| *Decision Tree* | 1: Gerhard Schroeder | 2: Colin Powell | 3: Tony Blair | | 4: Donald Rumsfeld | | 5: George W Bush |
| 1: Gerhard Schroeder | 43 | 3 | 2 | | 2 | | 1 |
| 2: Colin Powell | 0 | 89 | 6 | | 0 | | 1 |
| 3: Tony Blair | 0 | 5 | 65 | | 0 | | 0 |
| 4: Donald Rumsfeld | 4 | 1 | 2 | | 49 | | 0 |
| 5: George W Bush | 7 | 1 | 4 | | 8 | | 196 |
| *Random Forest* |  | | | | | | |
| 1: Gerhard Schroeder | 47 | 0 | | 1 | | 0 | 3 |
| 2: Colin Powell | 0 | 94 | | 1 | | 0 | 1 |
| 3: Tony Blair | 0 | 4 | | 63 | | 2 | 1 |
| 4: Donald Rumsfeld | 1 | 0 | | 0 | | 55 | 0 |
| 5: George W Bush | 0 | 1 | | 2 | | 3 | 210 |
| *Logistic Regression* |  | | | | | | |
| 1: Gerhard Schroeder | 0 | 0 | | 0 | | 0 | 51 |
| 2: Colin Powell | 0 | 74 | | 0 | | 0 | 22 |
| 3: Tony Blair | 0 | 16 | | 0 | | 0 | 54 |
| 4: Donald Rumsfeld | 0 | 1 | | 0 | | 0 | 55 |
| 5: George W Bush | 0 | 0 | | 0 | | 0 | 216 |

*Table 4: TensorFlow Inception*

|  |  |
| --- | --- |
| **Test Image** | **Predictions** |
| Colin Powell | gerhard schroeder 0.547417  colin powell 0.35788  donald rumsfeld 0.0423357  tony blair 0.0319602  george w bush 0.0204071 |
| George W Bush | gerhard schroeder 0.56776  george w bush 0.289871  colin powell 0.111549  tony blair 0.0297545  donald rumsfeld 0.0010649 |
| Tiger Woods | george w bush 0.555008  tony blair 0.234888  gerhard schroeder 0.13264  colin powell 0.0765645  donald rumsfeld 0.000900364 |

***Part D***

I found that the out-of-the-box algorithms for graph analysis were difficult to interpret in the context of my data set. For example, PageRank runs on all vertices, treating images and people as the same. The resulting ranks are identical for images which are connected to only one person (two if there is an errata), while for people there is a rough correspondence to number of images in the set but a person with one image is given a rank of 1.1 (even though images that have the same number of relationships have a rank of 0.6), whereas Gray Davis with 26 images in the set has a page rank of 14.5. This indicates that people are given slightly more weight than images in the ranking, but is only useful in identifying people with relatively large numbers of images, it can't be filtered based on real values. I was successful however in examining edge counts to identify highly-connected people, in this context a person with a large number of images. The results from this analysis were consistent with those from the structured streaming in identifying the same 5 people with more than 100 images.

## Conclusion

Although the modules of Apache Spark may seem incompatible with image analysis I was able to find useful applications for them in examining the Labeled Faced in the Wild data set. Structured Streaming and GraphFrames are both useful in analyzing metadata about the images, and the Random Forest classifier available through MLlib would provide a good solution in setting up a streaming facial recognition pipeline. Such a pipeline would not be able to support images directly since Structured Streaming does not support image file formats, however using one of the Python libraries available to convert images to arrays of RGB values would be appropriate.

Using the TensorFlow Inception model unfortunately would not be appropriate for a streaming facial recognition pipeline. My results in testing that Inception were consistent with the fact that image classification models that function through pixel analysis have poor performance with facial recognition. Instead a CNN that measures the face and generates embeddings needs to be used, as with OpenFace. Google created FaceNet [9], a facial embedding CNN, using TensorFlow thus the framework is generally appropriate for this type of classification problem but since FaceNet is proprietary creating a facial recognition pipeline using TensorFlow would require designing and training an embedding model rather than using one of the models Google has released to the public.

## References

[1] Huang, Gary B., Manu Ramesh, Tamara Berg, and Erik Learned-Miller. "Labeled faces in the wild: A database for studying face recognition in unconstrained environments." *Vol. 1, no. 2. Technical Report 07-49*, University of Massachusetts, Amherst, 2007.

[2] <https://docs.databricks.com/_static/notebooks/structured-streaming-python.html>

[3] Amos, Brandon, Bartosz Ludwiczuk, and Mahadev Satyanarayanan. "Openface: A general-purpose face recognition library with mobile applications." *CMU School of Computer Science* (2016).

[4] <https://docs.databricks.com/spark/latest/mllib/binary-classification-mllib-pipelines.html>

[5] <https://docs.databricks.com/spark/latest/mllib/decision-trees.html>

[6] <https://www.tensorflow.org/tutorials/image_retraining>

[7] <http://vis-www.cs.umass.edu/lfw/>

[8] <https://docs.databricks.com/spark/latest/graph-analysis/graphframes/user-guide-python.html>

[9] Schroff, Florian, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815-823. 2015.