Building Statistical Models using Random Forest and Regression

Siddhartha Vale, *Author, IEEE* and Sai Prasanth Thota, *Author, IEEE*

***Abstract* – To replicate H2O.ai automl function using Apache Spark and TensorFlow. These functions will include models such as Random Forest, Boosted Trees and Regression methods with ensemble techniques etc. Spark with Databricks IDE is gaining popularity but still has some limitations and lacks popular machine learning techniques. Hence, to overcome these drawbacks we can come up with our own automl function built on spark clusters using TensorFlow.**

I. Introduction

The main concept behind the project is to create a model that the end user can apply to their data. The created model must keep up with future development and be easy to use for the client. Companies that intend to use our model have large datasets, that need high processing powers and a program that can handle the extensive sizes these datasets tend to be. Using an older environment like Jupyter Notebook is not going to be helpful, as it will not be able to keep up with new developments and the size of data the clients have. PyCharm, an advanced IDE within the Python language is also not a recommended environment as it requires experience to use and cannot document the data in an easy to follow flow. Considering these requirements our project was designed on Databricks, a relatively new program that runs large datasets with ease and can documented in an easy format for low experience users.

Applying four models (Logistic, Ridge and Lasso regression & Random Forest) we notice that for a random forest model with 5 and 20 decision trees, a forest with higher trees leads to a more accurate model, however the 20 tree model takes a substantial larger amount of time to process, for this reason five trees is better for clients with less time to run their data. By creating a dictionary with a range of values for maxDepth, maxBins and numTrees (number of trees) along with using cross validation techniques we can run the model to estimate which combination of the three factors lead to the best model parameters, within a given time. The datasets in this instance used were databricks datasets, however, when the client chooses to use another dataset, the key parameters can be easily changed. The end user can also add or remove maxDepth, maxBins and numTrees in order to make the model more suitable to their needs, however

II. Methodology

A. Getting Started with Databricks

The first step to getting started with Databricks is creating an account, what you want to do is to create an account on the community edition, this enables you to all the features as long as you are an eligible user (student, faculty member, employee, etc) for the package. (refer to Figure 1)

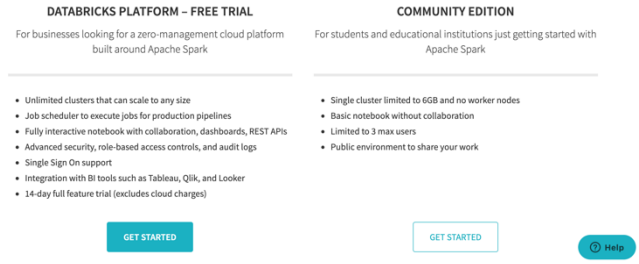

Figure 1: Databricks package to install and signup to

B. Opening .dbc file

First download the .dbc file and locate in destination within the system you use. Next, Go to Home, then workspace, after that click the drop-down button and select import. (refer to Figure 2)

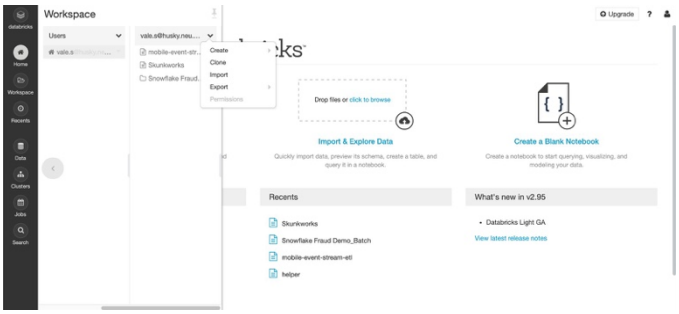

Figure 2: Databricks UI for importing .dbc files

Once that is done, select file, locate the file on your system and click the import button you are given on the Import Notebooks tab. (refer Figure 3 & 4)

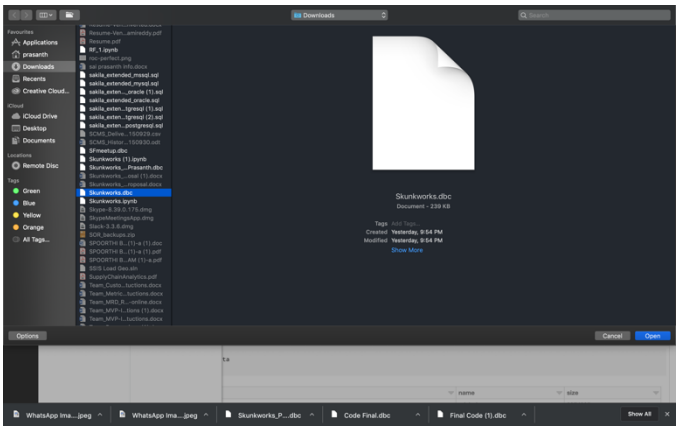

Figure 4: Selecting file on system

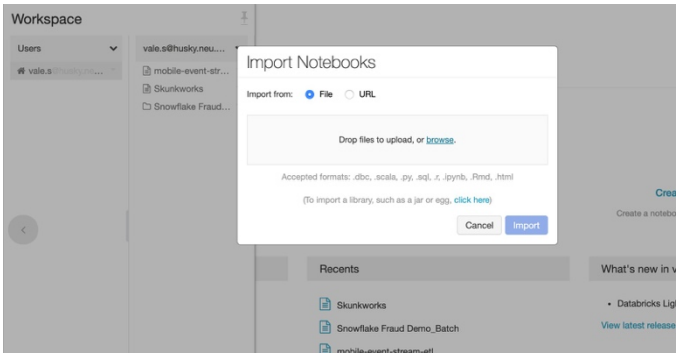

Figure 3: Import Notebook Tab on databricks

C. *Creating a cluster*

In Databricks, clusters are required in order to run the notebook seamlessly and with minimal runtime. This is done in the following steps, first the Clusters tab on the left bar is selected and then a new cluster is created (refer to Figure 5), then you can name the cluster as you wish. In this case it is named Skunkworks, use the same environments show (refer to Figure 6)

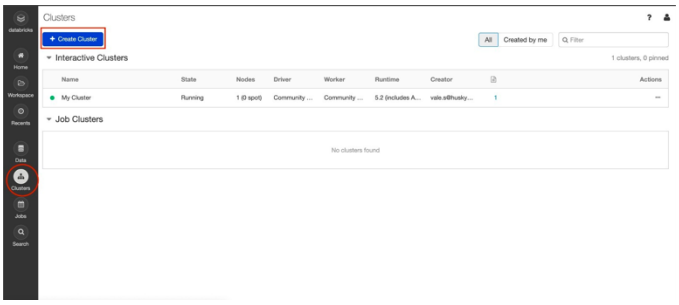

Figure 5: Creating a cluster in databricks

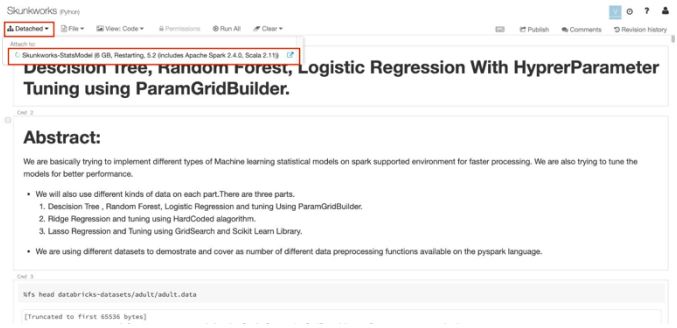

Figure 6: Environment to be used

D. *Running the Model*

This is the final step in getting the model to work on your desired system. Press the button that says Run All and give it the time in requires to complete simulation. (approximately 10 minutes).

Importing Dataset and Data Pre-processing

In this notebook a variety of models were created and run with a number a dataset. To get the exact accuracies we were able to achieve, the following steps must be followed. The first dataset used was about employment, where we are predicting salary based on the individual's marital status, gender, race and educational status. The string values in this dataset, which the prediction is based on were converted to integer values using the OneHotEncoder method. What once it does is, it converts categories into binary vectors with at most one nonzero value, for example (Blue: [1, 0]), (Green: [0, 1]), (Red: [0, 0]). Later using StringIndexer we encode a string column of labels to a column of label indices.

Logistic Regression, Random Forest & ParamGridBuilder

When implementing Logistic Regression with the dataset, we measure the correctness of the values with the ROC (receiver operating characteristic) curve. This shows us the true positive and false positive rates of our dataset values.

After accessing the correctness of the data, we must split it, with a ratio of 70:30 between training and testing respectively. This helps improve the accuracy of our models when running real world tests. For the LR (logistic regression) model we run a K-Fold CV (cross validation) with 5 fold, meaning there are 5 iterations that take the sample data while running the model.

With Decision Tree, we must take an additional step and give the model not only the number of cv folds of 5 but also setting a maxBins and maxDepth. In model design maxBins is the number of bins used when discretizing continuous features, by increasing maxBins the algorithm allows to consider more split candidates and make fine-grained split decisions. However, it also increases computation and communication. While maxDepth is the maximum depth of a tree. Deeper trees are more expressive (potentially allowing higher accuracy), but they are also more costly to train and are more likely to overfit.

With Random Forest, the only difference is the addition of the number of trees, higher the tree count in a random forest the higher the accuracy of you model would be.

Ridge Regression & Functions Search

Using a dataset that measures the price of diamonds based on their cut, clarity, no. of carats and color we use a hardcoded function that separates the prediction accuracy defining values and zip them back together in an H2O.ai method to find the best predictor value (ROC) and the values that lead to the greatest accuracy.

Lasso Regression & GridSearchCV

With Lasso regression, another dataset from the databricks library was used. This along with the GridSearch method to separate test values using a negative mean squared error scoring method lead to a best estimator with an alpha of 1.0000000000000001e-05, copy X = True, fit intercept = True, maxIter =1000, normalize = True, positive = False, precompute = False, random state = None, selection = 'cyclic', tol = 0.0001, warm start = False)

III. Results

A. *Logistic Regression*

After conducting the simulations of models within Logistic Regression, we find that with a maxIter of 10, using the 'label' column and 'feature' column we get the expected accuracy of

Equation 1: Initiating Logistic Regression

```
from pyspark.ml.classification import LogisticRegression

# Create initial LogisticRegression model
lr = LogisticRegression(labelCol="label", featuresCol="features", maxIter=10)

# Train model with Training Data
lrModel = lr.fit(trainingData)
```

Equation 2: Evaluation of LR prediction

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

# Evaluate model
evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
evaluator.evaluate(predictions)
```

```
Out[13]: 0.9032867661805299
```

Equation 3: Best LR Evaluation Metric

```
evaluator.getMetricName()
```

```
Out[14]: 'areaUnderROC'
```

B. *Decision Tree*

After conducting the simulations of models within Decision Tree, we find with various depths and bins the areaUnderROC as the metric to judge accuracy.

Equation 4: Importing Decision Tree library

```
from pyspark.ml.classification import DecisionTreeClassifier

# Create initial Decision Tree Model
dt = DecisionTreeClassifier(labelCol="label", featuresCol="features", maxDepth=3)

# Train model with Training Data
dtModel = dt.fit(trainingData)
```

Equation 5: Node and depth values

```
print("numNodes = ", dtModel.numNodes)
print("depth = ", dtModel.depth)
```

```
numNodes =  11
depth =  3
```

Equation 6: Impurity Type

```
dt.getImpurity()
```

```
Out[29]: 'gini'
```

Equation 7: Decision Tree Results

```
Out[33]:
    areaUnderROC  maxBins  maxDepth
0       0.500000       20         1
1       0.500000       40         1
2       0.500000       80         1
3       0.700164       20         2
4       0.700806       40         2
5       0.700036       80         2
6       0.641860       20         6
7       0.713192       40         6
8       0.701003       80         6
9       0.754657       20        10
10      0.755803       40        10
11      0.780281       80        10
```

C. *Random Forest*

Random Forest is much the same as Decision Tree with one major difference, decision tree uses one tree only, while with random forest we specify the number of trees.

Equation 8: Random Forest accuracies stacked

```
Out[54]:
    areaUnderROC  maxBins  maxDepth  numTrees
0       0.812337       20         2         5
1       0.815858       60         2         5
2       0.860549       20         4         5
3       0.861325       60         4         5
4       0.877924       20         6         5
5       0.878750       60         6         5
6       0.873244       20         2        20
7       0.873812       60         2        20
8       0.886692       20         4        20
9       0.887544       60         4        20
10      0.896270       20         6        20
11      0.898867       60         6        20
```

D. Ridge and Lasso Regression

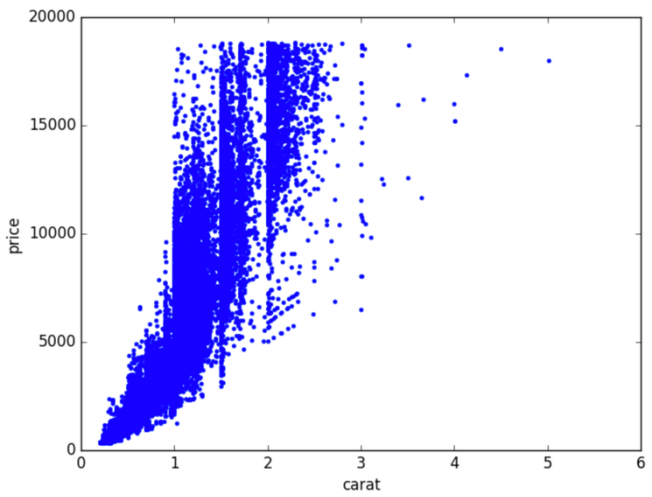

Figure 7: Price vs. Carat for diamond in this Dataset

After training and testing the dataset along with running the data through ridge regression once without tuning the hyper parameters we achieve a low score of 0.557955

Equation 9: Original Score, without hyperparameter tuning

```
# Score the initial model.  It does not do that well.
origScore = origClf.score(trainingFeatures, trainingLabels)
origScore
```

```
Out[65]: 0.55795511154308175
```

Once we conducted the tuning of hyperparameters like, setting number of folds to 3, creating an RDD of tasks, broadcasting the training labels & features and finally applying the training function, we get:

Equation 2: Tuned Parameters for Ridge Regression model

```
# Use bestAlpha, and train a final model.
tunedClf = linear_model.Ridge(alpha=bestAlpha)
tunedClf.fit(trainingFeatures, trainingLabels)
```

```
Out[76]:
Ridge(alpha=0.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
```

Equation 1: Alpha and Train/Test ROC Values

Model	Alpha	Training	Test
Original	0.5	0.557955	0.561176
Tuned	0	0.894043	0.896083

IV. Discussion

Building a set of Statistical Model is important as it makes predicting accuracies for tasks simple. Machine Learning and the use of Spark is at the forefront for this. Spark and TensorFlow are two environments that are at the top of their fields currently and creating models that work within these environments will make working with this document easy for our client and their desired end users. This is where the use of Databricks takes form, Databricks is a notebook documentation IDE that can implement Spark and TensorFlow with the use of Python, Scala, Java or SQL, making the use of this program essential in the success of marketing.

Starting this project, we built a logistic regression model with Spark and TensorFlow libraries and swiftly moved towards Ridge and Lasso regression. Our aim was to build a range of models and test them on a variety of datasets to show its large implementation scope. Working with multiple search functions, we also show how GridSeachCV and ParamGridBuilder operate, leading to different prediction ways.

The work we leave behind for other team to pick up and further grow, is a great base with a variety of models and datasets, implementing the most important models in Machine Learning and applying the best and worst model parameters to show the extensive variation in scores collected.

V. Conclusion

In conclusion, Random Forest and Decision Tree are shown to be the best models for these datasets, giving proof that it would be ideal for most datasets that clients choose to use. Personally, we recommend the use of Random Forest as it increases the chance of improving the accuracy and you can also add or removing trees depending on the time the client wishes to set.

References

1. Apache spark MlLib Methods. https://spark.apache.org/docs/1.2.0/mllib-linear-methods.html
2. https://spark.apache.org/docs/latest/index.html
3. Spark Community. Spark community. https://spark.apache.org/community.html, 2015.
4. Xiangrui Meng, Joseph Bradley, Evan Sparks, and Shivaram Venkataraman. Ml pipelines: A new high-level api for MLlib. https://databricks.com/?p=2473, 2015.
5. MLlib. MLlib user guide.https://spark.apache.org/docs/latest/mllib-guide.html, 2015.
6. NumPy. Numpy. http://www.numpy.org/, 2015. Spark Packages.

7. Spark packages. https://spark-packages.org, 2015.

8. David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. Journal of Machine Learning Research, 3, 2003.

9. Mahout. Apache mahout. http://mahout.apache.org/, 2014.

10. Ameet Talwalkar. BerkeleyX CS190-1x: Scalable machine learning. https://www.edx. org/course/scalable-machine-learning-uc-berkeleyx-cs190-1x, 2015.

11. Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In USENIX Symposium on Networked Systems Design and Implementation, 2012.

12. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. *NSDI 2012*. April 2012. **Best Paper Award**.

13. https://stackoverflow.com/questions/39430998/why-gridsearchcv-return-score-so-different-from-the-score-returned-by-running-mo

14. https://towardsdatascience.com/automated-machine-learning-hyperparameter-tuning-in-python-dfda59b72f8a

SCOPE

All the work was divided between the two authors and completed together, for the common goal of completing with multiple successful models for the client. Here is a table with the weekly deliverables set and achieved:

Week 1	Research about Spark and its functions
Week 2	Implemented Linear regression with EDA
Week 3	1. Implemented tuning on Ridge without using a library. 2. Research about GCP and its usage.
Week 4	1. Build Lasso, Random Forest, Decision Tree and Logistic Regression 2. Tuned the above models using ParamGrid (Spark Function from Tuning) and GridSearchCV using scikit learn library. 3. Used numerous spark based preprocessing functions including EDA.

CONTEXT

1. Grid search: https://stackoverflow.com/questions/39430998/why-gridsearchcv-return-score-so-different-from-the-score-returned-by-running-mo
2. Tuning without the use of library, we have used the Idea nut not the code: https://towardsdatascience.com/automated-machine-learning-hyperparameter-tuning-in-python-dfda59b72f8a
3. For the rest of the code we have mainly referred the documentation and its examples. (refer to REFERENCES)