*Big Data Analytics & Management Final Project*

Matthew S. Hartstein

College of Engineering

University of Arkansas

Fayetteville, AR, U.S.A.

mshartst@uark.edu

*Abstract*—This document is a report that defines the components of my paper in its style sheet. Throughout this report, I will discuss my final project for this class, as well as the results.

Keywords—document, report, components, paper, results

# Introduction

For this assignment, I was instructed to do the following objectives: (1) Brainstorm a research problem, (2) determine at least 2 datasets for my experiments, (3) implement at least 2 machine learning algorithms to analyze my two, selected datasets, (4) comprise the experimental results, then (5) to provide a written document for my report, which must be at least 3 pages long, in IEEE double column format.

# Body

To begin, I needed to formulate a research problem for my two collected datasets. The research problem that I have implemented deals with Linear Regression Machine Learning, as well as Decision Tree Machine Learning. The first dataset that I chose includes and extracts an Excel file that is full of data, then converts it to readable machine language to output two charts: an expected linear regression vs. an actual linear regression. This dataset involves two Columns: ‘Machine Age (Months)’ and ‘Mean Time Between Failure (Days)’. A research topic can be formulated from this data to determine if there is a linear correlation between the age of a machine vs. the mean time failure in order to determine the best line of fit between these two variables.

As for my second dataset, I decided to simply create a sample dataset, or collection, of pre-defined numbers within my source code. I did this for simplicity’s sake because I was having a hard time pre-processing my dataset while implementing my source code for a Decision Tree algorithm. Regardless, I believe this second dataset will suffice because the output still shows how the Decision Tree Algorithm operates and displays a correct output to the console window.

Before moving on to the results section, I wanted to briefly mention that around 10 hours was spent implementing and debugging my source code (for both datasets) to ensure a high-quality outcome that meets the expectations from our problem statement. Dr. Zhan has done an amazing job at preparing myself for this assignment. Overall, I feel like it was a complete success! Now, let us move on to how I implemented my source code, along with each output.

# Implementation

First, we will begin by observing my first program, which exemplifies my knowledge in implementing a Linear Regression Model from any given dataset and/or database.

After opening and observing my source code for my Linear Regression Algorithm Modeling program, you will immediately notice that it is written in the Python programming language. In particular, the following libraries were used to construct my program: ‘tensorflow.campat.v1’, ‘numpy’, ‘pandas’, as well as ‘matplotlib.pyplot’. All of these libraries have very handy built-in functions that simplified the entire implementation process. Next, I instantiate a randomly generated number on line 14. Moving on, I set up the dataset by using the ‘pandas’ library to extract the information from the Excel file and/or dataset. Also, I added two variables (on line 20 and 21) to collect the column names to ensure high quality and efficient work. Moving on, I created Hyperparameters, which define something that goes into a model. Underneath these lines of code, I initialized a single parameter variable titled, ‘display\_step’, which is located on line 28.

Next up, I created two instances for model training: variables ‘train\_X’ and ‘train\_Y’. Underneath this, I determined the length of the dataset by assigning a number to a new variable, ‘n\_samples’. Next, I simply define my placeholders on lines 38 and 39. On lines 41, 42, and 43, I decided to define variables for my linear model. Starting at line 45, I create the actual data model with three – very simple – lines of code. On line 51, I initialize all variables.

Now is the fun part. On lines 53 through 95, I begin the training process by running the initializer, fitting all training data, displaying logs per each step, graphical displays (2 total), a testing example with two, pre-defined arrays, then outputting the results to the console/terminal window screen.

In total, implementing the Linear Regression Modeling Algorithm via Python was fairly straightforward, even though it did take quite some time to get started. Learning about the pre-define libraries was a time-consuming process but in the end, it made my code extremely efficient, clean, organized, and very easy to read and understand. Therefore, for this section, I am very please and satisfied with my overall results, which I will share as screenshots in the following section, along with explanations as to why I received such outputs for each of the algorithms that I implemented: a Linear Regression Machine Learning Algorithm and a Decision Tree ML algorithm.

Moving on to the second algorithm that I decided to implement – which is a Decision Tree Machine Learning Algorithm – I had to perform extensive research to fully-understand what I was expected to accomplish. In the next couple of paragraphs, I will carefully explain what I did to accomplish the second task that is associated with our problem statement.

First, in order to implement a Decision Tree Classifier Algorithm in pure Python, I started by including the proper libraries: ‘from \_\_future\_\_ import print\_functions’. Next, I implemented a training dataset in the form of a simple dictionary. The reason why I decided to do this was because I was having a hard time pre-processing the dataset that I had previously collected on Kaggle.com. Thankfully, this worked which permitted me to move forward.

After creating a personalized dataset/Python dictionary, I collected the headers by assigning ‘color’, ‘diameter’, and ‘label’ into a ‘header’ array. Moving down, I use the ‘unique\_vals’ function to find the unique values for a specific column within the dataset. In the ‘class\_counts’ function, I count the number of each type of example in the dataset. It is important to note that within our dataset format, the label is always the last column! Lastly, the ‘is\_numeric’ function, I simply test to see if a value is numeric.

Now, you will notice that I created a ‘Question’ class to record column numbers and column values. The ‘match’ method is used to compare the feature value in an example to the feature value stored in the question. This class contains three functions that are fairly self-explanatory, therefore, I will exclude them from this report.

Next, we have a ‘partition’ function that partitions a dataset. Specifically, for each row in the dataset, we check to see if it matches the question. If it does, we add it to ‘true rows’. Otherwise, we add it to ‘false rows’. Also, within the ‘gini’ function, I calculate the Gini Impurity for a list of rows. There are many ways to do this, but I found this to be the most efficient method, by far. In the ‘info\_gain’ function, my program calculates the information gain, which is the uncertainty of the starting node, minus the weighted impurity of two child nodes.

Moving on to the ‘find\_best\_split’ function, my program iterates over every feature and/or value and calculates the information gain. Inside this function, I keep track of the best information gain, keep track of the feature and/or value that produced it, as well as the number of columns that are being used. Inside this function, I also implemented source code that tries to split the dataset but if the split is skipped, it is because it does not divide the given dataset. Below this, you will notice that my program will then calculate the information gain from this split. Overall, this function was very tedious, but I am accomplished for finally understanding it and getting it to properly work.

Next, we have a class titled ‘Leaf’, which represents a leaf node, which classifies data. This class holds a dictionary of a class and the number of times it appears in the rows from the training data that reaches this specific leaf.

As for the next class, I declared a class titled ‘Decision\_Node’, which asks a question and holds a reference to the question, as well as the two child nodes. The ‘build\_tree’ function builds the actual tree by trying to partition the dataset on each of the unique attributes, then calculates the information gain, and returns the question that produces the highest gain. As for the base case, there is no further information gain since we can ask no more questions to the Machine Learning program. Therefore, my program will return a leaf. However, if we reach this point, we have found a useful feature and/or value to be partitioned on. After this step, I recursively build the true branch, then I recursively build the false branch, and return a question node; this records the best feature and/or value to ask at this moment in time, as well as the branches to follow, depending on the answer that is given.

Inside the ‘print\_tree’ function, the base case is synonymous to reaching and locating a proper leaf. If so, my program will print the question at this node. This particular function also calls a function recursively on a true branch. Oppositely, a different function is called if this function recursively returns on the false branch. Moving on to the ‘classify’ function, recursive rules heavily applied to this particular method. The base case indicates that we have reached a leaf. The function also indicates and decides whether to follow the true-branch or the false-branch by comparing the features/values that are stored in the node. As a side note, the following function, ‘print\_leaf’, is fairly straight-forward, therefore, I will omit this from my report; this function simply prints the predictions at a leaf.

# Results

Now that I have very carefully explained the implementation process for each of the two Machine Learning Algorithms that I implemented for this assignment, I will now showcase several screenshots that signify my knowledge in developing and implementing several Machine Learning Algorithms. The first set of screenshots will correlate with the first Machine Learning Algorithm that I implemented (i.e. Linear Regression), and the second set of screenshots will correlate with the second Machine Learning Algorithm that I implemented for this particular assignment.

A screenshot of a cell phone

Description automatically generated

**Figure #1: Expected Regression Graph from Data**

After observing **Figure #1**, you will notice a Line-of-Best-Fit that is projected in a downward (negative) slope. Therefore, this graph indicates that the older a machine is, the less likely it is to work in the future! This prediction is fairly common-sense but was interesting to apply to this particular assignment.

A screenshot of a cell phone

Description automatically generated

**Figure #2: Tested Regression Graph Results**

In comparison to **Figure #1**, this new graph (**Figure #2**) depicts the tested results from my program. This graph also has a negative slope, which indicates that the older a machine is, the more likely it is to break down and/or malfunction. Again, this is fairly common-sense but was used synonymously with **Figure #1** to showcase that my test results were adequate.

A screenshot of text

Description automatically generated

**Figure #3: Linear Regression Model Console Output**

After observing **Figure #3**, you will be able to tell what my program is doing prior to creating both of the aforementioned graphs. Epoch’s are increased by 50 per each iteration, along with error percentages, weights, and a bias for each of the Epochs that are running. Eventually, my program will stop at Epoch number 1000 and returns a Training Error number of ~2.5, a weight of ~-0.46, and a bias of ~24.7. In total, the testing error value returns ~1.62 and the absolute mean square loss difference returns a value of 0.8870877. Overall, this program executed in approximately 36.47 seconds.

A close up of text on a black background

Description automatically generated

**Figure #4: Custom Decision Tree Dataset**

As previously mentioned, I had a very difficult time pre-processing the dataset that I had planned to use for this assignment, which was located on Kaggle.com. Therefore, I found a workaround by simply creating a Python dictionary. The overall goal of this Decision Tree Machine Learning Algorithm is to determine what ‘label’ should output based on it is diameter size, as well as the color of the object. Therefore, after observing **Figure #4**, you will notice that I have created a dictionary of five fruits, each with a corresponding color and diameter size. This information will be used in the Decision Tree Learning Algorithm that I have successfully implemented.

A screenshot of a cell phone

Description automatically generated

**Figure #5: Decision Tree Algorithm Output/Results**

After observing **Figure #5**, you will notice my program asks if the diameter is >= 3, which it is. Next, it asks if the color is yellow, which it is. Therefore, it predicts a Lemon! If the color is not yellow, my program predicts an Apple! Going back to diameter size, if the size is not greater than or equal to 3, my program correctly predicts a Grape! Underneath this section, you will notice the actual results with associated percentages that represent the accuracy of the decisions that have been made. Overall, this was a complete success!

# Conclusions

Now that I have carefully explained the two datasets that were used for this assignment, I hope that it is evident that I understand what Machine Learning algorithms are, how they can be utilized on various datasets, and how to draw conclusions from the expected vs. actual results. Overall, machine learning, deep learning, and artificial intelligence will be the way of the future. This class was a great experience and Dr. Zhan did an amazing job with his lectures. Great class!

Authors and Affiliations

**The template is designed for, but not limited to, six authors.** A minimum of one author is required for all conference articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

1. 68–73.
2. I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–35.

**IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove template text from your paper may result in your paper not being published.**