**Zoo Animal Classification**

**Steven Centeno, Richard Morrow**

**LIS4761 - Data Mining and Analytics**

**Professor He**

**Introduction**

The core problem facing this research project was learning how to “use machine learning methods to correctly classify animals based upon attributes” [1]. Researchers, such as ourselves, must learn to adapt to new research environments. At the beginning of the semester, we were taught the names and uses of a handful of algorithms used for machine learning. We learned, either in class or through research for this project, about Naive Bayesian Classification, Random Forest Regression, Logistic Regression, J48 Classification, and a few more machine learning algorithms.

For the first half of the project, we relied heavily on a program called Orange. This program had a quite easy-to-use user interface which allowed us to easily implement the machine learning algorithms that we had chosen to use. The algorithms that we had chosen to use were random forest regression, naive bayes classification, and logistic regression. Orange simplified the machine learning process so that we were able to utilize these three algorithms and obtain results within just about a day after installing the software itself. This let us have more time to analyze our results and to prepare for our mid-term presentation.

Researchers, such as ourselves, must learn to adapt to new research environments. This sentence was stated earlier, and it was already explained for the use of basic machine learning algorithms in Orange. Next, we were really pushed to adapt to a new problem. This new problem being to learn how to fully utilize Jupyter Notebook. After our midterm project presentation, we were urged to learn how to use Jupyter Notebook, which meant learning an entirely new programming language. Jupyter Notebook cannot be fully utilized as a platform without the use of some compatible languages of code. Jupyter Notebook shares compatibility with many programming languages, such as R and Python, and for this research project, we decided that Python would be most suitable to our needs. In order to utilize Jupyter Notebook to the fullest, we had to put all of the programming languages we are somewhat familiar with, such as C++, Java, Javascript, HTML, etc…, to the back of our minds. This is because Python is quite different these other languages.

The biggest difference between Python and any other object oriented programming language, for instance Java, is that Python is a dynamically typed language, whereas Java is a statically typed language. “This makes Python very easy to write and not too bad to read, but difficult to analyze” [2]. Lucky for us, the easy to write part proved true, but unluckily, however, the difficult to analyze did as well. Analyzing similar code in order to learn how to implement it into our specific code and data set proved to be the most difficult part of the Jupyter Notebook side of the project. Nevertheless, we were still able to make some breakthroughs and eventually were even able to not only properly run the algorithms but also visualize them in graphs created in the Notebook.

**Datasets**

The dataset that we used for this research project was a dataset called “Zoo Animal Classification” from the data science website, Kaggle. Kaggle is a platform for the world’s largest community of data scientists and machine learning engineers. The dataset we used was created by the machine learning team at University of California, Irvine. This same team is responsible for other datasets that were potential choices for the project, such as the Red Wine Quality dataset and the Horse Colic dataset. Clearly, Kaggle, and many of its users, are reputable sources from which we can trust the data we used.

The data is split into two sets, a zoo.csv file and a class.csv file. For the purpose of our research, we only needed to use the zoo.csv file, which included all of the animals’ traits, names, IDs, and most importantly, their class type. Luckily, we did not have to transform the data because both of our platforms, Orange and Jupyter Notebook (Python 3), are able to read directly from these file types.

**Tools Used**

The first tool that we had to download was Orange. Orange has a friendly user interface that makes it a good option for beginner data scientists such as ourselves. It took us only a few hours before we were able to figure out by ourselves how to load the data from the .csv file, run the data through our three chosen prediction algorithms of logistic regression, random forest regression, and naive bayes classification, and see some results. We then connected the test and results to a confusion matrix to better visualize our results. That is essentially the extent to which we used Orange.

The second item that we had to download, which actually includes the other tools we used in the project, was the Anaconda Distribution, “which includes Python, the Jupyter Notebook, and other commonly used packages for scientific computing and data science” [3]. These two tools, Python and Jupyter Notebook, were essential for getting our project done. Once the Anaconda Distribution was downloaded, one simply needs to run the command “jupyter notebook” in terminal (MacOS) or command prompt (Windows) to invoke the Notebook. It opens a new tab in your default browser, from which you can choose “New” then “Python 3” which will simply open a new blank notebook ready for input from Python code version 3. From there, we imported many other useful packages used for scientific computing, such as the numpy, pandas, or sklearn packages, which not only allowed us to actually import and read our .csv files but also to build our prediction models.

We also used Google Slides and Google Docs to create both our midterm and final presentations, as well as our weekly project reports. This web application allowed us to make edits on the same document simultaneously, and eliminate the need to meet up in person to work on assignments.

**Data Acquisition**

During the start of the semester, one of our first assignments was to create and submit a project plan to Professor He. This project plan consisted of the names of our group members and what data set we were going to use for the project. This data set had to be approved by the professor, but to make it easier for the students, Professor He created a list of pre-approved data sets that we could choose from. Out of the options listed, our group decided to use the Zoo Animal Classification data set. This data set was downloaded from Kaggle, the website we were instructed to use by the professor, which is also cited in our sources.

Since we were given the option to downloaded a data set created by someone else, our group did not have to go out and collect the data ourselves. In fact, none of the data included in our project is our own. The information included in our project was entirely gathered by the team who created the data set in the first place.

**Data Preprocessing**

Data was mostly already processed for us by the UCI Machine Learning team. In Orange, loading data from the .csv file was a breeze. After that, all we needed to do was set a target variable, in our case class\_type, and then run the data in our prediction algorithms. Data such as the names of the animals was set to “meta” because it is not pertinent to the prediction variables. In fact, the animal names are what constituted one of the few data preprocessing changes we had to make in Jupyter Notebook. However, before we could make those changes, we had to learn how to simply load data into the Notebook from the .csv file using Python.

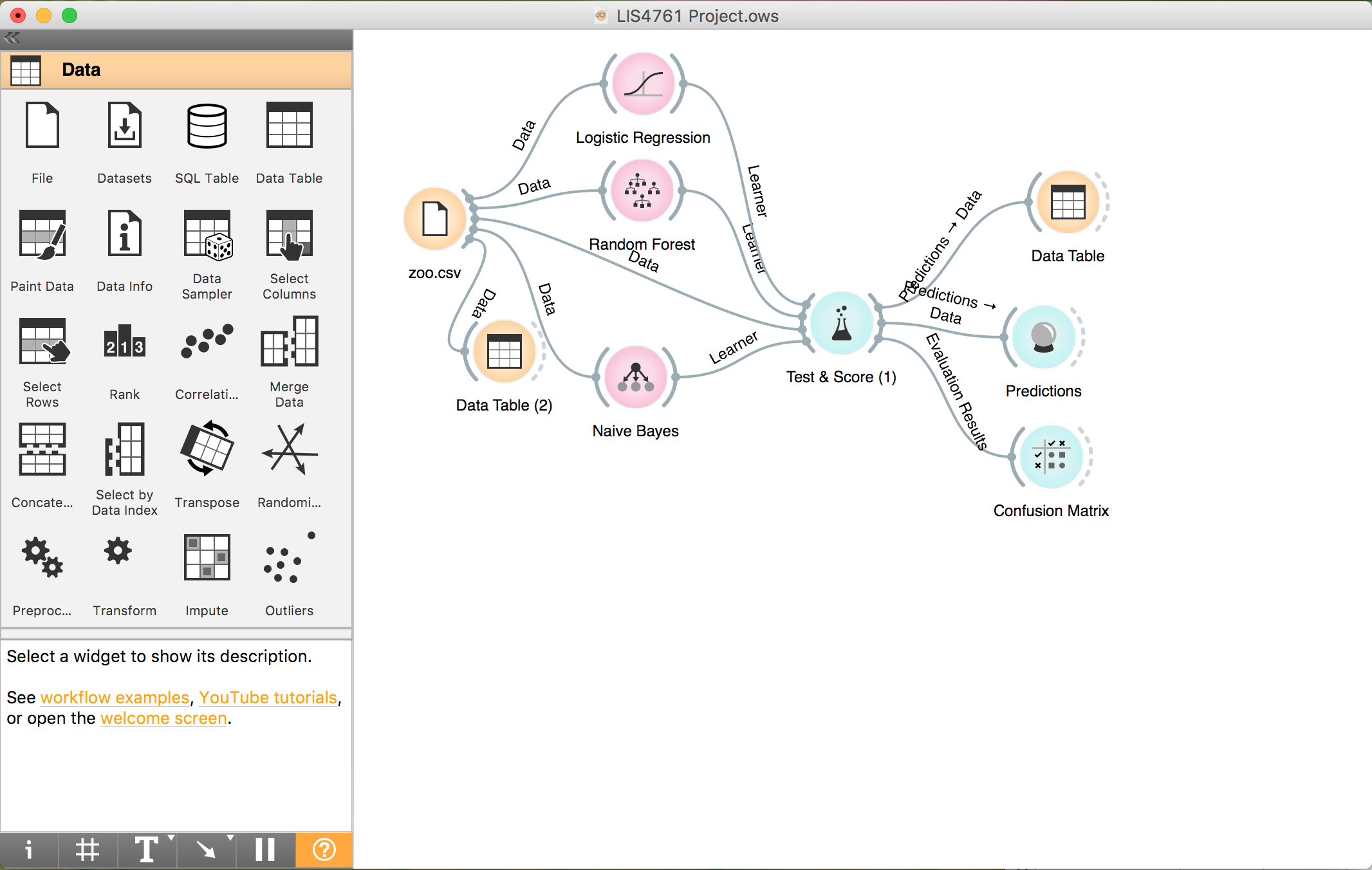
Using Python to import a local .csv file into the notebook ended up being a little trickier than originally suspected, and much trickier than in Orange. Many of the tutorials that we found online were using the Notebook in Windows, therefore the file path is a little easier to follow, and you can just start from the local (C:) drive. Of course, this is not quite how things work using a MacOS, so we had already hit our first roadblock. From past experience in coding Java in cloud9, we knew that we needed to find out what our current working directory was. In order to learn the current working directory in Python, you use the command !pwd. The results we received from this command allowed us to continue the file path all the way to the zoo.csv file, which is saved in a handful of subfolders on our desktops. We imported the pandas package, which allows us to read .csv files in Jupyter Notebook, and save it to use a function named pd. We set a variable, originally called df1 and later switched to zoo1, equal to pd.read\_csv (\*newly learned file path\*). We finally have our data loaded into Jupyter Notebook. Now, what to do with it?

The first thing we decided to do was remove the animal\_name attribute, as each animal was already assigned a unique ID and we were searching for class\_type, not the specific animal, therefore we decided that it was irrelevant data in our zoo.csv file. After dropping the animal\_name attribute, we had to assign feature and target variables. We assigned a variable “target” equal to zoo[‘class\_type’]. Then we assigned each individual feature to their own variable. We then assigned each of those feature variables into one single “features” variable. Finally, we are ready to create a training test group.

Next, we split our data into a training group used to create our prediction models. After this, we set our new training data to fit our predictive models and then used the prediction algorithms to predict training data based on our actual data from the zoo.csv file. We then compared the predicted classes to the actual classes to see how often our prediction models correctly predicted the class of an animal.

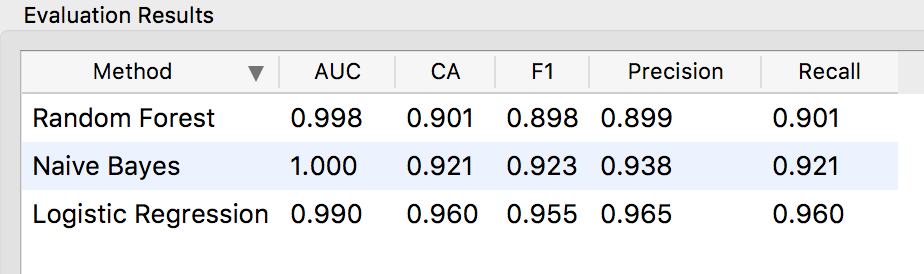
**Data Analysis and Results**

In orange, the setup for utilizing our predictive models was relatively straight-forward. They have many different options for loading in your data, building predictive models, visualizations, and more.

Figure 1: Setup in Orange****

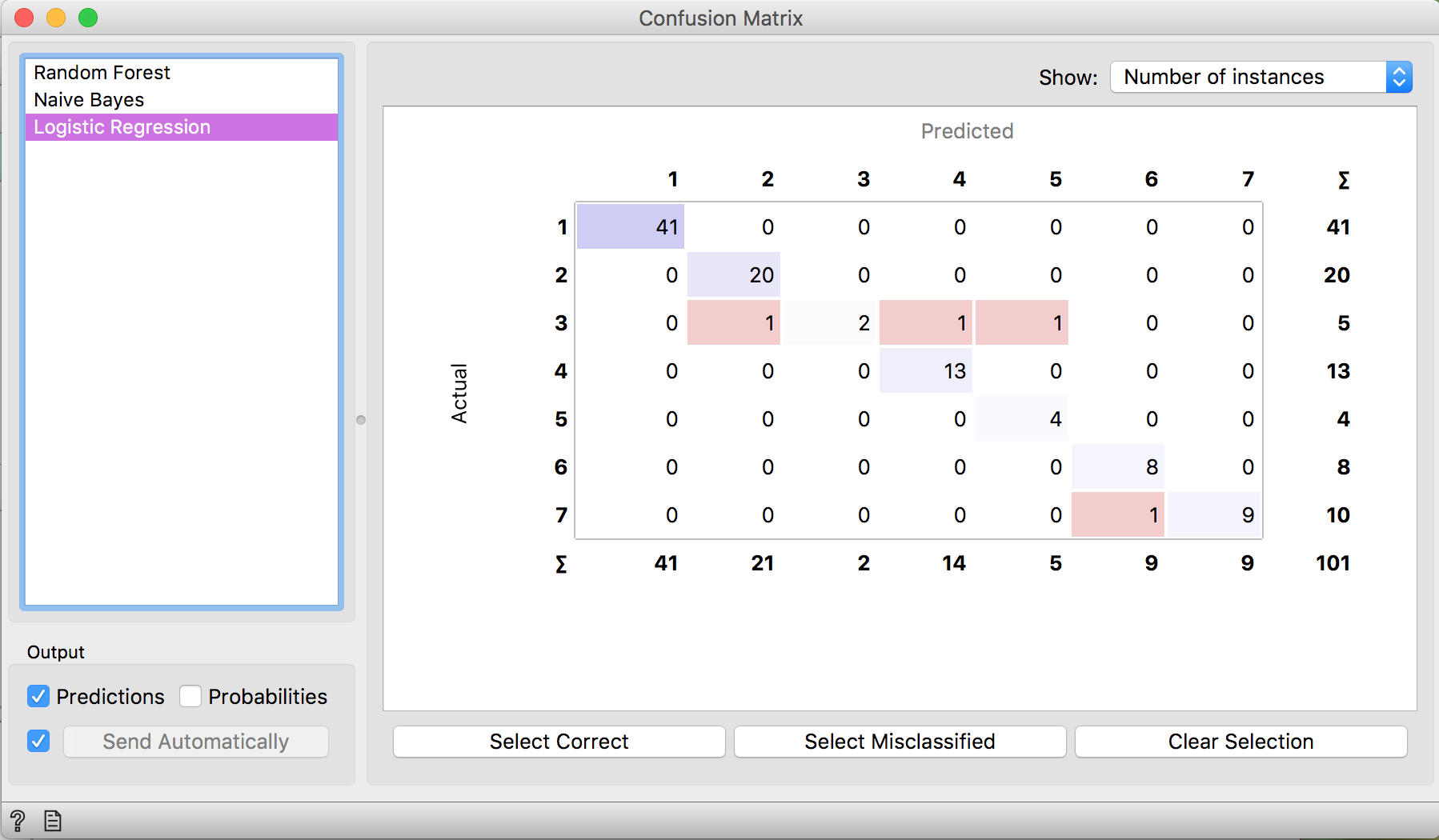
In the figure above (Figure 1), you can see that everything stems from the zoo.csv file. In the properties of this file are listed all of the animal attributes, as well as their class type. We simply had to set the target variable to class\_type and ensure that all of the other variables were set to either feature or meta. The only instance where we used meta was for our animal names attribute, which we found to be unnecessary in determining the class type of an animal. Next we connect that file to our three prediction algorithms, as well as the “test & score” evaluator which needs the original data in order to score the predictions. We then connect our three prediction algorithms to test & score which is finally where we see Table 1 come from. We extended test & score to a confusion matrix in order to better visualize our results. An example of this being useful, in regards to logistic regression, is shown with Image 1.

Table 1. Results of Predictive Models in Orange



The table above (Table 1) shows the accuracy of each prediction model in Orange.

Image 1. Correct and Incorrect Predictions



As seen in the results table above, Logistic Regression did the best job at accurately predicting the class of an animal. However, after running continuous iterations and never achieving perfect accuracy, our group began to discuss potential reasons as to why this accuracy cannot be reached. After looking into the test results further, we realized that the prediction models were mistakenly classifying animals into classes that have similar characteristics as others. These similarities between animal species seem to be the only thing causing the prediction models to make mistakes in Orange.

As for using Python in Jupyter Notebook, after fitting our data for each of our prediction models, we simply had to use a function that would determine how our predicted values stacked up against our actual values. We learned later that it was not a single function that could do this, but two separate functions. This is because one function we used, accuracy\_score, can only be used as a classification metric, and therefore cannot be used to grade regression models, which two of our models are. For Naive Bayes Classification, we were able to determine the accuracy of the predicted data using the accuracy\_score function. For our regression models of random forest and logistic regression, however, we needed to use another function to determine how their predicted values stacked up against the actual values. Here is where we turn to , or R squared, which is the coefficient of determination. This is essentially a way of seeing the percentage of predicted points that were correct compared to our actual data. If a regression line had an score of 1, that would mean that it was perfectly aligned with the data.

Table 2. Results of Predictive Models in Jupyter Notebook with Python

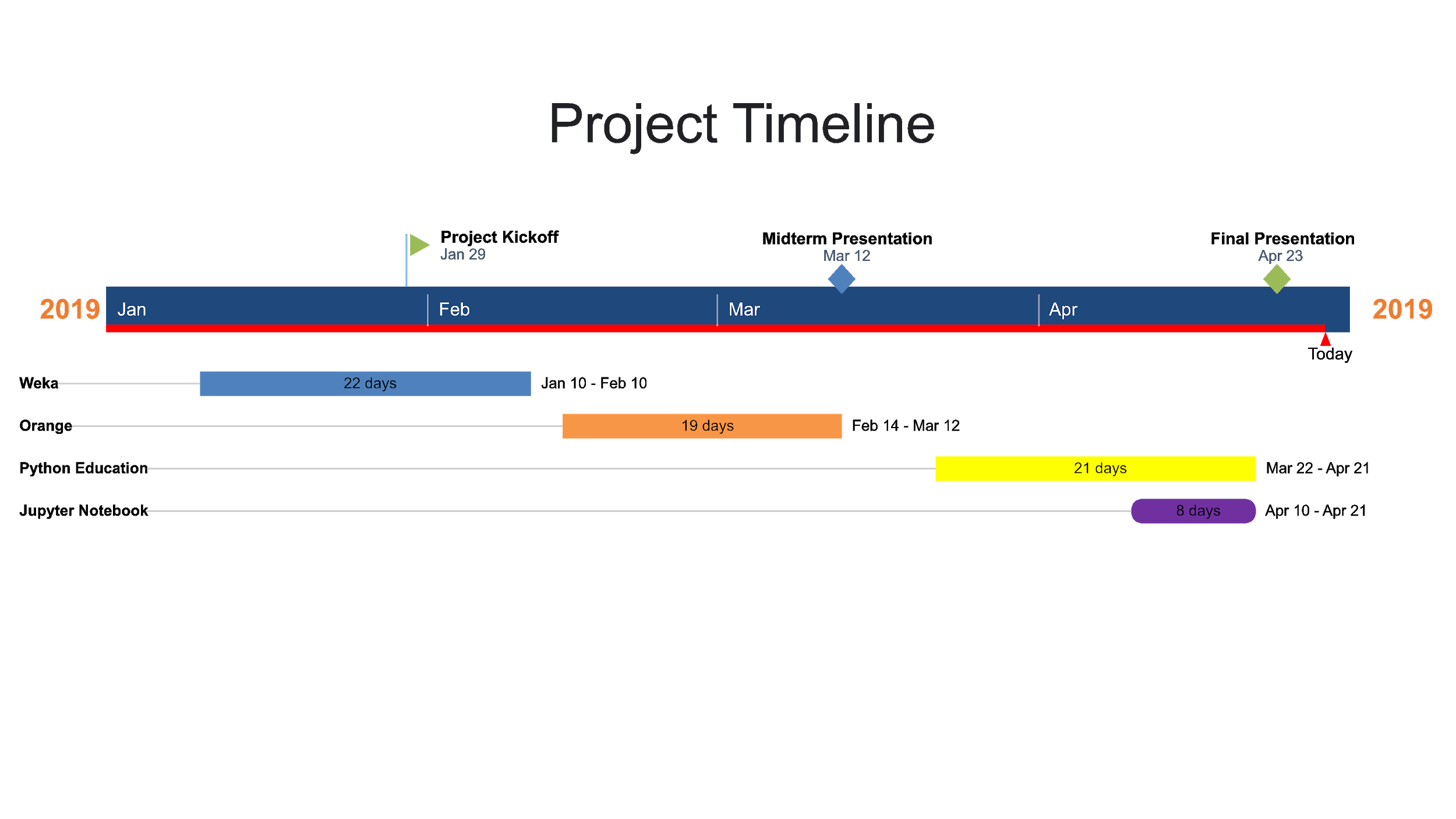
|  |  |
| --- | --- |
| **Prediction Model** | **Percentage of Predictions That Were Correct** |
| Random Forest | 0.998439365 ( Score) |
| Naive Bayes | 0.970588235 (Accuracy Score) |
| Logistic Regression | 0.965423728 ( Score) |

As one can see from the table above (Table 2), clearly all of our prediction models have an even higher than acceptable accuracy. Acceptable in these circumstances would need to be at least 85% accuracy, but our lowest result did not drop below 96.5%. As labeled, random forest regression and logistic regression were scored using the function, whereas naive bayes was scored using the accuracy\_score function. Clearly though, in this instance, the best of these three predictive models was random forest regression. With an score of 0.998, a mere 0.002 away from 1, which would indicate an exact match with the actual data, random forest regression was easily the most accurate prediction model of the three we chose to use to predict our class types.

In conclusion, using Orange, we see that Logistic Regression is the best at being able to accurately predict our target variable given the animal features. Using Jupyter Notebook/Python, however, it would appear that Random Forest Regression can get as accurate as 99.8%. This could just be a random occurrence, as we were only able to create one prediction instance in each of the testing environments (Orange, Jupyter Notebook). The next time we run a prediction algorithm, and in turn create a new prediction instance, it could come back with lower/higher accuracy. In order to improve accuracy of our results we would need to duplicate the data into hundreds, even thousands of training sets. We would find the accuracy of each individual training set and then find the average of those accuracies. This average would be a better representation of the accuracy of the prediction algorithm.

Finally, when using all three prediction models, in both Orange and Jupyter Notebook we were able to see 90%+ accuracy when predicting the class of an animal based off of their traits. That being said, there are some instances where the models incorrectly predict the class of an animal. This happens when the animal shares traits with a class it is not actually in. To wrap up, Jupyter Notebook is an excellent platform for machine learning specialist but requires much more extensive knowledge (especially with Python) than Orange, which is extremely user-friendly and easy-to-use

**Project Timeline:**



We began working on our project after submitting our project plan on January 29th. Our first lesson on how data mining and machine learning works was the introduction of classifiers. Shortly after using weka for basic class assignments we moved onto Orange. Most of the content of our midterm presentation was on the accuracy we were able to achieve using classifiers in Orange. For our final presentation, we were challenged to do the same thing in Jupyter Notebook using Python code. We began educating ourselves in Python code using internet resources. The almighty google was our good friend on this project for learning resources. About a week before the final presentation, we began working in Jupyter Notebook to create predictive models. About two days before the presentation, we created a powerpoint using Google slides.

**Team Workload and Roles**

Our group split the work and learning this project required fifty-fifty. Each week we both worked on project reports, worked with Weka, Orange, or Jupyter Notebook, and kept up with our Python education. We decided this way would be best so that we were always on the same page as one another and could turn to each other as resources for obstacles. We turned out to be a very cohesive team and helped one another solve problems that held us up. Before big presentations such as the midterm and final presentations, we would meet at Richard’s house to go over things in person and predetermine which parts of the presentation we felt more confident in speaking about.

**References**

1. Zoo Animal Classification. (2019). Retrieved from https://www.kaggle.com/uciml/zoo-animal-classification.

2. Radcliffe, T. (2019). Python vs. Java: Duck Typing, Parsing on Whitespace and Other Cool Differences | ActiveState. Retrieved from <https://www.activestate.com/blog/python-vs-java-duck-typing-parsing-whitespace-and-other-cool-differences/>.

3. Project Jupyter. (2019). Retrieved from https://jupyter.org/install.html

Morales, M. (2019). How To Build a Machine Learning Classifier in Python with Scikit-learn | DigitalOcean. Retrieved from https://www.digitalocean.com/community/tutorials/how-to-build-a-machine-learning-classifier-in-python-with-scikit-learn

Koehrson, W. (2017). Random Forest in Python. Retrieved from https://towardsdatascience.com/random-forest-in-python-24d0893d51c0

Barn, B. (2016). Python/Scikit-Learn - Can't handle mix of multiclass and continuous. Retrieved from https://stackoverflow.com/questions/37367405/python-scikit-learn-cant-handle-mix-of-multiclass-and-continuous

sklearn.metrics.r2\_score — scikit-learn 0.20.3 documentation. (2018). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2\_score.html

sklearn.naive\_bayes.GaussianNB — scikit-learn 0.20.3 documentation. (2018). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.naive\_bayes.GaussianNB.html

sklearn.linear\_model.LogisticRegression — scikit-learn 0.20.3 documentation. (2018). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.LogisticRegression.html