Prateek Prakash

EZ7543

11 December 2015

**CSC 5991 – Final Project Report**

1. **OBJECTIVE**

The objective of this project was to be able to create a classifier that is successful in classifying unknown fossil data in the proper taxonomy. The classifier should implement machine learning methods and should be implemented in Python. The objective is to try out at least two different machine learning methods to implement two different classifiers, and then compare the results.

1. **DATASET DESCRIPTION**

Our dataset was composed to two separate sets. Our first set of data was our training data. This will be used to train each classifier respectively. Our second set of data was our testing data. This will be used to test the accuracy of each classifier.

* 1. **Training Data**

The training data consists of text data. Data for fossils have been collected into a reputable database, and this is the data I will be using. The data consists classification such as family, suborder, and etcetera. The data will also have descriptions for each type of fossil. The descriptions, and titles such as family and suborder shall be used to train our classifier. The classifier should be able to take descriptions of fossils and output their classification.

* 1. **Testing Data**

The testing data was gathered. It was gathered from publicly available sets online. The most used source was MicroWorld. Descriptions of different fossils was gather and put into a text file. These descriptions were classified through the both classifiers.

1. **METHODS (MODELS)**

The two models that I found are part of in SciKit-Learn. The first one is the “Decision Tree Classifier”. The second one is the “Multinomial Naïve Bayes”. Both classifiers take in species and descriptions as their input, and they will try to predict the family that they belong to.

* 1. **“Decision Tree Classifier”**

Reference: <http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier>

Source: <https://github.com/scikit-learn/scikit-learn/blob/a95203b/sklearn/tree/tree.py#L401>

* 1. **“Multinomial Naïve Bayes”**

Reference: <http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB>

Source: <https://github.com/scikit-learn/scikit-learn/blob/a95203b/sklearn/naive_bayes.py#L516>

1. **PYTHON CODE**

This section contains all the Python code that was used during the project. They contain code given to use which I then had to modify for my use and code that was completely created by me. Descriptions for what each Python code does is found in the following subsections. These will be referenced later in Section 5 of this document.

* 1. **Identify.py**

The core code here was given to us by Zeyad. I had to edit it a little for my purposes. The purpose of this code was to output Orders, Suborders, Families, Species, and etcetera into a text file. Using the delimiter “##” I was able to change the code so it creates a CSV file that could be opened in Excel. I also made sure it extracted the descriptions for each classification as well.

* 1. **Noise.py**

This code was also given to us by Zeyad. There was no editing done to this one. The code essential removes all characters other than alphabets, spaces, newline characters, and tab characters. This was used to clean the data before being run through TF-IDF and the classifiers.

* 1. **TF-IDF**

This code is self-explanatory. It essentially was a test code to implement TF-IDF on the train and test data to get a matrix. It is just the basic TF-IDF code, and nothing more.

* 1. **TF-IDF-DTC**

This code include the TF-IDF from Section 4.3 as well as code to merge the train and test matrix into a bigger, singular matrix. From there it runs the Decision Tree Classifier and runs some statistical methods on the results.

* 1. **TF-IDF-MNB**

This code include the TF-IDF from Section 4.3 as well as code to merge the train and test matrix into a bigger, singular matrix. From there it runs the Multinomial Naïve Bayes Classifier and runs some statistical methods on the results.

1. **STEPS**

This section contains all the steps I took in order to complete this project. This includes any mistakes, redoes, and etcetera. Overall there are 9 sub sections within this section. Each section contains descriptions of what occur during that step. Some of them reference Python code that was used which is referenced in Section 4 of this document.

* 1. **Extract Into CSV**

So the first step was to get the RAW data into some sort of readable and useable format. The decision was made to turn it into a CSV file. Using the Identify.py (4.1) code, I was able to extract each taxonomy and description on individual lines. Then using the “##” delimiter in the code, I was able to set it up as a CSV table which could be opened up in Excel to read in a nice manner.

* 1. **Condense To 3 Column Format**

When using the data, I didn’t need all the information. I just needed Family names, Species names, and Species descriptions. So in order to change this, I just opened the previous file in excel and deleted all other columns.

* 1. **Condense To 2 Column Format**

After discussing with Zeyad, the 3 column format wasn’t working, so we switched to a 2 column format where Families were in one column, and Species and the Species descriptions were merged in the next column. In addition, I combined all Species belonging to a single Family in one line. So essentially, a line would have: Family – All Species and All Descriptions.

* 1. **Clean All Data**

The next step was to clean up the data to how we needed. For this, I just ran the Noise.py (4.2) on the test data set.

* 1. **Create TF-IDF Code**

This was just the creation and running of the TF-IDF code. The output was a matrix and I output it into a file just to make sure things were calculated correctly and for my own reference. This was the code in TF-IDF.py (4.3). The TF-IDF was run on the Family and Species and Species descriptions.

* 1. **Gather Test Data**

After this, I began to collect my testing data. I looked up Species from the training data online. My main source was MicroWorld. Here I found man Species I needed as well as good descriptions of the Species. I then formatted a table similar to the 2 column format mentioned above (5.3).

* 1. **Change Data Format**

At this point, I created the code for the Decision Tree Classifier. This was the TF-IDF-DTC.py (4.4). However when I ran the classifier, I got a result of 0.0. This was not ideal. After looking back, I decided to change up the format of the data set. I reverted to a 2 column format. However, I didn’t group all Species belonging to 1 Family together. This time, every Species would have its own line, thus Families would be repeated. Next I split the Family and Species and Species descriptions into two separate files. One file contained a column of just Families while the other file contained the actual data. At this point, I also combined all 4 of my data sets, into one large one.

* 1. **Clean All Data**

After reformatting my data, I once again had to clean it. This time however, I had 4 files. 1 Test Family, 1 Test Data, 1 Train Family, 1 Train Data. I cleaned each one separately and proceeded.

* 1. **Apply Classifiers**

The final part was to create the code for the classifiers. I already had the Decision Tree Classifier code ready, and I just copied that code and replaced the classifier with the Multinomial Naïve Bayes, which is the code TF-IDF-MNB.py (4.5). After this I also included some metrics methods from SciKit Learn in order to obtain some statistics about the performance of both classifiers and to allow quantitative comparison of the two.

1. **RESULTS**

Below are the results I got from my final run of the data through both classifiers. Each one has the Score method applied within each classifier. Then I ran the “precision\_recall\_fscore\_support” method from SciKit Learn Metrics. The results are below…

* 1. **Decision Tree Classifier Results**

*Score:*  0.0337552742616

*Precision:* 0.04540493857467176

*Recall:* 0.033755274261603373

*F-Score:* 0.028333067709387018

* 1. **Multinomial Naïve Bayes Results**

*Score:* 0.111814345992

*Precision:* 0.023615698147621132

*Recall:* 0.11181434599156118

*F-Score:* 0.036131969772819371

1. **CONCLUSION**

Overall, the project was very insightful in terms of data mining. Learned some valuable skills through this and knowledge. As far as which classifier was better. In my case, the Multinomial Naïve Bayes produced significantly better results. This might just be that the Decision Tree Classifier was not the best model for this type of textual data. Or it might just mean the Multinomial Naïve Bayes just was a better model. As for getting a result of 0.0 the first try, I believe that was due to the format of the data itself and the fact that TF-IDF was run on the labels as well as the train and test data. This might have thrown off the classifier. However after splitting the labels and data, I got better results and things worked out.