Phase 3 Final Deliverable

A description of the web crawler algorithm:

The web, a large directed graph of documents and resources, can be autonomously crawled. Each document can be seen as a node, and each link that connected the documents are the edges of the graph. Web crawlers must make heavy use of various algorithms to be both efficient and effective in tagging web documents. A crawler is able to get caught in loops, however the designers of the crawler should be able to limit the search space of each child node (say to maybe 5 links) in order to avoid an infinite cycle.

One major efficiency fix includes implementing multi-threading, this way the crawler does not need to traverse only one node at a time. One thread can now traverse one direction of the graph, while others choose other traversal paths. Other issues that can arise while crawling the mostly unstructured content of web documents include reaching non-existent documents, server timeouts, and restricted content.

There is unfortunately not much that can be done in these situations, the pages just need to be pruned from the graph and possibly attempted to be re-index at a later point in time.

A description of the J48 Algorithm:

J48 is an open source Java implementation of the C4.5 algorithm used in the Weka data mining tool. It is a decision tree learning algorithm that builds models of classes from a set of records that contain class labels. Another use for this algorithm is to find out the way the attributes-vector behaves for a number of instances. Also on the bases of the training instances the classes for the newly generated instances are being found.

J48 is an extension of ID3. The additional features of J48 are accounting for missing values, decision trees pruning, continuous attribute value ranges, derivation of rules, etc. The WEKA tool provides a number of options associated with tree pruning. In case of potential over fitting pruning can be used as a tool for précising. In other algorithms, the classification is performed recursively till every single leaf node is pure, that is the classification of the data should be as perfect as possible. This algorithm it generates the rules from which particular identity of that data is generated.

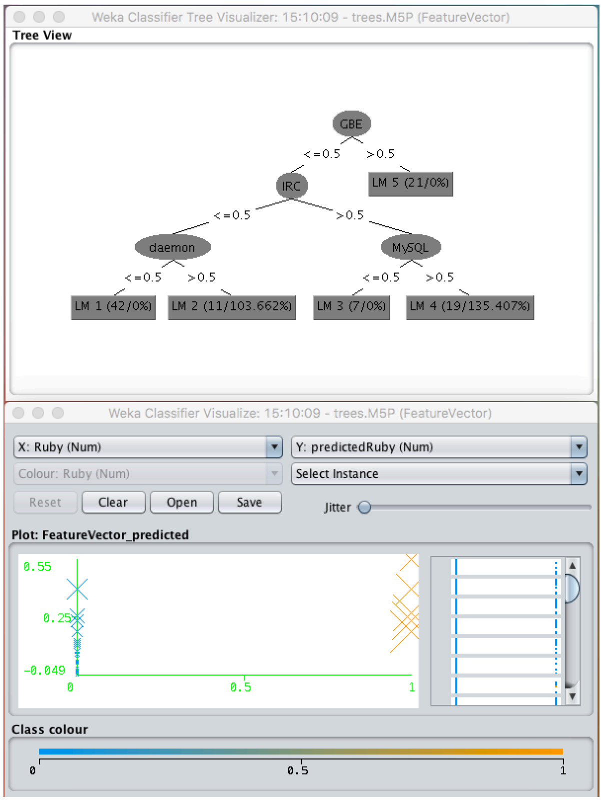
The classification is performed on the instances of the training set and tree is formed. The pruning is performed for decreasing classification errors which are being produced by specialization in the training set. Pruning is performed for the generalization of the tree.

That being said, for the purpose of this project, we used the M5P tree algorithm since J48 was not available (greyed out).

Report on Weka Data:

Classifier output from the original feature vector set [M5P used since J48 does not show up]: SEE NEXT PAGE

```
=== Run information ===

Scheme:       weka.classifiers.trees.M5P -M 4.0
Relation:     FeatureVector
Instances:    100
Attributes:   100
              [list of attributes omitted]
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

M5 pruned model tree:
(using smoothed linear models)

GBE <= 0.5 :
|   IRC <= 0.5 :
|   |   daemon <= 0.5 : LM1 (42/0%)
|   |   daemon >  0.5 : LM2 (11/103.662%)
|   IRC >  0.5 :
|   |   MySQL <= 0.5 : LM3 (7/0%)
|   |   MySQL >  0.5 : LM4 (19/135.407%)
GBE >  0.5 : LM5 (21/0%)

LM num: 1
Ruby =
           0.0272 * daemon
           - 0.0491 * x86
           - 0.0143 * GBE
           + 0.0603 * IRC
           + 0.0181

LM num: 2
Ruby =
           0.0597 * daemon
           - 0.1642 * x86
           - 0.0143 * GBE
           + 0.0603 * IRC
           + 0.1132

LM num: 3
Ruby =
           -0.0928 * daemon
           - 0.0417 * x86
           - 0.0143 * GBE
           + 0.1164 * MySQL
           + 0.084 * IRC
           - 0.155 * user
           + 0.1716

LM num: 4
Ruby =
           -0.1586 * daemon
           - 0.0417 * x86
           - 0.0143 * GBE
           + 0.0753 * MySQL
           + 0.084 * IRC
           - 0.2807 * user
           + 0.3911

LM num: 5
Ruby =
           -0.0324 * x86
           - 0.0372 * GBE
           + 0.0635 * IRC
           + 0.021

Number of Rules : 5

Time taken to build model: 0.02 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correlation coefficient                  0.6781
Mean absolute error                      0.0867
Root mean squared error                  0.1813
Relative absolute error                 76.8708 %
Root relative squared error             76.3459 %
Total Number of Instances              100
```

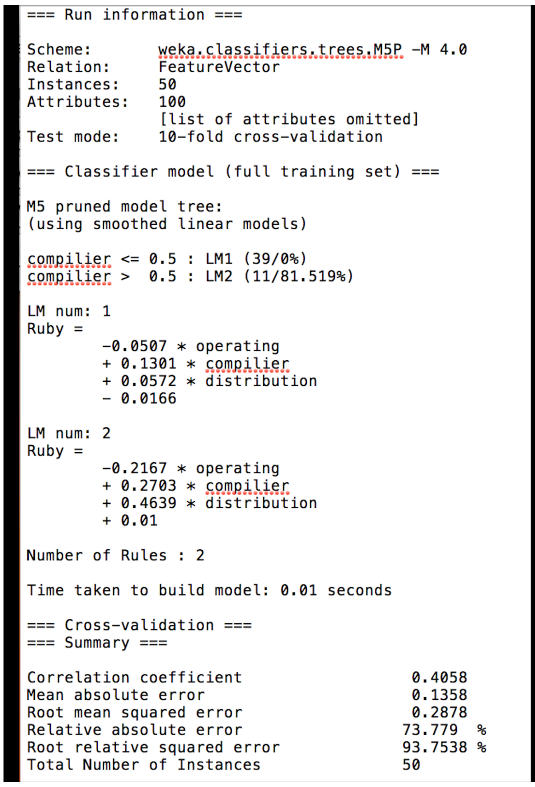


The above screenshots show the classifier output and visualizer along with the tree visualizer for the results. It's showing statistics of what keywords are most important in classifying data. From these results, we see that what Weka is going to do is use a "training set" to produce the model. This then takes a data set with known output values and uses this data set to build our model. Then, whenever we have a new data point, with an unknown output value, we put it through the model and produce our expected output.

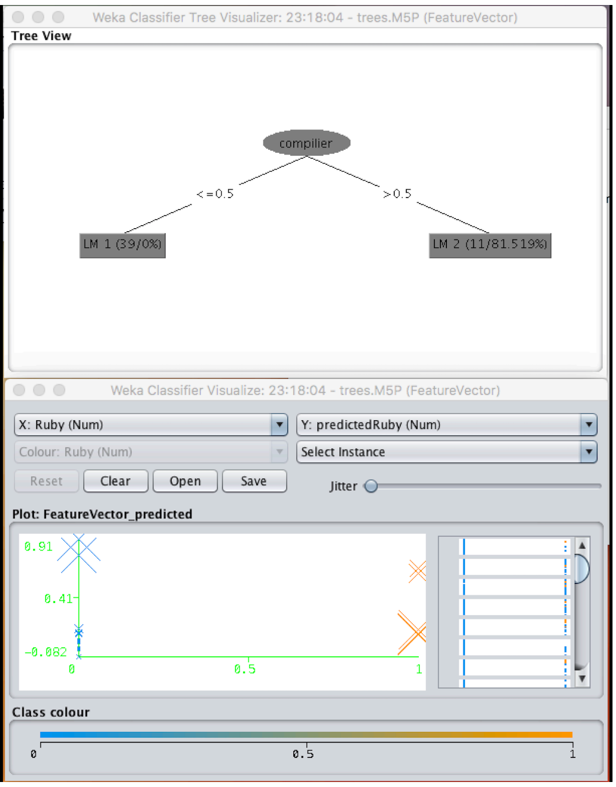
There is a limitation of training the pattern associator. By overfitting the model with data, we will have an algorithm that will work perfectly but only for the data we supply. It should work well for a variety of data.

<u>Data Set Two</u>:

Below are the screenshots from the second set of data that was run after the original training set. This is from the new-links.arff file that was created during this phase:



```
=== Run information ===

Scheme:       weka.classifiers.trees.M5P -M 4.0
Relation:     FeatureVector
Instances:    50
Attributes:   100
              [list of attributes omitted]
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

M5 pruned model tree:
(using smoothed linear models)

compilier <= 0.5 : LM1 (39/0%)
compilier >  0.5 : LM2 (11/81.519%)

LM num: 1
Ruby =
        -0.0507 * operating
        + 0.1301 * compilier
        + 0.0572 * distribution
        - 0.0166

LM num: 2
Ruby =
        -0.2167 * operating
        + 0.2703 * compilier
        + 0.4639 * distribution
        + 0.01

Number of Rules : 2

Time taken to build model: 0.01 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.4058
Mean absolute error                      0.1358
Root mean squared error                  0.2878
Relative absolute error                 73.779  %
Root relative squared error             93.7538 %
Total Number of Instances               50
```



As before; the above data shows us the classifier output along with the decision tree. Having trained with a feature vector set already, we see how the tree is pruned and has fewer leaves than before. Immediately we can see evidence of the pattern associator 'learning' from the original set. It now makes sense of how that was used as a training set.

Our studies with the Hebb & Delta rule now allow us to understand the algorithm used and the technique in which the weight vectors have been modified to give us a better result.

Looking at the original classifier output and the second output, we can see that out relative absolute error has gone down by 3.092%! This may not seem like much, but it is quite a good improvement in only one generation of 'learning' from the data set. If we were to supply more .arff feature vectors, the errors would probably drop quickly and then plateau at a certain point. At that point, it is probably better to not continue training the associator as it would then be effective only for the specific data sets which we create, and not for external data.

<u>Further Improvements & Conclusion</u>:

Looking back on this entire experiment, an improvement we would make is to automate the creation of the ARFF file. In the case of our team, we felt that we could do it simpler, so we wrote a collector.py script which scanned each website for all our keywords. The output was then saved as an ARFF format and we were then able to import it directly into Weka. The script written by Aaron and I is a python script called "collector.py". It uses the files "links.txt" (which contains all the links) and "keywords.txt" (Containing all 100 keywords) to generate the ARFF file "featureVector.arff".

Another change would be to use a data mining tool with a more detailed visual description of the classified data. Weka is powerful, and has a lot of capabilities, but the sheer amount of options and the general overloaded user interface made it very difficult to understand the data and comprehend what the results were indicating.

- Tushar Iyer & Aaron Shea