**Assignment 1**

**Aydin Atay, CS 7641, Fall 2018**

**Georgia Institute of Technology**

**Computer Science Department,**

**Introduction**

I was born in Bulgaria where almost everyone used to make their own wine. I too learned how to make home-made wine, and it is extremely difficult to make good wine. Reproducing the same wine year after year is almost impossible, it depends on that year’s temperatures, precipitation, natural events, fertilizing, when the grapes are picked and so on and so forth. I picked a wine dataset because I want to know if predicting the wine quality based on variables is possible. On the other hand, wine taste is a subjective quality which also varies from a consumer to another.

The second dataset I choose is a car quality dataset. The dataset has relatively small number of attributes, e.g. *number of doors, number of seats, maintenance cost, lug boot,* and *safety* which should give an idea of what is important for a buyer while choosing a car. It would be quite interesting if a buyer could definitely make a selection based on only 5 factors!

In general, I see a slight similarity between these two dataset: both are about the quality of a product based on some attributes related to its production and testing. A friend of mine is using [www.brightcellars.com](http://www.brightcellars.com) which uses some kind of algorithm trying to predict a customer’s wine choice based on a 7 question quiz. They are asking the questions below:

*1. What chocolate type do you like*

*2. What tea type do you like*

*3. What alcoholic drink do you like*

*4. What fruit juice do you like*

*5. What do you pair with wine? fine dining, friends, beach etc*

*6. How adventurous are you with new food and drink?*

*7. Red or white or something in between?*

**Datasets**

Both datasets were downloaded from Weka datasets suggested on Piazza. These dataset are in original Weka .arff format and both come with a training and test datasets. The download link is below:

<http://www.cs.ubc.ca/labs/beta/Projects/autoweka/datasets/>

More information on Weka .arff dataset format can be found here:

<https://www.cs.waikato.ac.nz/ml/weka/arff.html>

**Wine Dataset:** This dataset is has 12 attributes of which the last is a class attribute. There are 3429 instances in the training set and 1469 instances in the test set . This is a reduced set of only white wines.

**Car Dataset:** This dataset has 6 attributes and 1 class. There 1210 instances in the training set and 518 instances in the test set.

**Methodology:**

**Learning Curve Experiments:** In Weka I used GUI to perform experiments. During experiments 10 fold cross validation was used with changing the sample size of the training set from 5% to 100%. In Weka Explorer :

1. I used Meta Classifier -> FilteredClassifier.
2. From the FilteredClassifier configuration tab I can choose the classifier and the filter.
3. As filter I use unsupervised -> instance -> Resample.
4. From Resample configuration tab I chose noReplacement = True and vary the sample size from 5% to 100%. I use an excel sheet to plot training performance % versus training data % and training time and test performance
5. Supplied test set, load the test.arff
6. Start modeling and measuring on train set,
7. after training and train run finishes, run model on test set

I repeated the experiments for all the algorithms using the above procedure.

**Decision Tree Experiments:**

Weka is using a modified version of C45 algorithm written in Java, so they call it J48. The learning curve experiments were conducted as described in “Methodology” above. For parameter optimization the Confidence Factor ‘C’ and Minimum Number of Objects ‘M’ were changed. After adjusting parameter values the model was built and tested on training set, then the model was tested on the test set.

Chart 1. Decision Tree Learning Curves

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Confidence Factor: C | Minimum Number of Instances per leaf: M | Number of Leaves | Size of tree | Build Time | Training Performance (% correct) | Testing Performance (% correct) |
| 0.1 | 2 | 388 | 775 | 0.15 | 55.85 | 56.09 |
| 0.1 | 3 | 264 | 527 | 0.11 | 56.05 | 54.53 |
| 0.2 | 2 | 477 | 953 | 0.11 | 56.14 | 58.2 |
| 0.2 | 3 | 381 | 761 | 0.12 | 56.13 | 55.68 |
| 0.3 | 2 | 531 | 1061 | 0.16 | 56.61 | 58.34 |
| 0.3 | 3 | 428 | 855 | 0.14 | 55.85 | 56.2 |
| 0.4 | 2 | 534 | 1067 | 0.12 | 56.58 | 58.34 |
| 0.4 | 3 | 440 | 879 | 0.1 | 55.91 | 56.02 |

Figure 2. Model Complexity and Parameter Optimization on Decision Tree

**Neural Network Experiments:**

Weka is using an ANN algorithm which they call MultilayerPerceptron. According to Weka documentation the default layer parameter ‘a’ is calculated as 'a' = (attribs + classes) / 2. The learning curve experiments were conducted as described in “Methodology” above. For the parameter optimization experiments the default values were used. The optimization was made on Learning Rate and Momentum values.

**Figure 3.** ANN Learning Curves

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Layers | Learning Rate | Momentum | Build Time (s) | Training Performance (% correct) | Testing Performance (% correct) |
| 6 | 0.1 | 0.1 | 12.03 | 55 | 57.68 |
| 6 | 0.2 | 0.1 | 12.71 | 55.18 | 58.94 |
| 6 | 0.3 | 0.1 | 12.2 | 55.26 | 59.2 |
| 6 | 0.1 | 0.2 | 13.35 | 54.97 | 57.66 |
| 6 | 0.2 | 0.2 | 11.78 | 54.74 | 59.32 |
| 6 | 0.3 | 0.2 | 11.84 | 55.35 | 58.5 |
|  |  |  |  |  |  |

**Figure 4.** Model Complexity and Parameter Optimization on ANN

**Boosting Experiments:** Weka’s boosting algorithm is called AdaboostM1. As the weak learner the J48 Decision Tree algorithm was used. The learning curve experiments were conducted as described in “Methodology” above (Figure 5). The tree size depends on the Confidence factor and number of objects on the branches so to optimize parameters I experimented with Confidence Factor ‘C’ and number of iterations (Figure 6).

**Figure 5.** Adaboost Learning Curves

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Confidence Factor: C | Iterations | Number of Leaves | Size of tree | Build Time (s) | Training Performance (% correct) | Testing Performance (% correct) |
| 0.25 | 10 | 445 | 889 | 1.35 | 60.96 | 61.47 |
| 0.25 | 25 | 438 | 875 | 3.25 | 64.1 | 62.97 |
| 0.4 | 10 | 443 | 885 | 1.23 | 61.94 | 61.74 |
| 0.4 | 25 | 421 | 841 | 3.08 | 63.11 | 63.85 |

**Figure 6.** Model Complexity and Parameter Optimization on AdaboostM1

**SVM (Support Vector Machines) Experiments:** Weka has an SVM algorithm which is called LibSVM. The learning curve experiments were conducted as described in “Methodology” above (Figure 7). SVN is an algorithm which tries to separate classes into hyperplanes by calculating distances between instances close to boundaries. In my optimization experiments I used all available kernels to see how they behaved with the default values and tried to interpret what it meant.

Figure 8: SVM Learning Curves

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Kernel | degree | gamma | Coeff | Build Time | Training Performance (% correct) | Testing Performance (% correct) |
| Linear u\*v | 3 | 1/maxIndex | 0 | 99.96 | 52.87 | 50.58 |
| Polynomial (gamma\*u\*v +C)^degree | 3 | 1/maxIndex | 0 | 732.92 | 45.26 | 48.06 |
| Radial Basis Function exp(-gamma\*|u-v|^2) | 3 | 1/maxIndex | 0 | 3.67 | 57.01 | 54.6 |
| Sigmoid tanh(gamma\*u\*v +C) | 3 | 1/maxIndex | 0 | 0.68 | 45.11 | 44.31 |