Amber Fisher

CAP 4770

22 March 2021

Assignment 4 Report

Important: The data files are already in the zip file. If you decide to adjust the file structure, you must ensure you change the file paths in the first cell (under heading “File paths”) to correspond with the correct files. If you do not, the Jupyter notebook will not be able to read the data.

1. Partitioning the data: For this project, I did binary classification on all four datasets to improve the overall results. As such, there was a decent amount of preprocessing work done to convert the labels from multiclass to binary. I also used the same data splits for all four models, so the explanation for each dataset applies to the Random Forest, SVM, Neural Network, and Naïve Bayesian models for that dataset. You can find data statistics in the “About the data” table below.

* Car evaluation: Once the data was imported, I immediately split it into features and labels. From there, I used a custom function[[1]](#footnote-1) to ordinal encode the categorical features, then min/max-scaled the entire features array. Once that was complete, I was able to split the data into training data and testing data[[2]](#footnote-2). Unfortunately, Julia does not mix well with the Scikit Learn Python module, so I had to spend an entire step on converting the DataFrames to Arrays after I had split them into training and test data. The final step was to convert the training and testing labels from multiclass to binary. For this, I changed the “vgood” and “good” values to “acc”. This resulted in a binary classification of acceptable versus unacceptable cars.
* Abalone: Once the data was imported, I immediately split it into features and labels. From there, I used a custom function1 to label encode the categorical features (only sex in this dataset), then min/max-scaled the entire features array. Once that was complete, I was able to split the data into training data and testing data2. Unfortunately, Julia does not mix well with the Scikit Learn Python module, so I had to spend an entire step on converting the DataFrames to Arrays after I had split them into training and test data. The final step was to convert the training and testing labels from multiclass to binary. Since the original data had age (as a function of rings) as the value, I converted any labels with a value of 9 rings or less (11.5 years or less) to a value of “young”. All other labels then got a value of “old”.
* Madelon: This was the easiest dataset to import by far. The data was split into training, validation, and testing data already. Since I decided to use cross-validation for all the models, I combined the training and validation sets into a single training set. The values were all integers of similar size, so no extra transformations were needed. Additionally, the labels already formed a binary classification, so there was no need to change those labels either. The final step was to simply convert the DataFrames to Arrays.
* KDD Cup 1999: For this dataset, I used the 10 percent version of the files as all the model training was taking way too long with the full dataset. Once the data was imported and the columns were named, I immediately split it into features and labels. The data was already split into training and testing sets, so this split actually resulted in the training features and labels as I already had the testing features. From there, I used a custom function1 to ordinal encode the categorical features, then min/max-scaled the entire features array. Unfortunately, Julia does not mix well with the Scikit Learn Python module, so I had to spend an entire step on converting the DataFrames to Arrays after I had split them into training and test data. The final step was to convert the training and testing labels from multiclass to binary. For this, I changed any label values that were not “normal” to “abnormal” to see if it was possible to detect an abnormal connection.

2. Hyperparameter tuning:

* Random Forest: For this model, I did hyperparameter tuning on the Car evaluation, Abalone, and Madelon datasets. The tuning method I used was a randomized search on the maximum number of features, maximum depth, minimum number of samples to split a node, minimum number of samples for a node to be a leaf, and whether or not to bootstrap samples. There was no tuning done on the KDD set as it was taking an extremely long time for even five iterations of the search.
* SVM: There was no hyperparameter tuning done for this model as it was taking too long for even 10 iterations of a randomized search on Car evaluation dataset. Given a more powerful computer and more time, I would have done a randomized search with 25+ iterations on hyperparameters “C”, “penalty”, “dual”, and “loss”. However, I did use bagging on the model to help improve the model performance. I also used the LinearSVC model rather than the traditional SVC model to improve training time, though it had some pitfalls of its own.
* Naïve Bayesian: No hyperparameter tuning was done as there really isn’t any hyperparameters to tune with this model.
* Neural Network: For this model, I did a grid search on hyperparameters “alpha” and “learning rate” to find the best parameters. This grid search was done for all datasets, though the KDD dataset search takes a long time.

3. The error of any given model was calculated by finding the average of the 5-fold cross validation scores. The exact performance of a particular model on a particular dataset can be found in the “Training/testing statistics” table below. To determine if a model performed well on a dataset, I primarily used OpenML to observe others’ attempts at modelling the sets, as well as some critical analysis skills for the datasets that did not have much past testing.

* For the Car evaluation set, most of my models did moderately well. The evaluation I found on OpenML managed to get nearly perfect classification of the features, whereas my best score was around 87% for the Naïve Bayesian model. The worst model by far was SVM with a score of around 69%. However, this may have been due the use of the LinearSVC model rather than the normal SVC model, which would have allowed for a different kernel. The other two models did adequate at classifying the data with scores of 75 and 85 percent.
* For the Abalone set, I couldn’t find any evaluations that showed a binary classification of the data. However, there were evaluations that did multi-class classification. These runs had a maximum accuracy of 28%. Considering I managed to get in the 70s for my binary classification accuracy, I feel that these models did do fairly well, despite the lackluster scores on the face of it. All four models were within five percentage points of each other, which was quite interesting. The highest scores were 78-79% for three of the models, and the worst was the Naïve Bayesian model with a score of almost 74%.
* The Madelon set had prediction values almost exactly equal to what the evaluation set I found had for each model. The Random Forest model did the best by miles with a score of 85%. The other three models did much more poorly at classifying the data. All three had scores below 60%. These statistics held true in the evaluation set that I found on OpenML.
* The KDD set performed extremely well for three of the four models. With the exception of the Naïve Bayesian model, the set had an accuracy of 98.8+%. The Naïve Bayesian was the only one that did somewhat poorly, and that one still had an accuracy of 94%. I expect that the high performance of all the models was due to the large size of the dataset. This set had far more samples than any other set.

4. The size of the models were determined by dumping the model to a file using “joblib” and then applying the Base.stat function from Julia to get the size of the file. The exact sizes can be found below in the “Training/testing statistics” table. The largest models were the Random Forest models, and the smallest were generally the Naïve Bayesian models.

5. The testing time was determined by timing how long a particular model took to predict values for the testing features. The exact times can be found below in the “Training/testing statistics” table.

6. The training time was determined by timing how long a particular model took to fit to the training data. However, it should be noted that I also timed how long it took to complete the randomized searches for the sets that I tuned hyperparameters for. This could sort of be considered training time, though in reality, you would likely find the model ahead of time with a smaller subset of the data before fitting to the entire dataset. The exact times can be found below in the “Training/testing statistics” table.

About the data:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Number of features | Training size | Testing size |
| Car Evaluation | 6 | 1209 | 519 |
| Abalone | 8 | 2923 | 1254 |
| Madelon | 500 | 2600 | 1800 |
| KDD Cup 1999 (10%) | 41 | 494,021 | 311,029 |

Training/testing statistics:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Classifier across,  dataset down | Random Forest | SVM (with Bagging) | Naïve Bayesian | Neural Network |
| Car Evaluation | Accuracy: 0.87684  Size: 1.703 MB  Training: 0.120 sec  Testing: 0.013 sec | Accuracy: 0.68901  Size: 6.585 KB  Training: 0.034 sec  Testing: 0.006 sec | Accuracy: 0.74856  Size: 0.903 KB  Training: 0.001 sec  Testing: 0.0004 sec | Accuracy: 0.86772  Size: 34.321 KB  Training: 0.508 sec  Testing: 0.0007 sec |
| Abalone | Accuracy: 0.79062  Size: 6.610 MB  Training: 0.202 sec  Testing: 0.036 sec | Accuracy: 0.77865  Size: 6.905 KB  Training: 0.217 sec  Testing: 0.038 sec | Accuracy: 0.73793  Size: 0.967 KB  Training: 0.012 sec  Testing: 0.005 sec | Accuracy: 0.77933  Size: 39.864 KB  Training: 0.864 sec  Testing: 0.005 sec |
| Madelon | Accuracy: 0.85192  Size: 4.101 MB  Training: 0.538 sec  Testing: 0.030 sec | Accuracy: 0.54961  Size: 85.640 KB  Training: 5.118 sec  Testing: 0.182 sec | Accuracy: 0.59346  Size: 16.696 KB  Training: 0.031 sec  Testing: 0.012 sec | Accuracy: 0.50346  Size: 1.611 MB  Training: 1.171 sec  Testing: 0.011 sec |
| KDD Cup 1999 (10%) | Accuracy: 0.99683  Size: 5.097 MB  Training: 3.880 sec  Testing: 0.501 sec | Accuracy: 0.98814  Size: 12.221 KB  Training: 21.900 sec  Testing: 1.599 sec | Accuracy: 0.93945  Size: 2.055 KB  Training: 0.446 sec  Testing: 0.112 sec | Accuracy: 0.99259  Size: 143.035 KB  Training: 42.399 sec  Testing: 0.379 |

Note: The accuracy value is stated as a decimal. However, it could be changed to a percentage.

Ex: Car evaluation/Random Forest: 0.87684 => 87.684% accuracy

1. The function used to transform the data was “transform\_features”. See Jupyter file comments for more information about this function. [↑](#footnote-ref-1)
2. The function used to split the data was “train\_test\_split”. See Jupyter file comments for more information about this function. [↑](#footnote-ref-2)