p2_report

Joseph Kim

January 27, 2016

Contents

Chapter 1

Classification vs Regression

This is a binary classification problem because the target variable we're trying to predict is one of two values, "yes" or "no", which represents whether each student passed or failed, respectively. The students that need intervention are ones who are classified to fail by the resulting model.

Chapter 2

Exploring the data

```
Out[11]:                                              Fact   Value
        0                         Total number of students     395
        1                    Number of students who passed     265
        2                    Number of students who failed     130
        3                   Graduation rate of the class (%)  67.09%
        4  Number of features (excluding the label/target...      30
```

Chapter 3

Training and Evaluating Models

3.0.1 Perceptron

The Perceptron model is used for binary classification problems. Its main strengths are that it is simple, training and predicting times are fast, and the model performs well for many linearly-separable data, but won't work well on non-linear data. Perceptrons are the basis of more complicated neural networks, which combine multiple perceptrons in multiple layers to solve more complex problems that single perceptrons couldn't solve.

I chose this model because I thought it'd be a good benchmarking point to start off the analysis, since the problem of predicting failing students is a binary classification problem.

```
Out[17]:          clf    description f1_score_test f1_score_train time_predict_test  \
        0  Perceptron  set size: 100      0.781457       0.785276       0.000150919
        1  Perceptron  set size: 200      0.766234       0.798799       0.000133038
        2  Perceptron  set size: 300      0.568421       0.583333       0.000219107

          time_predict_train   time_train
        0        0.000164032  0.000788927
        1        0.000155926  0.000953913
        2        0.000240088   0.00161505
```

3.0.2 K-nearest neighbors

k-nearest neighbors is a non-parametric model used for both classification and regression problems. It works well when there are exponential amounts of rows of data with respect to the number of columns, such that the data is evenly spread out among the dimensions. If the training data does not cover the space of each dimension well, then the model may not work well.

This algorithm performs "lazy" training, which means that when new data is added, it doesn't have to recompute the model, but querying for predictions takes longer since the model has to seach through the data and find the nearest neighbors of the query.

I chose this model because it is a classification algorithm known to work very well in many cases, and though I knew that the dataset for this assignment did not have vast amounts of data points with respect to the number of features, I thought it might be worth a try.

```
Out[18]:                    clf    description f1_score_test f1_score_train  \
        6  KNeighborsClassifier  set size: 100      0.666667       0.823529
        7  KNeighborsClassifier  set size: 200      0.666667        0.86121
        8  KNeighborsClassifier  set size: 300      0.721805       0.886878

          time_predict_test time_predict_train   time_train
        6       0.000865936         0.00100303  0.000437975
```

```
7           0.001261          0.00243902  0.000473022
8         0.00220585          0.00481796  0.000613928
```

3.0.3 Support Vector Machine

Support vector machines are used for both classification and regression problems, and generally work well for many types of problems. The algorithm also has ways to fit data using higher-dimensional feature space, which makes it versatile. A drawback of this algorithm is that training time is slower than simpler models like the perceptron, though prediction time is fast.

I chose this model because of its versatility and knew that it performs well for binary classification problems. Given that the dataset in this assignment is not that big (300 rows and 30 columns in training data doesn't seem very big compared to a lot of other datasets these days), I felt confident that running SVM on it won't take very long despite the expected high training time.

```
Out[19]:    clf    description f1_score_test f1_score_train time_predict_test  \
         3  SVC  set size: 100      0.753846       0.924242       0.000540018
         4  SVC  set size: 200      0.772727       0.895833       0.000828981
         5  SVC  set size: 300      0.776978       0.887892       0.000988007

           time_predict_train  time_train
         3        0.000566006  0.00175405
         4         0.00188613  0.00866389
         5         0.00325894    0.011225
```

3.0.4 Random Forest

Random Forests are used for both classification and regression problems, and also work generally well for many types of problems. Their drawbacks are that training takes a long time, and that because it learns so well, the user has to be especially careful about overfitting, especially if data is noisy.

I used this model because they are known to perform very well whether the dataset is big or small, and also I thought it'd be a good thing to compare the results of the random forest classifier with support vector machines.

```
Out[20]:                        clf    description f1_score_test f1_score_train  \
         9   RandomForestClassifier  set size: 100      0.730159       0.992126
         10  RandomForestClassifier  set size: 200      0.730159       0.996226
         11  RandomForestClassifier  set size: 300       0.75188       0.992701

            time_predict_test time_predict_train time_train
         9        0.000771046        0.000905037  0.0230501
         10       0.000986099          0.0015111  0.0218599
         11        0.00090313         0.00156689   0.018795
```

3.0.5 Gradient Boosting Classifier & Ada Boost Classifier

Boosting methods are powerful ways to increase the performance of other learning algorithms ("weak learners"), but they may not perform well if the data is noisy.

I choose these algorithms because I wanted to see if I can improve the performance of the other learning algorithms using "boosting".

```
Out[21]:                            clf    description f1_score_test f1_score_train  \
         12  GradientBoostingClassifier  set size: 100      0.695652              1
         13  GradientBoostingClassifier  set size: 200      0.742424              1
         14  GradientBoostingClassifier  set size: 300      0.737705              1
         15          AdaBoostClassifier  set size: 100      0.765957       0.805369
```

```
16          AdaBoostClassifier  set size: 200      0.758621       0.811688
17          AdaBoostClassifier  set size: 300      0.761905       0.819876

   time_predict_test time_predict_train time_train
12       0.000345945        0.000527143   0.103263
13       0.000380993        0.000712872   0.179376
14       0.000837088         0.00142908   0.304954
15          0.028378          0.0286541   0.157823
16          0.023634           0.050462   0.164545
17          0.136949           0.403715    1.13411
```

Chapter 4

Choosing the best Model

The most important indicator for the performance of the different models is the f1_score for the test set using all the training data. Among the top performing models, the time taken to predict and train should then be considered to choose the most practical model for this problem, given the resource constraints. The table below shows the models sorted by decreasing F1 scores on the test data

```
Out[31]:                         clf f1_score_test   time_train time_predict_test
         5                        SVC      0.776978     0.011225       0.000988007
         17        AdaBoostClassifier      0.761905      1.13411          0.136949
         11    RandomForestClassifier       0.75188     0.018795        0.00090313
         14  GradientBoostingClassifier    0.737705     0.304954       0.000837088
         8       KNeighborsClassifier      0.721805  0.000613928        0.00220585
         2                 Perceptron      0.568421   0.00161505       0.000219107
```

The top 3 candidates are SVC, AdaBoostClassifier, and RandomForestClassifier, whose f1 scores are all within .03 of each other. I felt that the right parameters of each of the models can affect the final performances of these models, so I went ahead and optimized all three models.

On the test dataset, SVM's final f1 score was 0.795, Random Forest scored 0.808 and the Ada Boost Classifier using SVM as the base model scored 0.766. Between SVM and Random Forest, the time to predict (0.011 seconds vs 0.019 seconds, respectively) and time to train are very similar to each other (both less than .001 second on the test set). They are both known to have issues with slower training times with large datasets, and both are able to run predictions very quickly. Therefore, since Random Forest had a higher final f1 score, the Random Forest Classifier is the best choice. Furthermore, it provides an additional parameter, n_jobs, which utilizes parallel processing to find results more quickly, while SVC doesn't. Therefore, we can reduce the time it takes to run it, and since many cloud services charge based on server uptime rather than number of job processes, and each node usually has multiple processors, we can utilize those models to increase cost-efficiency.

4.1 Brief review on Random Forests Classifiers

4.1.1 Random Forests

Random forests are a tool used to make predictions, in our case, whether a student is at risk of failing a class or not. We feed the program a bunch of historical data about past students that have passed and other students that have failed, and the tool "learns" the data in such a way that when we present it with a new student who is currently taking the class, the program will make a prediction on whether he or she will pass the class. It's not always 100% accurate, but it predicts with a high level of accuracy such that we can use it to help students who are at risk of failing, and help them pass the class.

Random forests are basically more than one (could be just a few, could be hundreds or even thousands) decision trees, each that casts a vote to decide the final prediction for an observed data point. In our case, the optimized model had 15 trees. That means that each of the 15 trees would predict whether a student is going to fail or not, and each would cast a vote with how certain it is of a given prediction provided as a probability. For example, for a given student Bob, if 10 trees predicted that Bob would pass with a probability of .75, and 5 trees predicted that Bob would fail with probability .5, then the entire random forest would make the final prediction that Bob would pass.

Out[1]:

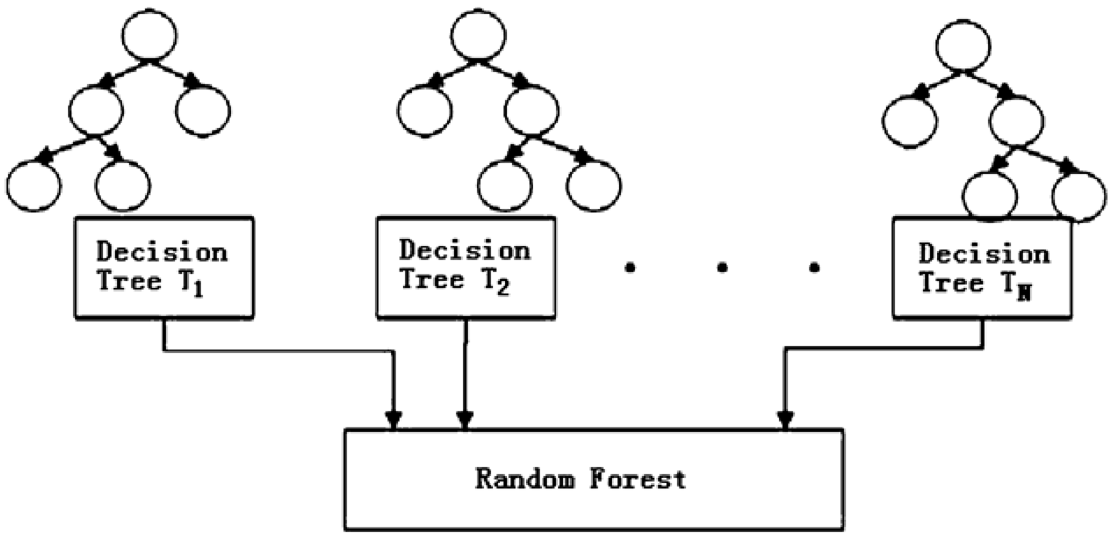
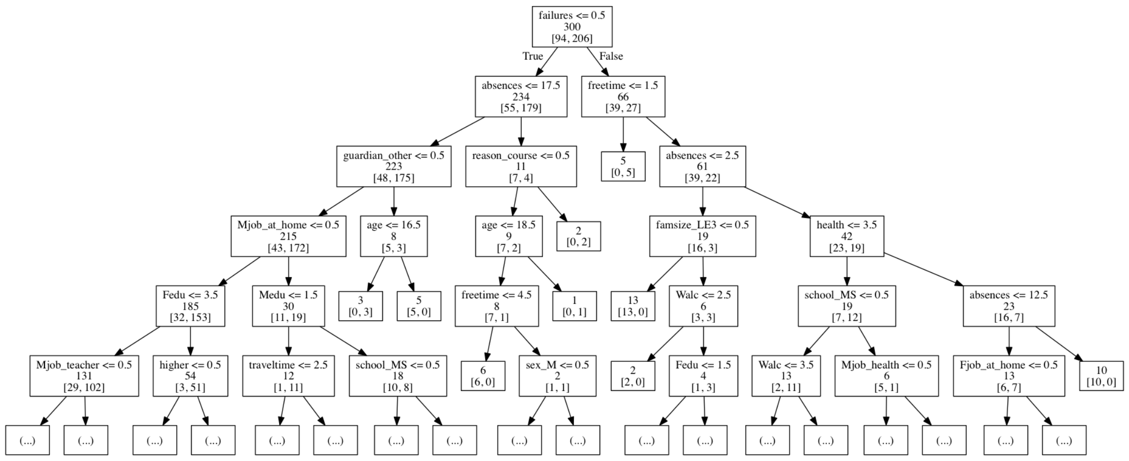


Image taken from http://what-when-how.com

4.1.2 How does each decision tree work?

Each decision tree takes some of the historical data about past students who have passed and some who have failed, and creates a set of questions. Based on the answers to these questions, the decision tree will decide whether a new student like Bob is at risk of failing or not. The followign diagram shows an example of a decision tree on our student data.

Out[4]:


failures <= 0.5 300 [94, 206]
True
False
absences <= 17.5 234 [55, 179]
freetime <= 1.5 66 [39, 27]
guardian_other <= 0.5 223 [48, 175]
reason_course <= 0.5 11 [7, 4]
5 [0, 5]
absences <= 2.5 61 [39, 22]
Mjob_at_home <= 0.5 215 [43, 172]
age <= 16.5 8 [5, 3]
age <= 18.5 9 [7, 2]
2 [0, 2]
famsize_LE3 <= 0.5 19 [16, 3]
health <= 3.5 42 [23, 19]
Fedu <= 3.5 185 [32, 153]
Medu <= 1.5 30 [11, 19]
3 [0, 3]
5 [5, 0]
freetime <= 4.5 8 [7, 1]
1 [0, 1]
13 [13, 0]
Walc <= 2.5 6 [3, 3]
school_MS <= 0.5 19 [7, 12]
absences <= 12.5 23 [16, 7]
Mjob_teacher <= 0.5 131 [29, 102]
higher <= 0.5 54 [3, 51]
traveltime <= 2.5 12 [1, 11]
school_MS <= 0.5 18 [10, 8]
6 [6, 0]
sex_M <= 0.5 2 [1, 1]
2 [2, 0]
Fedu <= 1.5 4 [1, 3]
Walc <= 3.5 13 [2, 11]
Mjob_health <= 0.5 6 [5, 1]
Fjob_at_home <= 0.5 13 [6, 7]
10 [10, 0]
(...)

In the tree shown above, each box represents a question, and basd on the answer, we'd follow the corresponding arrows to the next question until we run out of questions, at which the tree will present us with an answer. For example, we take Bob, and the first question we'd ask is "What is the failure rate of Bob in previous classes?". If the answer is less than or equal to 0.5, then the next question is "How many absences has Bob had?". Based on that answer, we'd go left if True, right if false, and so on. The numbers in each box respresent statistics about the historical data that the tree used to train on.

Each tree has a different set of questions that are built from a different subset of the historical students data, so not all trees would come up with the final prediction at the end. That is why the random forest takes the votes of several trees into account, in order to get a higher-confidence prediction.

For those that are interested in the technical details

Each tree is built by training on its sampled training data, which are the same size as the training set, but contains random repeats (bootstrapping), which is a way to decrease variance due to noise during training.

During training, each decision tree is looks for the best way to branch down based on the information content of each attribute in the training data. The attributes that make it easier to make predictions on the data points are selected first, which results in the shortest trees possible. The tree is branched down until only a single class of target values are left in the leaf of the branch, or until the `max_depth` or `min_samples_split` values have been reached.

Once the forest of decision trees are trained, when new data points are inputted, each tree will traverse down its branches using the values of the data points' attributes until the final leaf containing the prediction target value, and cast its vote. The final output of the random forest classifier is the elected prediction based on weighted average of the probability scores of each tree.

4.2 Final F1 Score

The final F1 Score of the random forest classifier is 0.808