P2: Student Intervention System

Jan 16, 2016

OVERVIEW

Find the most effective model with the least amount of computation costs to model the factors that predict how likely a student is to pass their respective classes.

GOALS

Choose and develop a model that will predict the likelihood that a given student will pass, thus helping diagnose whether or not an intervention is necessary.

SPECIFICATIONS

Develop this model in conformation with this Rubric: https://docs.google.com/document/d/1eyMT5SkhK4qiiFfTz1TrSnSqkV9gbH8g0Jfh3kW6apI/pub?embedded=true

MILESTONES

Classification OR Regression

Primary objective of this project is to classify if a student needs intervention or not. Hence this is a classification problem. Output hypothesis of chosen model will predict if the features corresponding to a specific student classify her / him in either of the only 2 classes - those needing intervention or not.

Some of the input features, like absences, are continuous. So techniques useful in regression, like outlier identification, can be used to analyze such continuous features, or could be used for feature reduction. But most other features are either categorical or boolean. Further data exploration will be required to better understand feature set.

Exploring the data

Here is statistical summary of data, as generated from program:

Total number of students: 395

Number of students who passed: 265

Number of students who failed: 130

Number of features: 30

Graduation rate of the class: 67.09%

One observation is that feature count is less than amount of data available. Although the amount of data might still not be sufficient to create a perfect model, if there is much general or feature-specific noise in it.

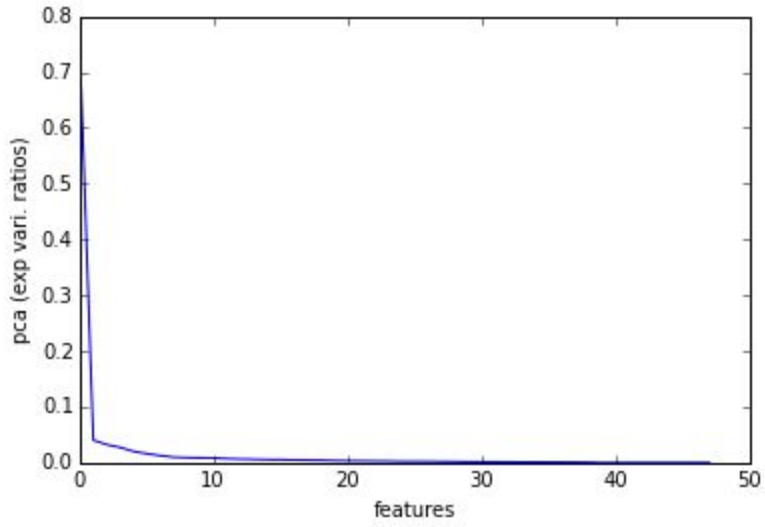
Preparing the Data

Since some of the features are boolean, while some others are categorical, the data has been pre-processed to convert it into numeric data. But this has increased the number of features / columns from 30 to 48, thus making it difficult to create great models with the given amount of data.

As a side project, I did two things to reduce features from this modified data:

1. Visualize relationships between various features.
2. And ran a PCA to reduce features, and plotted variance ratios against feature numbers. This showed that about top 25 features after PCA fitting are main influencers.

But since this was not required for this project, I have not included that version of code in submission.



After pre-processing data, it has been split into Training (count=300) and Test sets (count = 96), using cross_validation.train_test_split method from sklearn.

Training and Evaluation Models

Given the mostly non-continuous nature of feature set, and its relatively small size, we can start with 3 classification models:

1. Support Vector Machines

This is good for a mixture of boolean and continuous data, and can handle non-linear decision boundaries as well, by means of kernel trick. Although it is not good for large data sizes, which is not the case here with only about 400 records. And this is simple to understand, and visualize. We will use **SVC** algorithm for this project.

Given the inherent strengths of Ensemble methods, we can also try include them in search for best model. Both these models can be scaled easily for available computing resources.

So, we are using 1 from each type of ensemble methods - Averaging and Boosting methods.

2. Boosting

Boosting starts with a basic simple model, and then improves upon it by way of weak learners. Since it learns more from mistakes, it concludes quickly on correct model. We will use **AdaBoost** algorithm, as it allows multiple base estimators, and allows scaling as per available computing resources.

3. Bagging

Bagging models start with simpler models, and then use randomization to reduce overfitting and variance. We will be using **RandomForestClassifier** implementation for our project. This implementation support parallelization as well.

To start with we will use default parameters, to see the starting performance, and then select one model to improve upon it.

Here we are training and testing all 3 models on Training Sizes of 50, 100, 150, 200, 250, and 300. This would be helpful in visualizing the patterns.

Here are required data points for the 3 models:

SVC:

SVC data matrix

	50	100	150	200	250	300
Training time (secs)	0.00097	0.00153	0.00268	0.00400	0.00605	0.00782
Prediction time (secs)	0.00081	0.00079	0.00121	0.00144	0.00172	0.00191
F1 score for training set	0.90625	0.85906	0.87081	0.86928	0.87918	0.86920
F1 score for test set	0.73438	0.78378	0.77143	0.77551	0.75862	0.75862

AdaBoost:

Ada Boost data matrix

	50	100	150	200	250	300
Training time (secs)	0.00356	0.00304	0.00319	0.00357	0.00397	0.00435
Prediction time (secs)	0.00043	0.00031	0.00031	0.00063	0.00036	0.00032
F1 score for training set	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
F1 score for test set	0.68966	0.70085	0.71545	0.70312	0.72269	0.71667

Random Forests:

Random Forest data matrix

	50	100	150	200	250	300
Training time (secs)	0.03088	0.02436	0.02497	0.02566	0.02574	0.02475
Prediction time (secs)	0.00091	0.00109	0.00110	0.00093	0.00139	0.00098
F1 score for training set	0.98305	1.00000	0.98936	0.99254	0.99415	0.99034
F1 score for test set	0.74016	0.77863	0.71875	0.74809	0.78571	0.77165

Above results from one of the several runs, which had similar results. Prediction times and F1 scores for Test Features using the 3 selected models varies a little. More specifically, F1 scores varied within +/- 0.07. But what was consistent was that AdaBoost took least amount of time, always.

Choosing the Best Model

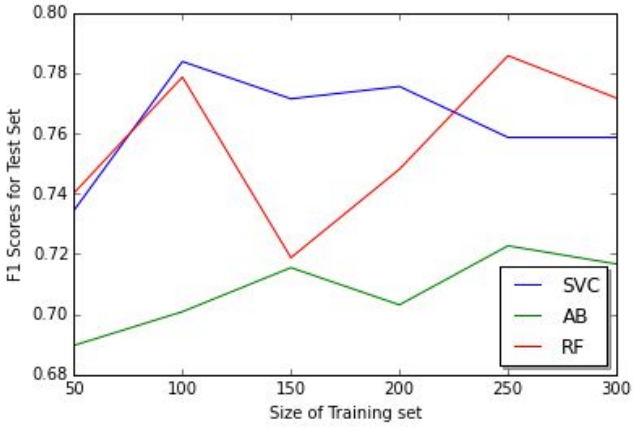
To choose most optimal model, we have an additional matrix / plot of accuracies and time taken to predict each model:

Aggregated model performance numbers:

```
----------------------------------------------------------------------------------
Accuracy score and time taken for various models, given diffferent training data sizes.

----------------------------------------------------------------------------------
     TST_F1_SVC TST_F1_AB TST_F1_RF  Trng_size TST_TIME_SVC TST_TIME_AB TST_TIME_RF
50      0.73438    0.68966    0.74016          50       0.00081      0.00043      0.00091
100     0.78378    0.70085    0.77863         100       0.00079      0.00031      0.00109
150     0.77143    0.71545    0.71875         150       0.00121      0.00031      0.00110
200     0.77551    0.70312    0.74809         200       0.00144      0.00063      0.00093
250     0.75862    0.72269    0.78571         250       0.00172      0.00036      0.00139
300     0.75862    0.71667    0.77165         300       0.00191      0.00032      0.00098

----------------------------------------------------------------------------------
```



To make a more informed choice of model, I also tried to get memory consumption of the 3 models, in the manner explained below:

1. Clone the main program into 3 different versions, each having common code until train_predict() method, plus a model-specific code.
2. So, student_intervention_SVC.py has common code plus code for SVC model.
3. Similarly, student_intervention_RF.py had common code and code for Random Forest model, and so on student_intervention_AB.py for AdaBoost model.

4. Use program available here: https://github.com/FrancescAlted/ipython_memwatcher for memory usage analysis.
5. Ran following sequence of commands 3 times, one for each model
    a. import numpy as np
    b. from ipython_memwatcher import MemWatcher
    c. mw = MemWatcher()
    d. mw.start_watching_memory()
    e. run student_intervention_SVC.py OR run student_intervention_AB.py OR run student_intervention_RF.py

At the end of program execution, output similar to following was generated on command line:

In [5] used **44.332 MiB RAM** in 1.951s, peaked 0.000 MiB above current, total RAM usage 100.953 MiB

Repeated this on a separate ipython instance, twice for each model.

These memory numbers are reported below. Although there is small difference in memory usage, but still the pattern apparent is that memory utilization of SVM < Adaboost < Random Forest.

But time takes was always least for AdaBoost.

Of-course this was using default features.

Model / Time & Memory	**SVM**	**AdaBoost**	**Random Forest**
Memory Used:	1.) 42.895 2.) 43.152	1.) 43.371 2.) 43.871	1.) 44.227 2.) 44.172
F1 Score on Test Features while using Trng Size=300	0.75862	0.72881	0.76119
Prediction time on Test features while using Trng Size=300	0.00171	0.00031	0.00096

Choosing the Best Model

Here I have tried 2 things:

1. Manual tuning of AdaBoost.
2. Tuning of RandomForest using GridSearchCV, as per requirements.

Manual tuning of AdaBoost:

Since accuracy of these models is similar, but time taken by AdaBoost is the least, we can try tuning AdaBoost to improve its F1 score. This way AdaBoost will be able to handle very large data set in production. Albeit out-of-scope for this project, in production we can train several model for some time with all available features and all available data (probably GBs/ TBs), and then make an even more practical choice.

Tuned Parameters:

```
learning_rate=0.70
for i in range(1,7):
    # Creating a new instance of DecisionTree Classifier to avoid any mixup with previous instance.
    dt_clf=DecisionTreeClassifier(presort=True, min_samples_leaf=1,max_depth=1,max_features=26)
    clf = AdaBoostClassifier(dt_clf,algorithm="SAMME",n_estimators=20,learning_rate=learning_rate)
```

Model Stats:

```
--------------------------------------------------------------------------------
Selected Model (Ada Boost) data matrix
                              50      100      150      200      250      300
Training time (secs)      0.02700 0.02900 0.03200 0.03200 0.03200 0.03300
Prediction time (secs)    0.00100 0.00100 0.00100 0.00100 0.00100 0.00100
F1 score for training set 0.93103 0.80851 0.81731 0.83276 0.83110 0.84279
F1 score for test set     0.71074 0.75556 0.78261 0.79433 0.78571 0.80282
--------------------------------------------------------------------------------
```

Now this tuned AdaBoost model has higher F1 score while taking equal or less time than any of the models originally selected.

Auto-tuning using GridSearch

Since Random Forest is a bagging algorithm and can work with shallow trees, we can choose it to tune using GridSearch. Main parameters we can use in param grid are max features, criterion, and depth, along with samples split and bootstrap.

Using these features, as coded at the end of student_intervention.py, we get following values for params and F1 score.

Best model parameter: {'bootstrap': True, 'min_samples_leaf': 2, 'min_samples_split': 1, 'criterion': 'entropy', 'max_features': 48, 'max_depth': 2}

f1 score: 0.81429

Now this F1 score is slightly better than that from previous, non GridSearch, version of RandomForest.

Conclusions:

1. There is not much difference in F1 scores of SVM, AB, and RF algorithms.
2. GridSearchCV could be used to tune meta-parameters, and can help.
3. Identifying relationships between various features, and using that domain knowledge to manually tune params can also improve accuracy.
4. 0.81 is the ceiling for F1 score, for all the models we tried.
5. Although, F1 scores are best when using all 300 data points, but there is negligible difference in F1 when using 200 data points and 300 data points.
6. Increasing estimator count, and number of jobs takes increasingly more resources with small or no improvement in accuracy. This means the choice of base estimator, and pre-processing (noise-reduction and feature reduction) are very important. Just giving more resources will not help always.

Further analysis required:

1. Reason for ceiling of 0.81 for F1 score.
2. Reason for variation in F1 scores with increase in training size.
3. Find bias and variance in each of the models.
4. Further analyze the models, if more data is available from different sources or over time from same institute.

Explanation to board of supervisors

Our goal for this project is to create a model that can predict with fair amount of certainty if a student needs intervention to pass a class.

Over last week, we have analyzed student data, and came across interesting patterns. For example, there is negligible correlation between consuming alcohol during weekdays and absenteeism. Also, students whose both parents are either very less educated or highly educated, have not failed in past.

Based on our analysis, we have found that while some of the attributes of student data have more impact on prediction, other attributes are not of much significance.

We started with 3 models, and planned to choose the best mode that can predict best while using minimal computing resources and time. Time parameter considered here was time required by model to predict if intervention is required or not, as this is a repeatable task. Time taken to train the model is very less frequency, hence time taken in training model is of less importance.

But we have found 2 models that are equally good in terms of accuracy of prediction, but differ in terms of computing resources and time required to predict. One of these models uses Boosting to learn more from its failure to predict. Another model uses Bagging to start with a multiple simpler models and combine accuracy from each of these to create more complex and accurate models.

So we recommend using both these models for a longer period and fine tune both these models with increasingly more data points. and feature sets. During this phase we will also analyze both their computing needs, and decide on what inhouse / cloud platform is best to host this model.

F1 score from manually-tuned **AdaBoost: 0.803**

F1 score from Grid-search-tuned **RandomForest: 0.814**