Udacity - Machine Learning Engineer Nanodegree

P2: Build a Student Intervention System

A supervised learning project using the provided “student-data.csv” data file. Code performed in ipython notebook file “student_intervention.ipynb”.

Project Description

As education has grown to rely more and more on technology, more and more data is available for examination and prediction. Logs of student activities, grades, interactions with teachers and fellow students, and more are now captured through learning management systems like Canvas and Edmodo and available in real time. This is especially true for online classrooms, which are becoming more and more popular even at the middle and high school levels.

Within all levels of education, there exists a push to help increase the likelihood of student success without watering down the education or engaging in behaviors that raise the likelihood of passing metrics without improving the actual underlying learning. Graduation rates are often the criteria of choice for this, and educators and administrators are after new ways to predict success and failure early enough to stage effective interventions, as well as to identify the effectiveness of different interventions.

Toward that end, your goal as a software engineer hired by the local school district is to model the factors that predict how likely a student is to pass their high school final exam. The school district has a goal to reach a 95% graduation rate by the end of the decade by identifying students who need intervention before they drop out of school. You being a clever engineer decide to implement a student intervention system using concepts you learned from supervised machine learning. Instead of buying expensive servers or implementing new data models from the ground up, you reach out to a 3rd party company who can provide you the necessary software libraries and servers to run your software.

However, with limited resources and budgets, the board of supervisors wants you to find the most effective model with the least amount of computation costs (you pay the company by the memory and CPU time you use on their servers). In order to build the intervention software, you first will need to analyze the dataset on students’ performance. Your goal is to 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. Your model must be developed based on a subset of the data that we provide to you, and it will be tested against a subset of the data that is kept hidden from the learning algorithm, in order to test the model’s effectiveness on data outside the training set.

Your model will be evaluated on three factors:

- Its F1 score, summarizing the number of correct positives and correct negatives out of all possible cases. In other words, how well does the model differentiate likely passes from failures?
- The size of the training set, preferring smaller training sets over larger ones. That is, how much data does the model need to make a reasonable prediction?
- The computation resources to make a reliable prediction. How much time and memory is required to correctly identify students that need intervention?

1. Classification vs Regression

Identifying students who might need early intervention is a classification supervised machine learning problem. Specifically, the goal is to model the factors that predict how likely a student is to pass their high school final exam. Therefore, our output is a discrete label, “pass” or “not pass”, where our model can find a decision boundary between the two.

2. Exploring the Data

Statistic to Analyse	Explored Data
Total number of students	395
Number of students who passed	265
Number of students who failed	130
Graduation rate of the class (%)	67.09%
Number of features(excluding the label/target column)	30

Table 2.1 Data Analysis

3. Preparing the Data

To prepare the data for modeling, training and testing, the code will display the following outputs:

- Identifying feature and target columns

```
Feature column(s):-
    ['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu', 'Mjob', 'Fjob', 'reason', 'guardian',
    'traveltime', 'studytime', 'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet',
    'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Walc', 'health', 'absences']
    Target column: passed
```

- Preprocessing feature columns

```
Processed feature columns (48):-
    ['school_GP', 'school_MS', 'sex_F', 'sex_M', 'age', 'address_R', 'address_U', 'famsize_GT3', 'famsize_LE3',
    'Pstatus_A', 'Pstatus_T', 'Medu', 'Fedu', 'Mjob_at_home', 'Mjob_health', 'Mjob_other', 'Mjob_services',
    'Mjob_teacher', 'Fjob_at_home', 'Fjob_health', 'Fjob_other', 'Fjob_services', 'Fjob_teacher', 'reason_course',
    'reason_home', 'reason_other', 'reason_reputation', 'guardian_father', 'guardian_mother', 'guardian_other',
    'traveltime', 'studytime', 'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet',
    'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Walc', 'health', 'absences']
```

- Splitting data into training and test sets

```
Training set: 300 samples
    Test set: 95 samples
```

4. Training and Evaluating Models

Three supervised learning models were chosen for this dataset: Support Vector Machines, AdaBoost, and Naive Bayes.

Support Vector Machine

Support Vector Machines (SVMs) work on classification and regression problems by finding robust separating hyperplanes in high dimensional space. This hyperplane maximizes the margin, which is the distance between the line and the nearest points, or support vectors, relative to either class. Applications of SVMs have been used on text categorization, classification of images, hand-written character recognition, and many more problems.

Advantages of SVMs include :

- Effective in high dimensional spaces
- Prediction efficient: uses a subset of training points in the decision function called support vectors
- Versatile: different kernel functions may be specified for the decision function including linear, polynomial, rbf, sigmoid, etc.

Disadvantages of SVMs include:

- Not effective for any problem with many training examples because their compute and storage requirements increase rapidly
- Memory inefficient: all training data must be stored to make predictions
- Likely to overfit with too many features

The SVM classifier is chosen because of its flexibility of kernel functions for the decision boundary. Additionally, the data is high dimensional (30 features) which SVMs are effective at modeling.

Training Set Size	**100**	**200**	**300**
Avg training time (secs)	0.0020	0.0056	0.0103
Avg prediction time (secs)	0.0011	0.0020	0.0026
Avg F1 score for training set	0.7654	0.8024	0.7928
Avg F1 score for test set	0.8344	0.8344	0.8344

Table 4.1 Support Vector Machine Performance (using RBF kernel)

AdaBoost

AdaBoost is short for “Adaptive Boosting” and it works by fitting weak learners on the data and combining their predictions through a weighted majority vote, or sum, to produce the final prediction. Applications of Adaptive Boosting include face detection, general classification and regression problems, and many more problems.

Advantages of AdaBoost include :

- Much less tweaking of parameters needed

- Does feature selection: finds the best features for learning

Disadvantages of AdaBoost include:

- Sensitive to noisy data and outliers
- Slower training time

The AdaBoost classifier with Decision Tree estimators was chosen because decision trees are nonparametric and therefore do not have to worry about the data being linearly separable. Additionally, feature selection could be very helpful in such a high dimensional dataset (30 features).

Training Set Size	**100**	**200**	**300**
Avg training time (secs)	0.0080	0.0091	0.0110
Avg prediction time (secs)	0.0009	0.0009	0.0009
Avg F1 score for training set	0.8369	0.8288	0.8009
Avg F1 score for test set	0.8531	0.8085	0.8085

Table 4.2 AdaBoost Performance (using Decision Tree estimators)

Naive Bayes

Naive Bayes methods are based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features. Its applications include spam filters, classification of news articles, facial recognition, and many more problems.

Advantages of Naive Bayes include :

- Fast training and prediction
- Not sensitive to irrelevant features

Disadvantages of Naive Bayes include:

- Loss of accuracy by assuming conditional independence

The Multinomial Naive Bayes classifier is chosen because of its fast classification speed.

Training Set Size	**100**	**200**	**300**
Avg training time (secs)	0.0017	0.0017	0.0019
Avg prediction time (secs)	0.0002	0.0002	0.0002
Avg F1 score for training set	0.8108	0.8161	0.7991
Avg F1 score for test set	0.8212	0.8276	0.8194

Table 4.3 Naive Bayes Performance (using Multinomial NB)

5. Choosing the Best Model

To reach the district's goal of 95% graduation rate on a strict budget by the end of the decade, three factors must be evaluated. First, the chosen model must offer a reasonable average F1 score: SVM (83.44%), AdaBoost (80.85%), MultinomialNB (81.94%). Second, the chosen model must prefer smaller training sets. Average training time for AdaBoost and MultinomialNB scale linearly as training size is increased while SVM training time is scaled exponentially. Third, the model must offer a short average training time: SVM (10.3 ms), AdaBoost (11.0 ms), MultinomialNB (1.9 ms). Therefore, the chosen model must balance between the three performance metrics.

Taking these factors into account, the best model is Multinomial Naive Bayes for classifying whether a student needs intervention or not. The F1 performance of MultinomialNB is marginally worse than the best performer, SVM, by ~1.5%. However, MultinomialNB trains about 4x faster than SVM and a Decision Tree estimator AdaBoost. Another consideration, at the end of every school year the district has more examples of whether students passed or not. MultinomialNB affords the option of updating the dataset and retraining the model with minimal impact to computational resources. The training time will only increase linearly with more training examples unlike SVM which would increase exponentially. Therefore, Multinomial Naive Bayes optimally balances F1 score, computational resources, and preference of smaller training sets.

Naive Bayes works through Bayesian analysis by assuming each feature is independent of one another. Put simply, Bayesian analysis for the student dataset looks at the prior evidence, the features like school, age, sex, etc., to consider how likely a student passes or not. It is considered "naive" because the model simplifies its assumptions by claiming each feature is independent of another. An example of dependant features is a student is less likely to have extra paid classes when his or her mother or father has a lower income job. Naive Bayes will not take an example like this into account and therefore the model will lose accuracy. However, the model and industry examples listed in section 4 work effectively even with the simplification.

Not much tuning is required for this model because the only parameter to consider is the smoothing parameter 'alpha'. The tuned model's F1 score (84.21%) is better than its previous F1 score (81.94%). MultinomialNB's final performance metrics after gridsearch tuning is:

```
MultinomialNB(alpha=25, class_prior=None, fit_prior=True)
------------------------------------------
Model: MultinomialNB...
Training set size: 300
Avg training time (secs): 0.0018
Avg prediction time (secs): 0.0002
Avg F1 score for training set: 0.8035
Avg F1 score for test set: 0.8421
```