Udacity Machine Learning Nanodegree

Project 2: Supervised Learning
Building a Student Intervention System

Jingzhe Hu

1 Classification vs Regression

Identifying students needing early intervention is a **classification** problem in supervised machine learning. In this case, the predicted outcomes constitutes two categories, in which a student may be predicted to either pass or fail based on available data. In regression problems, the outcome shall be continuous rather than categorical.

Given limited resources, the goal of the project is to select the most effective model that predicts whether a student will pass, with the least amount of computational cost.

2 Exploring the Data

The students in this dataset have a graduation rate of around **67.1%** (265/395), according to Table 1. Other than the outcome of the students, 31 additional pieces of information are collected on each student. A quick inspection of the dataset using **pandas.DataFrame** reveals no missing data point.

total number of students	395
number of students (passed)	265
number of students (failed)	130
number of features	31
graduation rate of the class	67.1%

Table 1: Basic statistics of student_data.csv

3 Preparing the Data

To ensure compatibility with machine learning algorithms, categorical features from the input dataset can be converted into dummy variables, which is outlined in the accompanying project iPython notebook. Instead of the custom implementation, it's possible to utilize **pandas.get_dummies** to create the dummy variables, after which the number of features increases from 31 to 48.

In order to independently verify the model performance, the dataset can then be randomized and split into a training set (N=300) as well as a testing set (N=95). We can accomplish this step with scikit-learn's **train_test_split** function.

4 Training and Evaluating Models

Due to limited computing resources available, I elect to use the following three classifiers: decision tree (**DT**), k-nearest neighbors (**KNN**) and logistic regression (**LR**). In the following subsections, I will briefly outline their general applications, strengths as well as weaknesses of each choices.

4.1 Decision Tree

Decision tree can be used for both classification and regression. It has the following **advantages**: **a)** fast training and prediction speed, **b)** no need for feature-scaling, **c)** can be easily visualized, for example with GraphViz and **d)** can handle feature interactions. In terms of **disadvantages**, decision tree algorithm **a)** tends to overfit training data, which often requires pruning and **b)** doesn't support online-learning, i.e., model need to be re-trained from the ground up to account for new data.

Decision tree can be thought of as asking a series of question and breaking down the data by categories. Since many of the features in the dataset are categorical, decision tree algorithm is a natural choice. According to Table 2, for training size of 100, 200 and 300, the F1-scores for the testing set are **0.765**, **0.706** and **0.738** respectively. The F1 scores for the training set are all 1, clearly indicating overfitting with default parameters. Since both training and prediction time are on the order of 1 ms for current dataset, this model is suitable for limited computing resource setting.

	Training set size		
	100	200	300
Training time (ms)	0.6	1.0	1.4
Prediction time (ms)	0.1	0.1	0.1
F1 score for training set	1.0	1.0	1.0
F1 score for testing set	0.765	0.706	0.738

Table 2: Performance: **decision tree**

4.2 K-Nearest Neighbors

KNN can be applied in both classification and regression. The **advantages** are fast training speed and non-parametric while the **disadvantages** are **a)** dependence of prediction time on sample size and **b)** potential need for feature-scaling, feature-selection and dimensionality reduction.

KNN classifier does not make assumptions of the underlying dataset, which makes it potentially suitable in this case.For training set size of (100, 200, 300), the F1 scores are 0.803, 0.797 and 0.815 respectively for the testing set. Since KNN is a type of lazy algorithm, the prediction time is an order of magnitude higher compared to decision tree classifier. (Table 3)

	Training set size		
	100	200	300
Training time (ms)	0.3	0.4	0.6
Prediction time (ms)	0.8	1.1	1.6
F1 score for training set	0.832	0.872	0.862
F1 score for testing set	0.803	0.797	0.815

Table 3: Performance: **k-nearest neighbors**

4.3 Logistic Regression

Logistic regression can be used in classification tasks. The **advantages** are **a)** fast training/prediction and **b)** supports online-learning with stochastic gradient descent. Regarding **disadvantages**, logistic regression needs feature-scaling when regularizations are used.

Logistic regression may be applied due to limited demand of resources and moreover, the classification results can be interpreted as probabilities belonging to a certain class. From Table 4, the F1 scores for the testing set are 0.75, 0.797 and 0.794 for training set size of (100, 200, 300).

	Training set size		
	100	200	300
Training time (ms)	0.7	1.2	1.8
Prediction time (ms)	0.1	0.1	0.1
F1 score for training set	0.874	0.872	0.830
F1 score for testing set	0.765	0.797	0.794

Table 4: Performance: **logistic regression**

5 Choosing the Best Model

For KNN, training + prediction time is around 47% and 16% slower compared to decision tree and logistic regression for a training size of 300. Since computing resource is a hard constraint, KNN shall be eliminated from the viable options, despite having the highest F1 score of 0.815 on the testing set (n=95) without tuning. Decision tree algorithm has the best time efficiency, around 26% faster compared to logistic regression, while having a F1 score of 7% less on the testing set.

Given the available data, logistic regression appears to offer a good balance between computational cost and model performance. After conducting more rigorous testing (25 independent runs with reshuffling training/testing set and GridSearchCV), logistic regression is found to offer one of the best average F1-score of around 0.797 among several classifiers (n=7) including ensemble methods such as AdaBoost (F1-score=0.798) and Random Forest classifier (F1-score=0.797). Moreover, logistic regression also provides a good average precision of 0.705, second only to Stochastic Gradient Descent(precision=0.739) and AdaBoost (precision=0.735). Precision is crucial for this application since false positive, student that will fail but classified as passed, carries disproportionate negative impacts. As a result, **I would recommend logistic regression for building the student intervention system**.

After training the logistic regression model with the whole dataset and optimal parameters generated previously with GridSearchCV, the final F1-score is **0.820**, which is around 3.4% better compared to the model trained with default parameters.

Here, I will provide a brief description of the logistic regression model - how the classification works and how to train the model. In order to perform student classification, we first assign a weight to each of the student's attribute and then sum them up. Then we can feed this result into the logistic function to map out the probability that the student will pass. If the probability is greater or equal to 0.5, the student is predicted to pass by the model.

Since we already know the expected classification outcome of some students, we can use a portion of the data as training set to find the optimal weights for each attributes. For each iteration of the training process, we try to gradually push the weights to the direction which will maximize the chance that the training data points are classified correctly. Once the training process is finished, we can evaluate model performance with a testing set and then the model will be ready for production. When new student data points become available, the model can be retrained to potentially increase performance.