Building a Student Intervention System

George Liu

Overview

With the advent of the data analytics era, student learning data is becoming increasingly available through various online education tools such as Edmondo etc. The availability of these data, when coupled with powerful machine learning techniques, will provide invaluable insights that were never possible before. The aim of this project is to build a student intervention system that uses high school student data to make predictions about their likelihood of passing final exam, so that appropriate early intervention can be made to ensure student success.

Since our goal is to predict whether a specific student is going to pass or not, i.e., to decide which class a specific data point belongs to, this is a classification task. We can use supervised learning techniques to generate classification rules based on available data.

Data Summary and Preprocessing

The "student_data.csv" contains 395 data points and 31 columns including the label column "passed". Therefore, there are a total of 30 features. These features are:

['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']

The following is some quick facts of the data:

- 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%

Therefore, this is a small and unbalanced data set. Thus, an algorithm that works with small data set and stratified shuffle split cross-validation are necessary in later modeling phases.

We start preprocessing the data by splitting it into features("X_all") and labels("y_all"). Since a lot of columns are categorical variables containing non-numeric values, the data is further processed through value replacement and dummy variable creating. The data is then split into train test tests.

At a later stage, when the linear SVC algorithm is used, further processing including principal component analysis and scaling are conducted for the model to produce the best result.

Model Training and Evaluation

Based on the type of classification task on hand, four algorithms are initially chosen: Linear support vector classification (LinearSVC), K-Nearest Neighbors, Random Forest and Naïve Bayes. Following the machine learning map on Scikit Learn, LinearSVC is the first choice for supervised learning classification tasks with less than 100K samples. The next candidate in line for non text data is k-nearest neighbors. In the case it does not work well, both SVC and ensemble classifiers such as random forests can be considered. The following table provides the general applications, strengths/weaknesses of each algorithm.

Algorithms	General Applications	Strengths and Weaknesses
Support Vector Machines (SVM)	Classification (text, image etc.), regression	**Strengths** – robust and provides good out-of-sample generalization with right parameters, flexible and can handle complex non-linear classification thanks to different kernels, delivers a unique solution (Auria and Moro 2008), works well with limited data, effective in high dimensional spaces, absence of local minima (Shawe-Taylor and Cristianini 2004) **Weaknesses** – low transparency (Auria and Moro 2008), can be slow, hard to tune due to the choice of the kernel (Burgess 1998)
K-nearest neighbors (KNN)	Classification, regression	**Strengths** – simple (nickgillian.com), requires no training time (K Nearest Neighbors) **Weaknesses** – hard to decide distance type and features to use, can be very slow as distances between query instance and all data points need to be calculated (however it's worth noting that this calculation only happens with prediction as KNN is a so called lazy algorithm that does nothing other than storing the data when learning and only computes when prediction time comes. Reference: Strength and Weakness of K-Nearest Neighbors by Teknomo)

Random Forests (RF)	Classification, regression	**Strengths** – fast, scalable, can be easily interpreted, works well with large data sets, can easily handle feature interactions, even works with outliers and non-linearly separable data (Chen 2011), corrects for decision trees' habit of overfitting to their training set (Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome 2008) **Weaknesses** – Can be slow when complex
Naïve Bayes (NB)	Classification (text etc.)	**Strengths** –simple, fast even with large data sets, highly scalable, can be easily interpreted, requires a small amount of training data, not sensitive to irrelevant features, fast, handles well high dimensionality, (Gutierrez 2015) **Weaknesses** – zero probability issue happens when a given class and feature value never occur together in training data (but can be fixed by Laplace smoothing etc.), can't learn interactions between features

As the task we have is classification and the data set is not very large, I decided to test out model performance using the above four algorithms. In particular, SVM and Naïve Bayes are chosen as they work well with small sample size which is the case for our project. For KNN, since it is relatively easy to interpret and explain, and speed is not a big concern with small data sets, it is also a good candidate. Lastly, Random Forest can potentially help us handle feature interactions that might be present.

I also experimented with different training sizes with each model, recorded performance results including training time, prediction time and F1 score (on both training and testing sets) in the table below:

Algorithm	Train Set Size	Train Time	Prediction Time (on train set)	Prediction Time (on test set)	F1 Score (train set)	F1 Score (test set)
LinearSVC	**300**	**0.023**	**0.000**	**0.000**	**0.833**	**0.734**
	200	0.014	0.000	0.000	0.857	0.727
	100	0.005	0.000	0.000	0.924	0.677
KNeighbors	**300**	**0.001**	**0.006**	**0.003**	**0.874**	**0.771**
	200	0.000	0.003	0.002	0.857	0.754
	100	0.000	0.001	0.001	0.857	0.778
RandomForest	**300**	**0.014**	**0.001**	**0.000**	**0.990**	**0.722**
	200	0.014	0.001	0.000	0.992	0.756
	100	0.042	0.001	0.000	0.992	0.698
GaussianNB	**300**	**0.000**	**0.001**	**0.000**	**0.802**	**0.737**
	200	0.001	0.001	0.000	0.808	0.791
	100	0.001	0.000	0.000	0.815	0.707

Model Selection and Tuning

As shown in the above table, for the 300-sample size, K-nearest Neighbors algorithm has the worst prediction time (0.006 and 0.003 seconds), while the LinearSVC model gives the worst train time (0.023 seconds). Therefore, we are left with Random Forest and Gaussian Naïve Bayes which produce similar accuracy.

Between GaussianNB and RandomForest, the former is clearly the winner as it not only has a higher F1 score i.e. better prediction accuracy, the GaussianNB is also much faster to train - 0.000s vs. 0.014s. This is important because we now have a small data set and thus it is possible for us to retrain the model with more data later. Furthermore, the student intervention system will be used with cloud-based resources billed according to CPU and memory time which requires a more efficient model.

Therefore, I decided to choose GaussianNB for model building. Nevertheless, I still tuned the Random Forest model for the purpose of illustration since GaussianNB model doesn't provide any parameters for tuning. In terms of fine tuning the model, the following parameter choices are tested on the Random Forest model:

'randomforestclassifier__n_estimators': [10, 20],

'randomforestclassifier__max_features': ["auto", "sqrt", "log2"]

The best result is n_estimators = 20 and max_features = 'log2' with a maximized F1 score of 0.766422931671. When testing this model again using unseen data, the final F1 score is 0.78321678321678323, which is quite higher than the 0.722 score from the untuned model, proving the power of model tuning through cross-validation.

Let's now talk about Naïve Bayes in plain language. Now, we have some data about students. It shows that 80% of the students who failed the final exam share the following attributes:

- Family size is greater than 3
- Parents are divorced
- Parents education are secondary
- Study time is less than 2 hours
- ……

Now, if we are given a student's data, indicating that their profile matches with the above exactly, would you say this student more likely passes or fails? We would definitely say they are more likely to fail given the data we have. This is exactly how Naïve Bayes algorithm works. Before looking at the data, we don't have any specific reason to believe a student is particularly likely to pass or fail. However, given the data, we'll adjust our point of view and will be more confident to make a "educated guess".

To be more exact, say we have 100 students, if 50 of them pass, 50 fail, then without any further data such as family size, parent marriage status etc., we would think any student is equally likely to pass or fail (this likelihood is called "prior probability"). Now, if we are given data, say when a student passes the exam, 80% of them study for more than 2 hours, and for those students who fail, there is only a mere 1% who study for more than 2 hours. If we now have a student who studies for 3 hours ("data"), do we

believe they are more likely to pass or fail? (“posterior probability”). In order to decide this, we would look at the data given, which would lead us to believe that they will more likely to pass. So, combining the original likelihood of passing, with the data of study time, we can further conclude that the student is more likely to pass. The Naïve Bayes algorithm works in the same way except that there are more attributes of each student such as family size, parent education etc.

The algorithm is called “Naïve Bayes” because it assumes that the above mentioned attributes do not affect each other, which may not always be the case. In spite of this strong assumption, Naïve Bayes often provides great classification performance. The F1 score we get earlier is a good proof.

Reference

1. Machine Learning Map
2. http://MachineLearningMastery.com
3. Choosing a Machine Learning Classifier
4. When to choose which machine learning classifier?
5. Why use SVM?
6. How to choose algorithms for Microsoft Azure Machine Learning
7. https://commons.wikimedia.org/w/index.php?curid=3566688
8. Support Vector Machines (SVM) as a Technique for Solvency Analysis
9. What are the disadvantages of Naïve Bayes?