Udacity Machine Learning Nanogree Project 2 - Build a Student Intervention System

Jason Osajima

1 Project Description

Design a student intervention system for a school district by analyzing a dataset of student profiles, testing the performance of three different machine learning algorithms, and optimizing the parameters of the algorithm which performs the best.

1.1 Background

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.

1.2 Task

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.

2 Classification v. Regression

Your goal is to identify students who might need early intervention - which type of supervised machine learning problem is this, classification or regression? Why?

This is a supervised machine learning classification problem because we want to classify students as being either in need of early intervention or not in need of early intervention. This would be a regression problem if we required a continuous output.

3 Exploring the Data

In the dataset, there are a total of 395 students. Of those, 265 students passed their high school final exam and 130 students failed. The graduation rate is therefore 67.09%. The dataset contains 30 features which capture different aspects of the student.

4 Preparing the Data

My first task was to separate the data into feature and target columns and preprocess the feature columns by converting columns which have non-numeric inputs into columns with numeric inputs. This was done in two ways. In the case where the non-numeric columns contained yes/no inputs, the column could be converted to a numeric column by simply replacing ’yes’ with 1 and ’no’ with 0.

Columns that had more than two values were split into multiple columns (one column for each value) and assigned a 1 when the example had that value.

Next, I split the data into training and test sets. I chose to use a Stratified ShuffleSplit cross validation iterator since my dataset is both small and unbalanced. This ensures that I can train my model and preserve percentages of samples for each class. When I used Stratified ShuffleSplit originally, the function separated my data into 300 training examples and 40 test examples in order to maintain a comparable ratio of passing/failing students. In order to produce a training set of 300 examples and test set of 95 examples, I set a value for both the train_size and test_size parameters.

5 Training and Evaluating Models

I chose to train the data using three different supervised learning models: Support Vector Machines (SVM), Naive Bayes, and k-Nearest Neighbors (k-NN). In each of the following subsections, I discuss each model's general applications, strengths and weaknesses, and why I think the model would work well for the dataset.

5.1 Support Vector Machines

The goal of a Support Vector Machine (SVM) is to design a hyperplane that classifies all training vectors into two classes. The best choice will be the hyperplane that leaves the maximum margin from both classes. SVMs can be used either for classification or regression problems. General applications of SVMs include bioinformatics, text mining, face recognition, and image processing.

SVMs work well in complicated domains where there is a clear margin of separation. SVMs don't work as well with large datasets because the training time increases exponentially as you increase the amount of training examples. Another instance when SVMs do not work well is when the dataset has a lot of noise because SVMs are prone to overfitting (especially when using a complex kernel). In this case, a Naive Bayes model is a better solution.

One consideration when selecting the best SVM model is choosing between a linear or nonlinear kernel. Linear kernels are more appropriate when the number of features is larger than the number of training examples or the training examples exceed 50,000, which could severely impact the training time. Nonlinear kernels are more appropriate when the number of training examples exceeds the number of features.

I decided to apply a SVM model to the data because of the complexity of the data (large number of features) and the small number of training examples. Since there is a small number of training examples, a large training time would not be a limiting factor. I chose to use a Gaussian rbf kernel because the number of training examples (300) exceeds the number of features (30). The results of training and testing the data using a SVM with an rbf kernel are shown below.

Table 1 - Training and Prediction Times and F_1 Scores for SVM[1]

	Training Set Size		
	100	200	300
Training Time (s)	0.00219	0.00306	0.00925
Prediction Time on Training Data (s)	0.00099	0.00225	0.00525
Prediction Time on Test Data (s)	0.00092	0.00119	0.00172
F_1 Score on Training Data	0.87417	0.88158	0.85653
F_1 Score on Test Data	0.81290	0.77778	0.80537

5.2 Naive Bayes

The goal of Naive Bayes is to train a classifier that can classify examples by calculating probabilities using Bayes' Rule. More specifically, the probability that a given example is a member of a certain class is calculated given that example's features. The example is then classified by selecting the label that has the highest probability. Examples of Naive Bayes applications include classifying emails, news articles, and other text-based classification problems.

Naive Bayes is very simple to implement and works well with big feature spaces. It is also simple to run and converges quicker than other, more complicated algorithms because of the assumption that the features are independent of one another. The disadvantages for using a Naive Bayes model are that it may suffer from high bias because it is a simple representation and, if the conditions of independence between features doesn't hold (i.e. the features are correlated), it may provide an inaccurate model for the data.

I decided to apply a Naive Bayes model to the data because it can be implemented quickly and efficiently. Also, Naive Bayes works better when there is more noise since it is a high bias, low variance option and is less prone to overfitting. Given the large number of features (31) this condition may exist.

[1]SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma='auto', kernel='rbf', max_iter= -1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)

The results of training and testing the data using a Naive Bayes classifier are shown below.

Table 2 - Training and Prediction Times and F_1 Scores for Naive Bayes[2]

	Training Set Size		
	100	200	300
Training Time (s)	0.04058	0.00161	0.00117
Prediction Time on Training Data (s)	0.00034	0.00029	0.00030
Prediction Time on Test Data (s)	0.00025	0.00023	0.00023
F_1 Score on Training Data	0.79412	0.79004	0.79147
F_1 Score on Test Data	0.68750	0.69841	0.70769

5.3 k-Nearest Neighbors

The goal of the k-Nearest Neighbors algorithm is to classify an example using the k nearest neighbors to that example as reference. The k nearest neighbors are determined using a distance metric (typically either by measuring euclidian or manhattan distance). The algorithm then decides on a classification based on the labels of its neighbors. While classification is relevant for our example, it can also be used for regression. Some common applications of k-NN include classifying handwritten digits or satellite image scenes.

k-NN is a non-parametric, lazy learning algorithm. It is non-parametric because it does not make any assumptions on the underlying data (unlike an algorithm such as Naive Bayes, which makes the assumption that the features are independent). This allows it to be more versatile for different datasets and makes it more robust to noisy training data.

It is lazy because it doesn't make any generalizations based on the training data, which allows the training phase to run very quickly. This feature also makes its testing phase more costly in terms of time and memory.

Another disadvantage for using the k-NN algorithm is that the parameters are difficult to select, such as the value for k (number of nearest neighbors). Also, it is difficult to know what distance metric should be used to identify which points are nearest neighbors.

[2] MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)

I decided to use the k-NN algorithm because it is robust to noisy training data and doesn't make assumptions about the underlying data, which we have little information about (i.e. we don't know the relationship between the features).

Table 3 - Training and Prediction Times and F_1 Scores for k-Nearest Neighbors[3]

	Training Set Size		
	100	200	300
Training Time (s)	0.00496	0.00113	0.00099
Prediction Time on Training Data (s)	0.00402	0.00464	0.00773
Prediction Time on Test Data (s)	0.00208	0.00196	0.00224
F_1 Score on Training Data	0.85517	0.84691	0.87133
F_1 Score on Test Data	0.76596	0.77241	0.77465

6 Choosing the Best Model

Based on the results, the Support Vector Machine (SVM) is the best model for this data. The SVM outperformed both models with a F_1 score of 0.80537 and, while slower than Naive Bayes, was faster than k-Nearest Neighbors in prediction time by 22.2%. Since the school district has a limited budget for this project and will be reaching out to a third party to provide software libraries and servers, SVM is a better option than k-NN because it will be computationally more efficient when making predictions for future students. However, both training and prediction time will scale with increasing training data for SVM. If the school district decides to increase the size of the training set, at some point it would make sense to use k-NN. Since it is an instance-based learning algorithm, it will be more computationally efficient to train when compared to SVM.

Table 4 - Comparison Table for the Three Models

	F_1 Score for Test Set	Training Time	Prediction Time
SVM	0.80537	0.00925	0.00172
Naive Bayes	0.70769	0.00117	0.00023
k-NN	0.77465	0.00099	0.00224

[3]KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=5, p=2, weights='uniform')

A Support Vector Machine (SVM) works by taking the data and drawing a boundary that divides students who have passed their high school final exam from those who haven't passed their final exam. Next, when we give the SVM a new student, it decides which side of the boundary the student belongs on based on their characteristics. The algorithm then predicts whether a student will pass his/her final exam based on what side of the boundary the student ends up in.

Using GridSearchCV, I tuned the 'C' parameter for the SVM Classifier and found that a 'C' value of 0.15 resulted in the best F_1 score of 80.6%.