Student Intervention System

Udacity Machine Learning Engineer
Nanodegree Program: Project 2

Anderson Daniel Trimm

February 8, 2016

1 Classification vs Regression

Our model predicts which students will pass or fail their high school final exam, and therefore need an intervention. Since we are predicting *discrete* labels (in this case, binary), this is a *classification* problem, as opposed to a *regression* problem, in which we would be predicting *continuous* labels.

2 Exploring the Data

In this section, we use the code in the accompanying ipython notebook to identify important characteristics of the dataset which will influence our prediction. Running the code, we find

```
Total number of students: 395
Number of students who passed: 265
Number of students who failed: 130
Number of features: 31
Graduation rate of the class: 0.67%
```

3 Training and Evaluating the Models

3.1 KNN Classifier

The k-nearest neighbor classifier finds the k training samples closest in distance to the query point, and predicts the label from these.

Pros:

- Fast training time - it simply remembers all the training points

- Being non-parametric, it can be useful in classification problems where the decision boundary is very irregular

Cons:

- Potentially long training time - it has to search through the dataset to find the k nearest neighbors
- Uses a lot of CPU memory, since it has to store the dataset (as opposed to a parametric model, which throws away the dataset after learning the parameters)

Despite the memory cost, since our dataset is not too large, kNN is a reasonable classifier to try. The results for our model with the default parameter settings are recorded in the table below.

	100	200	300
Training time (sec)	0.001	0.001	0.001
Prediction time (sec)	0.001	0.002	0.003
F1 score for training set	0.88	0.88	0.88
F1 score for test set	0.78	0.77	0.78

3.2 Gaussian Naive Bayes Classifier

Naive Bayes is a learning algorithm based on applying Bayes' theorem with the assumption that every pair of features are independent (hence the use of "naive"). Here, we use the Gaussian naive Bayes classifier, which assumes the likelihood of each feature is Gaussian. Pros:

- Querying is fast
- Few parameters (linear in number of variables)

Cons:

- Learning is slow
- Doesn't model interrelations between attributes

	100	200	300
Training time (sec)	0.001	0.001	0.001
Prediction time (sec)	0.001	0.000	0.000
F1 score for training set	0.33	0.83	0.80
F1 score for test set	0.24	0.74	0.76

3.3 Support Vector Machine Classifier

Pros:

- Effective in high dimensional spaces
- Can still be effective in cases where the number of dimensions is greater than the number of samples
- Uses only a subset of training points in the decision function (the “support vectors”), so it is memory efficient
- Versatile, since different kernel functions can be specified for the decision function

Cons:

- If the number of features is much greater than the number of samples, it is likely to perform poorly
- Does not directly provide probability estimates; these are calculated using a memory expensive five-fold cross-validation

	100	200	300
Training time (sec)	0.001	0.004	0.008
Prediction time (sec)	0.001	0.002	0.002
F1 score for training set	0.90	0.89	0.87
F1 score for test set	0.78	0.80	0.78

4 Choosing the Best Model

4.1 Optimal Model

Comparing the three tables, we see that both KNN and SVM outperformed Naive Bayes in terms of F1 score as the size of the training set decreased. In fact, their F1 score did not decrease by decreasing the size of the training set from 300 to 100. We can therefore restrict our training set to only 100 data points, which leads to savings on memory cost as well as training time. Between these two algorithms, support vector machines still uses less memory, as it only uses a subset of the training points in the decision function, whereas KNN uses them all. Since the training and prediction times, as well as the F1 score on the test set for these two algorithms are the same (training on 100 data points), we choose SVM as the optimal model.

4.2 Description of Model in Layman's Terms

To determine which students will pass or fail using our data, we first plot the data. We will pretend there are only two features, so that we can plot these points in the xy-plane on a sheet of paper. The general idea will be the same. Our model will determine where to draw a line that separates the xy-plane into two regions: data points falling into one region will all be passing, and all those falling into the other will be failing. This line will be chosen as to minimize the number of misclassifications, while at the same time maximizing its ability to correctly classify new, unseen data points.

4.3 Model Tuning and Tuned F1 Score

We tune our SVM model using gridsearch over the parameter "C", which is the penalty parameter of the error term. Running the code, we find the optimal C-value is 0.5. The optimized model gives has an F1 score of 0.82 on the test set, which is indeed an improvement of the F1 score of 0.78 of the default SVM model.