Build a Student Intervention System

Table of Contents

Project Background

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 behaviours 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.

Project Aim

The first project goal 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. The second goal is to find the most effective model with the least amount of computation costs.

Methodology

- Analyse the dataset on students' performance. The 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.
- The model is developed based on a subset of the data, and it is 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.
- The model is 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.

Data Source

https://github.com/udacity/machine-learning/tree/master/projects/student_intervention\student-data.csv

Software and Libraries

- Python 2.7
- NumPy 1.10
- scikit-learn 0.17
- iPython Notebook (with iPython 4.0)

Classification vs Regression

The problem at hands is a Boolean classification supervised machine problem, as the label prediction corresponds to a pass or fails exam state.

Exploring the Data

Total number of students	395
Number of students who passed	265
Number of students who failed	130
Graduation rate of the class (%)	30
Number of features (excluding the label/target column)	67%

Preparing the Data

The following steps were executed to prepare the data for modelling, training and testing:

- Identify feature and target columns[1]
- Pre-process feature columns
- Split data into training and test sets

Training and Evaluating Models

The following three models were selected for comparison:

- Decision Tree
- Support Vector Machine
- Gaussian NB

[1] When dealing with the new data set it is good practice to assess its specific characteristics and implement the cross validation technique tailored on those very characteristics, in our case there are two main elements:

1. Our dataset is **small**.
2. Our dataset is slightly **imbalanced**. (There are more passing students than on passing students)

We could take advantage of K-fold cross validation to exploit small data sets. Even though in this case it might not be necessary, should we have to deal with heavily unbalance datasets, we could address the unbalanced nature of our data set using Stratified K-Fold and Stratified Shuffle Split Cross validation, as stratification is preserving the preserving the percentage of samples for each class.
Please see the below links for further information:
http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.StratifiedShuffleSplit.html
http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.StratifiedKFold.html

Decision Tree

Description
Decision Trees are a non-parametric supervised learning method used for classification and regression. They create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

Strengths
- Simple to understand and to interpret. Trees can be visualised.
- Requires little data preparation.
- The cost of using the tree is logarithmic in the number of data points used to train the tree.
- Can handle both numerical and categorical data.
- Can handle multi-output problems.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by Boolean logic.
- Possible to validate a model using statistical tests.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

Weaknesses
- Decision tree learners can create over-complex trees that do not generalise the data well (overfitting).
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate.

Reasons for Selection
- The dataset is relatively small, therefore the tree length should be manageable.
- The decision tree algorithm is fast and has a good explanatory power.

Performance Metrics

Performance Measures / Training size (with constant test set)	**100**	**200**	**300**
Training time	0.001	0.002	0.002
Prediction time	0.000	0.000	0.000
F1 score on training set	1.0	1.0	1.0
F1 score on test set	0.63	0.72	0.66
Average F1 score on training /test sets **Average Training/ Prediction Times**	**1.0** / 0.67 **0.002** / 0.000		

Gaussian Naive Bayes

Description
Naïve Bayes is an algorithm that is used to find a decision surface. Decision surface separate one class from another class, in a way we can generalise new never before seen data points.

Strengths
- Fast to train (single scan) and fast to classify
- Not sensitive to irrelevant features
- Handles real and discrete data
- Relative simplicity

Weaknesses
- Assumes independence of features

Reasons for Selection
- Gaussian Naive Bayes was chosen for its strength to classify data quickly, which is desired in this scenario to save on computation time.
- It also works well on noisy dataset.

Performance Metrics

Performance Measures / Training size (with constant test set)	**100**	**200**	**300**
Training time	0.001	0.001	0.001
Prediction time	0.000	0.001	0.000
F1 score on training set	0.85	0.84	0.80
F1 score on test set	0.82	0.72	0.76
Average F1 score on training /test sets **Average Training/ Prediction Times**	**0.73** / 0.76 **0.001** / 0.000		

Support Vector Machine

Description
Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers' detection.[2]

Strengths
- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

Weaknesses
- If the number of features is much greater than the number of samples, the method is likely to give poor performances.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).

Reasons for Selection
- The data might not be linear, given that there are so many features. Using an appropriate kernel like 'rbf' should enable an effectively tune of the classifier for high f1 score.

Performance Metrics

Performance Measures / Training size (with constant test set)	**100**	**200**	**300**
Training time	0.001	0.003	0.007
Prediction time	0.001	0.003	0.004
F1 score on training set	0.88	0.87	0.87
F1 score on test set	0.77	0.78	0.78
Average F1 score on training /test sets	**0.87** / 0.78		
Average Training/ Prediction Times	**0.004** / 0.002		

[2] It is not mandatory to scale features for this project. For larger datasets, feature scaling should be implemented with SVMs, performance might be affected otherwise. Please refer to http://stats.stackexchange.com/questions/65094/why-scaling-is-important-for-the-linear-svm-classification for more details.

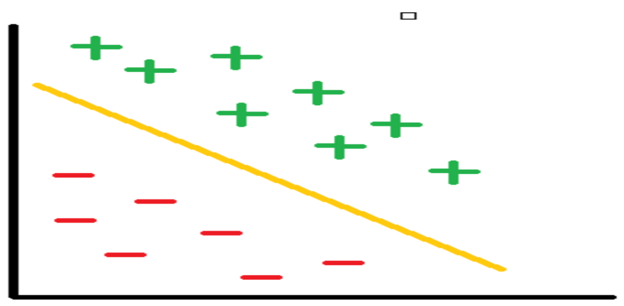
Choosing the Best Model

Looking at the relative training/test times Vs F1 (especially the training set F1) precision score, it transpires there is a trade-off between the 'Gaussian Naive Bayes' and the 'SVMs'. 'SVMs' is the slowest algorithm for training and prediction, however it also provides a very high F1 score (average 87%) across all training sizes. At the same time it does not suffer from overfitting like the 'Decision Tree' (F1=100% on the training data).
The 'Gaussian Naive Bayes' is the fastest running algorithm, however it provides slightly worst overall F1 training scores (around 73%), compared to the 'SVMs' algorithm.

Therefore, for this very small dataset, I would argue to select the 'SVMs' algorithm, as it provides better accuracy. The time to completion does not matter in this instance, as it remains below 10ms, which very fast running in any case. However, this would be a problem in the case of very large datasets (GB, PB or ZB). The 'SVMs' is on average 4 times slower than the 'Gaussian Naive Bayes' algorithm for only an extra 16% of accuracy, calculated as (0.73-0.87)/0.87. This means that if the 'Gaussian Naive Bayes' needed 6h to complete, the 'SVMs' would need 24h. This could be an issue for business application that run overnight for example.

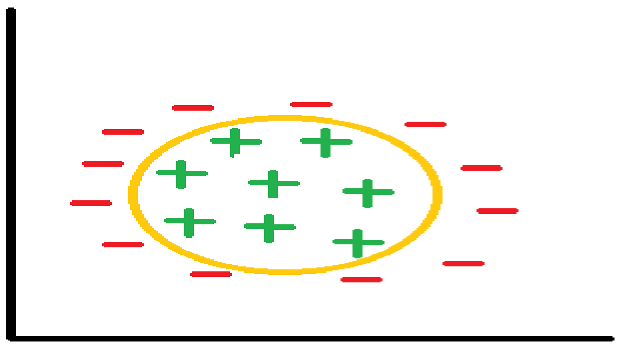
<u>High level description of the SVM model</u>

General Description
SVMs can be used for classification, regression and outliers' detection. The following section will focus on the classification case. SVMs algorithms are capable of performing multi class classification on linearly separable data in a dataset, as shown in the simplified Graph1 below.



Graph1 - Linear separable data divided by a hyperplane.

The power of SVMs models come with their ability to also classify data that do display an initial linearly separable behaviour, as shown by Graph2 below. In this case, the Kernel trick is used to find the best fit for the separator. Graph2 shows the case of a quadratic separator, but other shapes could be found to suit the dataset.



Graph2 - Data divided by a quadratic separator.

In other words, given labelled training data (*supervised learning*), the algorithm outputs an optimal hyperplane which categorizes new examples.

Training Description
A separation line (or shape) is considered as 'bad' if it passes too close to the points. This would have the effect producing a model too close to the training and consequently not generalising correctly with new data (a.k.a. overfitting). The goal is to find a separator passing as far as possible from all points. The operation of the SVM algorithm is based on finding the separator that gives the largest minimum distance (a.k.a. margin) to the training examples.

Prediction Description
The new data classification prediction the SVMs model (i.e. the separator) to find the group where the new data should belong to. Once a prediction is made, the analyst can fine tune the SVMs algorithm, for example by changing the model parameters to improve the accuracy of the prediction or the running time.

Fine Tuning the Model

It appears that changing the kernel of the SVM algorithm from the default 'rbf' to 'sigmoid' and the introduction of the GridSearchCV reduced the 'SVMs' algorithm F1 training accuracy to 0.81 from an average of 0.87, so loss of 7% of accuracy. However, the training was halved (from 0.007 to 0.004) [3][4]

Performance Measures / Training size (with constant test set)	**Entire set**
Training time	0.004
Prediction time	0.002
F1 score on training set	0.81
F1 score on test set	0.77

[3] We could have used a stratified shuffle split data-split. It preserves the percentage of samples for each class and combines it with cross validation. This could be extremely useful when the dataset is strongly **imbalanced** towards one of the two target labels. Please see the below link for further information: http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.StratifiedShuffleSplit.html
In this example a support vector classifier is used, any other classifier would do, please remind to change the parameters adapting them to the specific classifier you intend to deploy.

```
from sklearn.grid_search import GridSearchCV
from sklearn.svm import SVC
from sklearn.cross_validation import StratifiedShuffleSplit
from sklearn.metrics import f1_score
from sklearn.metrics import make_scorer

f1_scorer = make_scorer(f1_score, pos_label="yes")

# Some SVC parameters
parameters = { 'C' : [ 0.1, 1, 10, 100, 1000 ], 'gamma' : [ 0.0001, 0.001, 0.1, 10, 100 ] }
# 1. Let's build a stratified shuffle object
ssscv = StratifiedShuffleSplit( y_train, n_iter=10, test_size=0.1)
# 2. Let's now we pass the object and the parameters to grid search
grid = GridSearchCV( SVC(), parameters, cv = ssscv , scoring=f1_scorer)

grid.fit( X_train, y_train ) # 3. Let's fit it
best = grid.best_estimator_ # 4. Let's reteieve the best estimator found
y_pred = best.predict( X_test ) # 5. Let's make predictions!

print "F1 score: {}".format( f1_score( y_test, y_pred, pos_label = 'yes' ))
print "Best params: {}".format( grid.best_params_ )
```

[4] We could have gone well beyond grid search and implement 'pipelines' where the whole machine learning process becomes 'grid-searchable'. Then we can parameterize and search the whole process though cross validation. Please refer to below link for further information: http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html
We could have also tried several algorithms automatically too. Please refer to below link for further information: http://zacstewart.com/2014/08/05/pipelines-of-featureunions-of-pipelines.html