Project 2: Supervised Learning

Building a Student Intervention System

1. Classification vs Regression

This is a classification problem. We are making a model that will predict whether a student will pass or fail their high-school exam. The output of the model will therefore be true or false, in other words a discrete variable. If this had been a regression problem we would have created a model that predicts continuous variables.

2. Exploring the Data

The following table summarises some basic properties and content of the dataset.

Total number of students	**395**
Number of students who passed	**265**
Number of students who failed	**130**
Graduation rate of the class (%)	**67.09 %**
Number of features	**30 (48 after creating dummy variables)**

To summarise we have a fairly small dataset with a fairly high number of features compared to

4. Training and Evaluating Models

When selecting the models we have been given a set of constraints that we must take into consideration when selecting the models.

- The project has limited resources and will pay based on the CPU and memory usage of the model. A model with a low computational complexity and DSPACE (the total amount of memory used) to train the model and generate the predictions.
- The number of training examples needed to make reasonable predictions. Smaller training sets are preferred over large ones.
- The models performance, measured using the F1 score. The F1 score can be interpreted as the weighted average of the precision and recall, where F1 score reaches its best value at 1 and worst score at 0.
- The model must be available in scikit-learn

As this is a classification problem we shall look closer at some of the classification models available in scikit-learn.

Nearest Neighbour Classification is a type of instance-based learning. It does not attempt to construct an internal model, but simply stores the entire training set. K-nearest-neighbour, which is a nearest neighbour classification model has a computational complexity of O(1) and making predictions has a complexity of $O(k * n_{samples} * n_{feature})$ where k is the number of neighbours. This means that the model is not as memory efficient as other models that constructs a more compact model and predictions are generally slow as you must compute the distances to all other points. This method is also know to be less effective in high-dimensional spaces when we have sparse data due to the *curse of dimensionality.*

Support Vector Machines (SVMs) are one of the most popular classification models. It's advantages are that they are effective in high-dimensional spaces and only uses a subset of the training points in the decision function (the support vectors), meaning that it is memory efficient. One important thing to keep in mind when working with SVMs is that both the computational and storage requirements increase rapidly with the number of training examples. The Quadratic Programming solver used by scikit-learn has a computational complexity between $O(n_{features} * n^2_{samples})$ and $O(n_{features} * n^3_{samples})$. A linear SVM has a prediction complexity of $O(n_{features})$. SVMs, and in particular linear SVMs are less prone to overfitting as they attempt to find the line with the largest margin to all data-points.

Decision Trees creates a model based on simple rules. It's advantages are that they are easy to interpret (white-box), and the scikit-learn implementation for constructing the decision tree has a complexity of $O(n_{features} * n_{samples} *\log(n_{samples}))$ and the cost of making predictions is $O(\log n)$. The main disadvantages of decisions trees is that they are prone to overfitting, they are unstable and they have some issues representing some concepts. The **Random Forest Classifier** can mitigate some of problems related to overfitting and the instability of the Decision Trees. The Random Forest Classifier is an ensemble method based og bagging (Bootstrap aggregating). The ensemble consists of multiple trees (which make up a forest) where each tree is built from a sample drawn with replacement and a multitude of decision trees are created. The scikit-learn implementation combines the classifiers by averaging their probabilistic predictions, instead of using voting. The complexity of constructing the Decision Tree classifiers are the same, however the number of samples will be lower (as we draw samples) in addition to that we select features at random. On the other hand this method requires us to construct multiple decisions trees meaning that the computational complexity will be $O(M(n_{features} * n_{samples} *\log(n_{samples})))$, where M is the number of estimators. There are also some other processes such as the randomisation process which i have not factored in here. **AdaBoost** is a popular boosting algorithm that fit a sequence of weak learners (such as a Decision Tree) on repeatedly modified versions of the data. The predictions from all these learners are combined through a weighted majority vote to produce the final prediction. One of the main advantages of boosting is that it has been shown to be resistant to overfitting, in addition the models created are easy to interpret. The time complexity of AdaBoost is $O(M * f)$ where M is the number estimators and f is the complexity of the weak learner used.

Multinomial Naive Bayes is a method that is based on applying Bayes' theorem with the "naive" assumption between every pair of features. There are multiple Naive Bayes implementations in scikit-learn. As most of our features are discrete (with the exception of age) the multinomial or Bernoulli event models are the best fit for this particular problem. Bernoulli assumes that all of our features are boolean, which is not correct in our case. I have therefore chosen Multinomial Naive Bayes for this problem. The main advantages of naive bases methods are that they are highly scalable and require little training data, additionally it also tends to do very well on high-dimensional datasets based on e.g. bag of words representation of each observation. The theoretical computational complexity for learning a naive bases classifier is $O(n_{samples} * n_{features})$ and the space complexity is $O(n_{features} * q * r)$ where q is the number of values for each feature and r is the number of classes.

Based on the summaries above i would like to perform further experiments on SMVs, Multinomial Naive Bayes and Random Forest Classifier. Multinomial Naive Bayes has both a low computational complexity and space requirements in addition to requiring little training data, but with the downside being that it assumes all variables to be independent. This might be true for text documents, but it is most probably not the case for this dataset. I decided to choose SMV and RandomForestClassifier even though their computational complexities are higher than those than e.g. for decision trees, but due to the low size of our training set this will not make a big difference in practice. Both SVMs and Random Forest Classifiers are popular models and are used on a wide-array of real world problems.

Initial Experiments

The following tables shows the results from my initial experiments. I wanted to run multiple experiments and average the results, however as the supplied code was not written to reusable to for this purpose i assumed it was not necessary.

Random Forest Classifier

The following table shows my benchmark results using a random forest classifier with default settings. The default settings for the random forest classifier uses 10 estimators.

Training set size	**100**	**200**	**300**
Training time	0.019s	0.16s	0.017s
Prediction time (Training)	0.001s	0.001s	0.001s
Prediction time (Test)	0.001s	0.001s	0.001s
F1 score (Training)	1,0	1,0	0,99
F1 score (Test)	0,683	0,682	0,774

Support Vector Machine

The following table shows my benchmark results using a linear SVM with the default settings.

Training set size	**100**	**200**	**300**
Training time	0.002s	0.003s	0.005s
Prediction time (Training)	0.001s	0.002s	0.004s
Prediction time (Test)	0.001s	0.001s	0.001s
F1 score (Training)	0,884	0,884	0,876
F1 score (Test)	0,762	0,8	0,8

Multinomial Naive Bayes

The following table shows my benchmark results using a Multinomial Naive Bayes classifier using default settings.

Training set size	**100**	**200**	**300**
Training time	0.001s	0.001s	0.001s
Prediction time (Training)	0.000s	0.000s	0.000s
Prediction time (Test)	0.000s	0.000s	0.000s
F1 score (Training)	0,828	0,811	0,794
F1 score (Test)	0,667	0,752	0,794

Based on our preliminary results we can see that SVMs have the best performance on the test set for all training set sizes, however Multinomial Naive Bayes closes the gap as it is trained with more training examples. With regards to training and prediction time Multinomial Naive Bayes win by a large margin both in training time and time taken to make predictions. However, the training and prediction times are so low that it would hardly make a difference on the server load.

5. Choosing the Best Model

My initial experience showed that the SVM model is the best fit for this particular task. The SVM machine model outperformed all the other models in identifying the students at risk for all training set sizes in addition to have a very reasonable running time of 5 milliseconds using the entire training set and 1 millisecond to predict the outcome for all students in the test set. The Multinomial Naive Bayes model had a lower training and prediction time, but a lower F1 score. I would therefore say that correctly identifying the the students that are at risk and get closer to the goal of a 95% graduation rate takes precedence over saving a few milliseconds of running time and that the support vector machine would be the best model choice in this case.

Machine learning can be seen as a class of models with tuneable parameter such that the model automatically can adjust its ability to adapt to previously seen data. SVMs are a powerful supervised learning algorithm used for both classification and regression. SVMs are a discriminative classifier which means that they draw boundaries between different classes of data. When separating different classes of data there are multiple boundaries that can separate the two classes. SVMs tries to find the boundary that maximises the margin to both classes of data. Another interesting aspect of SVMs is that not all points are equally important, it is only the points supporting the model (the support vectors) and decide the decision function. SVMs get more interesting in conjunctions with kernels. Kernels can be seen as a transformation your input data in i way such that you can process it more easily. This typically involves taking a non-separable low-dimensional dataset and transforming it into a high-dimensional dataset that is more easily separable. When classifying a new example the SVM basically checks which side of the decision function the new data points is and classifies it accordingly.

My final model is a SVM function with a RBF kernel with a c setting of 100 and a gamma of 0.0001. The c parameter controls the tradeoff between misclassification and simplicity. A high c value means that you highly penalise misclassifications while sacrificing simplicity. The gamma value controls the flexibility of the model. A low gamma value means that you model will have limited flexibility. The model had a mean F1 score of 0.82049 doing 10-fold cross validation on the entire training set and a training time of 0.005s on the training set with 300 examples and a prediction time of 0.001s.