1. **Classification vs Regression**

Given problem is a *classification* problem as the output contains one of 2 classes namely pass: yes or no

2. **Exploring the Data**

Following are the characteristics 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%

3. **Training and testing on 3 different machine learning models**

I chose following three classifiers for this exercise. Since, I am going to choose the best classifier for the problem empirically, I wanted to try reasonably fit algorithms. GaussianNB was eliminated from the list because it was so simple and couldn't represent the data well.

1. Decision Tree Classifier
2. Bagging Classifier
3. Support Vector Classifier

As we see that the number of students passed is more than number of students failed; the samples are not evenly distributed among the output classes. So we need to use stratified cross validation which makes sure that subsets of data are in similar ratio of classes as the whole data set.

The test results of each of the classifier mentioned above are presented below.

Note: The F1 scores listed below are for test sets set aside from the training sets.

Decision Tree Classifier (max_depth =1)

Decision Tree Classifier works by using a series of if-then-else rules as nodes in a tree and uses them to classify a data point. It is easy to visualize its working as the model is actually a tree of rules and intuitive to understand. If not controlled properly, decision trees are prone to overfitting easily as a tree with all the data points in the leaf nodes will work pretty well on the training data but will fail to generalize.

I chose this model primarily because it is a white box model i.e. we can easily reason about the output it produced by visualizing the result produced. Also it works well with numerical and categorical data which is the case with this example.

Training Size	Training Time	Prediction Time	Training F1 Score	Testing F1 Score
100	0.000	0.000	0.82	0.82
200	0.000	0.000	0.84	0.82
300	0.001	0.000	0.81	0.82

Support Vector Classifier

Support vector classifier works by arriving at decision boundaries to classify the data. It is memory efficient because it uses only a set of data points called as support vectors to arrive at the decision boundary. It is also a large margin classifier and so is robust to noise in the data. Using kernels, it is possible to arrive at non-linear decision boundaries. Generally, the important drawback of a support vector classifier is it takes long to get trained especially for decision surfaces that are not linear and complex algorithms involved in training.

The memory efficiency of support vector machines is the reason I chose to try it. Also, the versatility of SVMs in terms of the decision surfaces available was another factor.

Training Size	Training Time	Prediction Time	Training F1 Score	Testing F1 Score
100	0.002	0.001	0.882	0.800
200	0.005	0.001	0.824	0.810
300	0.008	0.002	0.832	0.800

Bagging Classifier

Bagging classifier is a type of ensemble classifier. Ensemble means using more than one one model to classify the data points. This particular type of ensemble classifier works by training different decision trees on random subsets of the data and using all of them during final classification. This avoids overfitting which can occur while using a single tree. The disadvantage from the given problem perspective is that it is computationally expensive than using a single tree.

I wanted to evaluate how ensemble classifiers perform with respect to the performance metric and computational resources consumed and so went with Bagging Classifier.

Training Size	Training Time	Prediction Time	Training F1 Score	Testing F1 Score
100	0.014	0.001	0.97	0.7438
200	0.013	0.001	0.98	0.857
300	0.015	0.002	0.99	0.894

4. Choosing and training the best model

From above results, I choose Decision Tree Classifier as desired model for this classification problem. It performs reasonably well as can be seen from the F1 score. Also, looking at the computation times (both training and prediction times), it doesn't seem to be computationally expensive. Other models tested seem to produce results with better accuracy but are computationally expensive and I guess we can tolerate some error for significant savings in computing power during training as well as prediction times. [The training and prediction don't vary significantly for different models because of small size of data. But the argument holds good for data of significant sizes.] Since the available data is of small size and the output classes are not evenly distributed, we use StratifiedKFold cross validation to use the examples effectively.

Here is how decision trees work:

a. During the training phase, the decision tree algorithm learns trees by constructing them top-down starting with a basic question. Which feature in the sample should be tested at the root of the tree? It tries to pick the feature which by itself classifies the data well using some statistical tests. It then creates descendent nodes (one for each class) along with corresponding samples. The process repeats at each node till some end condition (in terms of depth of tree or minimum number of samples in each class) is met.
b. The end result of the algorithm post training is some optimal arrangement of nodes of the tree and their decision rules that maximizes some performance measure (F1 score in our case).
c. During prediction, for a given sample, the algorithm walks down the tree along the path governed by the values of the input features of the sample and decision rules.
d. This continues till the path terminates at the leaf of the tree and value derived from the corresponding leaf is the result of the prediction.

Fine tuning the best model chosen:

The model doesn't need any tuning in my opinion. Tuning, as I understand, is used to arrive at some optimal value of the hyper parameters of the model that gives the maximum possible performance measure. For decision trees, max_depth is important hyper parameter that needs tuning. In one of the earlier steps, I have performed cross validation search for a range of decision tree depths and found that the tree has its best score for max_depth 1. But for double checking, I used GridSearchCV with depths 1, 2 and 3. The expected result from grid search was max_depth 1. And it came up with the max depth value of 1.

The cross validation procedures (by cross_val_score and GridSearchCV) applied were, StratifiedKFold, the default option scikit-learn uses when it is classification problem and labels of the data are multi class.

F1 score for the final model: 0.82