## 1. Classification vs 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?

**Answer:**
The type of supervised learning problem depends on the output that we want to predict. In the case of regression, the output that we want to predict is a continuous value (real numeric value). In the case of classification, the output that we want to predict is categorical (discrete value). In this project, we want to predict if a student will pass or fail, that is, to predict which category will the student belong to. Hence, this problem is a classification problem.

## 2. Exploring the Data

Can you find out the following facts about the dataset?

**Answer:**

- 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 (excluding the label/target column) : 31

## 3. Preparing the Data

Execute the following steps to prepare the data for modeling, training and testing:

Answer:

- **Identify feature and target columns**
  **Feature columns:**
  ['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu', 'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime', 'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Walc', 'health', 'absences']
  **Target column:**
  passed

- **Preprocess feature columns**
  ['school_GP', 'school_MS', 'sex_F', 'sex_M', 'age', 'address_R', 'address_U', 'famsize_GT3', 'famsize_LE3', 'Pstatus_A', 'Pstatus_T', 'Medu', 'Fedu', 'Mjob_at_home', 'Mjob_health', 'Mjob_other', 'Mjob_services', 'Mjob_teacher', 'Fjob_at_home', 'Fjob_health', 'Fjob_other', 'Fjob_services', 'Fjob_teacher', 'reason_course', 'reason_home', 'reason_other', 'reason_reputation', 'guardian_father', 'guardian_mother', 'guardian_other', 'traveltime', 'studytime',

'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Walc', 'health', 'absences']

- **Split data into training and test sets**
  Training set: 300 samples
  Test set: 95 samples

## 4. Training and Evaluating Models

Choose 3 supervised learning models that are available in scikit-learn, and appropriate for this problem. For each model:

### A. Decision Tree:

- What are the general applications of this model? What are its strengths and weaknesses?
  **Answer:**
  Decision Trees are more commonly used in Classification problems. But algorithms like CART are also used in Regression problems. Decision tree is basically a tree data structure where feature variables reside at internal nodes and split occurs at these nodes based on the value of that node. The target variable/s reside at the leaf of this decision tree.

  Strength of decision trees lies in its simplicity. It is easy to imagine, to draw, and to analyze which makes it a great to understand the relation between the features and their importance in the classification problem. It is also computationally less expensive compared to other classification algorithms to carry out predictions as the prediction time depends on the maximum depth of the tree, which is logarithm in the number of training points.

  One of the most common drawback of decision tree is its propensity to overfit the data, but this can be tackled by performing feature selection and reduction, or by setting a limit on maximum depth of the tree. They are also known to be biased towards the class that as more training examples. This problem can be tackled with balancing the class weights or giving a balanced dataset to the classifier.

- Given what you know about the data so far, why did you choose this model to apply?
  **Answer:**
  After analyzing the data I realized that most of the features were categorical, hence they would be ideal to fit in nodes of decision tree. Also since the classification problem we are addressing is a simple 2-class problem, I thought that decision tree would be a simple and efficient algorithm for this.
  Then, I applied chi-square test on my training data and realized that many of the features had a very low score and the target value was hardly dependent on them. This made me realize that if I limit the depth of the tree or select best features for training, then my algorithm would do well.
  Hence, I choose decision tree as my first algorithm for this problem. I perform feature selection to select 8 best features based on their chi-square score, before training on the dataset. To address the class imbalance problem, I added the class-weight parameter to the training classifier.

- Fit this model to the training data, try to predict labels (for both training and test sets), and measure the F1 score. Repeat this process with different training set sizes (100, 200, 300), keeping test set constant.

- Produce a table showing training time, prediction time, F1 score on training set and F1 score on test set, for each training set size.

  **Answer:**

| | Training set size | | |
|---|---|---|---|
| | 100 | 200 | 300 |
| Training time (secs) | 0.001 | 0.001 | 0.002 |
| Prediction time (secs) | 0 | 0 | 0 |
| F1 score for training set | 0.953 | 0.957 | 0.942 |
| F1 score for test set | 0.698 | 0.794 | 0.809 |

## B. SVM

- What are the general applications of this model? What are its strengths and weaknesses?

  **Answer:**

  Support Vector Machines (SVM) are used in classification problems where the number of dimensions are large. SVM basically finds a hyper plane that divides the data according to the target variable. To find such a hyper plane, SVM maps the data to a higher dimensional space. It then finds support vectors near the hyper plane which represent that class. It can be used in Regression as well as Outlier detection.

  Strength of SVM lies in its ability to work in high dimensional spaces. If there are a huge number of features, even greater than the number of examples, SVM can still learn from it. SVM can work with different Kernel Function, for example, linear, rbf, polynomial. Hence, it can represent various kinds of dataset.

  One common drawback of SVM I feel is the lack of transparency unlike Decision Tree which is quite simple to visualize. I believe it is also computationally expensive, the fitting time for SVC is more than quadratic. It is also not appropriate to use for multi-class classification problems.

  Given what you know about the data so far, why did you choose this model to apply?

  **Answer:**

  After transforming the features to categorical variables, the number of features increased and the number of training examples remained the same. So I thought SVM could tackle this case of high dimensionality with limited training data. Hence, I chose to apply this algorithm.

  I was confused between choosing the kernel for SVM. I decided to use RBF kernel, because the number of dimensions are not in hundreds or thousands, hence there will be a need to map the data to higher dimensional space. Although, linear kernel will be faster but the predictive power of RBF kernel is more.

I do not perform feature selection as the intention behind using SVM was to take advantage of the fact that we have a high dimension feature space.

- Fit this model to the training data, try to predict labels (for both training and test sets), and measure the F1 score. Repeat this process with different training set sizes (100, 200, 300), keeping test set constant.
- Produce a table showing training time, prediction time, F1 score on training set and F1 score on test set, for each training set size.

  **Answer:**

  with RBF kernel and default values of C and gamma

| | Training set size | | |
|---|---|---|---|
| | 100 | 200 | 300 |
| Training time (secs) | 0.001 | 0.004 | 0.009 |
| Prediction time (secs) | 0.001 | 0.003 | 0.007 |
| F1 score for training set | 0.877 | 0.868 | 0.876 |
| F1 score for test set | 0.774 | 0.781 | 0.784 |

## C. AdaBoostClassifier with Decision Tree as the base estimator

- What are the general applications of this model? What are its strengths and weaknesses?

  **Answer:**

  AdaBoost, i.e. Adaptive Boosting is a kind of an Ensemble method. Ensemble methods use several estimators of a given learning algorithm to make predictions.

  They are of 2 types:

  Averaging methods – These methods take average of each estimator's predictions. e.g. Random Forest, Bagging.

  Boosting methods – In these methods, estimators build sequentially to reduce bias and they use several weak learners to build a robust ensemble. e.g. Adaptive Boosting, Gradient Tree Boosting.

  Adaptive Boosting – AdaBoostClassifier learns on the dataset repeatedly using the estimator (like Decision Tree), and changes the weights of incorrectly classified examples so that the next iteration of boosting focuses more on such wrongly classified examples.

  The main strength of Adaptive Boosting is that it is very less susceptible to overfitting as compared to other learning algorithms. It requires a very simple weak learner that works better than random guessing, hence there is less parameter tweaking required. It is also computationally efficient.

  One of the drawback of AdaBoost is that if the weak learner is too complex, then the ensemble tends to overfit. AdaBoost is also known to be susceptible to uniform noise.

Given what you know about the data so far, why did you choose this model to apply?

**Answer:**

After applying the Decision Tree Classifier on this data, I realized that it was giving better results than SVM and Nearest Neighbors algorithm. So I thought, to reduce overfitting and make the learner more robust, I could use decision tree classifier but with an ensemble. Moreover, AdaBoost is known to perform really well on 2-class classification, is computationally less expensive, and doesn't require much parameter tuning, I decided to go ahead with it.

Fit this model to the training data, try to predict labels (for both training and test sets), and measure the F1 score. Repeat this process with different training set sizes (100, 200, 300), keeping test set constant.

- Produce a table showing training time, prediction time, F1 score on training set and F1 score on test set, for each training set size.

  **Answer:**

| | Training set size | | |
|---|---|---|---|
| | 100 | 200 | 300 |
| Training time (secs) | 0.107 | 0.114 | 0.126 |
| Prediction time (secs) | 0.005 | 0.006 | 0.009 |
| F1 score for training set | 0.953 | 0.956 | 0.942 |
| F1 score for test set | 0.634 | 0.772 | 0.806 |

## 5. Choosing the Best Model

Based on the experiments you performed earlier, in 2-3 paragraphs explain to the board of supervisors what single model you choose as the best model. Which model has the best test F1 score and time efficiency? Which model is generally the most appropriate based on the available data, limited resources, cost, and performance? Please directly compare and contrast the numerical values recored to make your case.

In 1-3 paragraphs explain to the board of supervisors in layman's terms how the final model chosen is supposed to work (for example if you chose a decision tree or support vector machine, how does it learn to make a prediction).

Fine-tune the model. Use gridsearch with at least one important parameter tuned and with at least 3 settings. Use the entire training set for this.

What is the model's final F1 score?

**Answer:**

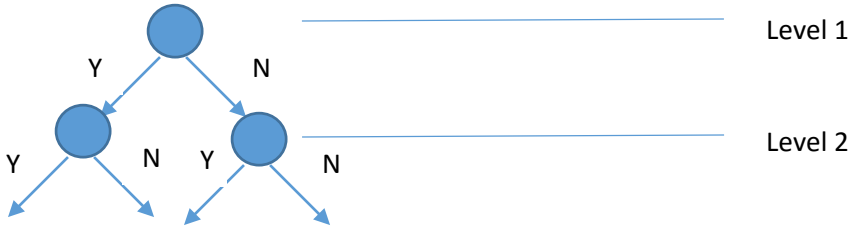
Based on the three classifiers listed and tested above, Decision Tree and Adaptive Boosting classifier perform quite well. Hence, the final decision for me was choosing between these two classifiers. The training and test set accuracy with full dataset are very close for these two algorithms.

The Decision tree classifier gives the accuracy of 94.2% on the training set and 80.9% on the test set.
The Adaptive Boosting classifier gives the accuracy of 94.2% on the training set and 80.6% on the test set.

In my previous submission, I argued that Adaptive Boosting would be a better classifier since it is an ensemble method built over Decision Tree. I also achieved a decent accuracy level using grid search on a few parameters. Though, I made an error in analyzing their running time. I did not realize that if the training examples increase by a factor of thousand or more then Adaptive Boosting would take a lot of time to learn. Hence, I decided to pick the computationally efficient Decision Tree as the final model due to its speed and accuracy.

Although Decision Trees tend to overfit with a lot of features, using GridSearchCV I can pick the best parameters to avoid such overfitting.

To state simply, Decision Trees are a list of if-else-then rules, and the order in which these rules come in the list is important. Imagine these rules to be arranged one below the other in a tree format.




So say you are given an email and you want to find out if this email is a spam. What Decision Tree does is, it passes this email through this tree from top to bottom, where each node poses a yes/no question and based on the answer, the email goes to the next level, until it reaches the leaf where it gets the label of “spam” or “not spam”. This is how classification of an email happens on a simple decision tree.

Each node is nothing but a decision-question on a particular feature of the training data.
For example in our email data, there could be 2 features:
1. The word “Lottery” exists in the email.
2. The email is sent by an unknown sender.

Level 1 could ask if the word “Lottery” exists in the mail. If yes, it goes to Level 2 which asks if the mail was sent by an unknown ID. As you go further down the tree and reach the leaf, you come to know if the email was a spam or not.

Now before classification can happen, Decision Tree has to be constructed and the order in which the levels (nodes) are present in the tree is quite important. What we want is the most important node to be present at the top. The importance of a node is captured through various functions such as information gain, gini impurity, etc. What these functions try to find out is, which node gives the best split (such that all the children have almost equal number of data points in them). This function finds out the node that maximizes the information you gain after splitting on that particular node. Going this way, nodes are then sorted according to

decreasing order of information gain (highest information gain being at the root), and the tree is constructed in this order. Hence, the decision tree eventually looks like a set of If-else-then rules starting from root node to the leaf node.

Now, when you want to predict whether a new email is a spam or not, you pass it through the tree, starting from the root and following the if-else-then rules. This will lead you to a leaf that will either mark the email as spam or not spam. Hence, after training is done, the prediction takes very little time.

Moving ahead with Decision Tree Classifier and fine tuning its parameters, I realized that the major improvement could be in the minimum number of samples required to make a split. I did not tune max-depth of the tree since I had already performed feature selection and selected 7 best features based on their chi-square values. I also use class_weight parameter since ‘yes’ class dominates ‘no’ class (number of examples in ‘yes’ class are almost double than number of examples in the ‘no’ class)

Finally, I was confused about using gini or entropy as the splitting criterion. I performed grid search and found that entropy index performs better. I did a bit research on this and found on a couple of forums that gini index is better for reducing misclassification and entropy is preferable for exploratory analysis. I would like to know how to pick the splitting criterion then.

**'min_samples_split':[2,3,4,5],**
**'class_weight':[{'yes':1},{'yes':2}],**
**'splitter':['best','random'],**
**'criterion':['entropy','gini']**

After running Grid Search over the parameter grid, I get the best parameters as:

**'min_samples_split': 2,**
**'splitter': 'random',**
**'criterion': 'entropy',**
**'class_weight': {'yes': 2}**

The final accuracy of the model on the **training set is: 94.2% and for the test set it is: 84.1%.**
It is more than 4% than the default setting of the classifier.