Project 2: Supervised Learning

Building a Student Intervention System

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: This is a classification problem. In regression, the output is continous; whereas in classification, it is discrete. In this project, the goal is to find whether a student "Needs Intervention" or "Does NOT Need Intervention", constituting a binary classification problem.

2. Exploring the Data

Let's go ahead and read in the student dataset first.

*To execute a code cell, click inside it and press **Shift+Enter**.*

In [1]:
```
# Import libraries
import numpy as np
import pandas as pd
```

In [2]:
```
# Read student data
student_data = pd.read_csv("student-data.csv")
print "Student data read successfully!"
# Note: The last column 'passed' is the target/label, all other are feat
ure columns
```

```
Student data read successfully!
```

Now, can you find out the following facts about the dataset?

- Total number of students
- Number of students who passed
- Number of students who failed
- Graduation rate of the class (%)
- Number of features

*Use the code block below to compute these values. Instructions/steps are marked using **TODO**s.*

In [3]:
```
# TODO: Compute desired values - replace each '?' with an appropriate ex
pression/function call
n_students = len(student_data)
n_features = len(student_data.columns)-1
n_passed = len(student_data[student_data['passed']=='yes'])
n_failed = len(student_data[student_data['passed']=='no'])
grad_rate = (float(n_passed)/float(n_students))*100
print "Total number of students: {}".format(n_students)
print "Number of students who passed: {}".format(n_passed)
print "Number of students who failed: {}".format(n_failed)
print "Graduation rate of the class: {:.2f}%".format(grad_rate)
print "Number of features: {}".format(n_features)
```

```
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
```

3. Preparing the Data

In this section, we will prepare the data for modeling, training and testing.

Identify feature and target columns

It is often the case that the data you obtain contains non-numeric features. This can be a problem, as most machine learning algorithms expect numeric data to perform computations with.

Let's first separate our data into feature and target columns, and see if any features are non-numeric.
Note: For this dataset, the last column (`'passed'`) is the target or label we are trying to predict.

In [4]:
```
# Extract feature (X) and target (y) columns
feature_cols = list(student_data.columns[:-1])  # all columns but last a
re features
target_col = student_data.columns[-1]  # last column is the target/label
print "Feature column(s):-\n{}".format(feature_cols)
print "Target column: {}".format(target_col)

X_all = student_data[feature_cols]  # feature values for all students
y_all = student_data[target_col]  # corresponding targets/labels
print "\nFeature values:-"
print X_all.head()  # print the first 5 rows
```

```
Feature column(s):-
['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fed
u', 'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime', 'f
ailures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'high
er', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Wa
lc', 'health', 'absences']
Target column: passed

Feature values:-
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjo
b  \
0     GP   F   18       U     GT3       A     4     4  at_home   teache
r
1     GP   F   17       U     GT3       T     1     1  at_home     othe
r
2     GP   F   15       U     LE3       T     1     1  at_home     othe
r
3     GP   F   15       U     GT3       T     4     2   health  service
s
4     GP   F   16       U     GT3       T     3     3    other     othe
r

    ...    higher internet  romantic  famrel  freetime goout Dalc Walc
health  \
0   ...       yes       no        no       4         3     4    1    1
3
1   ...       yes      yes        no       5         3     3    1    1
3
2   ...       yes      yes        no       4         3     2    2    3
3
3   ...       yes      yes       yes       3         2     2    1    1
5
4   ...       yes       no        no       4         3     2    1    2
5

  absences
0        6
1        4
2       10
3        2
4        4

[5 rows x 30 columns]
```

Preprocess feature columns

As you can see, there are several non-numeric columns that need to be converted! Many of them are simply `yes`/`no`, e.g. `internet`. These can be reasonably converted into `1`/`0` (binary) values.

Other columns, like `Mjob` and `Fjob`, have more than two values, and are known as *categorical variables*. The recommended way to handle such a column is to create as many columns as possible values (e.g. `Fjob_teacher`, `Fjob_other`, `Fjob_services`, etc.), and assign a `1` to one of them and `0` to all others.

These generated columns are sometimes called *dummy variables*, and we will use the `pandas.get_dummies()` (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html?highlight=get_dummies#pandas.get_dummies) function to perform this transformation.

In [5]:
```
# Preprocess feature columns
def preprocess_features(X):
    outX = pd.DataFrame(index=X.index)  # output dataframe, initially em
pty

    # Check each column
    for col, col_data in X.iteritems():
        # If data type is non-numeric, try to replace all yes/no values
with 1/0
        if col_data.dtype == object:
            col_data = col_data.replace(['yes', 'no'], [1, 0])
        # Note: This should change the data type for yes/no columns to i
nt

        # If still non-numeric, convert to one or more dummy variables
        if col_data.dtype == object:
            col_data = pd.get_dummies(col_data, prefix=col)  # e.g. 'sch
ool' => 'school_GP', 'school_MS'

        outX = outX.join(col_data)  # collect column(s) in output datafr
ame

    return outX

X_all = preprocess_features(X_all)
print "Processed feature columns ({}):-\n{}".format(len(X_all.columns),
list(X_all.columns))
```

```
Processed feature columns (48):-
['school_GP', 'school_MS', 'sex_F', 'sex_M', 'age', 'address_R', 'addre
ss_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_serv
ices', 'Fjob_teacher', 'reason_course', 'reason_home', 'reason_other',
'reason_reputation', 'guardian_father', 'guardian_mother', 'guardian_ot
her', 'traveltime', 'studytime', 'failures', 'schoolsup', 'famsup', 'pa
id', 'activities', 'nursery', 'higher', 'internet', 'romantic', 'famre
l', 'freetime', 'goout', 'Dalc', 'Walc', 'health', 'absences']
```

Split data into training and test sets

So far, we have converted all *categorical* features into numeric values. In this next step, we split the data (both features and corresponding labels) into training and test sets.

In [6]:
```
# First, decide how many training vs test samples you want
num_all = student_data.shape[0]  # same as len(student_data)
num_train = 300  # about 75% of the data
num_test = (num_all - num_train)
perc = float(num_test) / float(num_all)
from sklearn.cross_validation import train_test_split
# TODO: Then, select features (X) and corresponding labels (y) for the t
raining and test sets
# Note: Shuffle the data or randomly select samples to avoid any bias du
e to ordering in the dataset
#X_train = X_all.sample(n=num_train)
#y_train = y_all.sample(n=num_train)
#X_test = X_all.sample(n=num_test)
#y_test = y_all.sample(n=num_test)
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_s
ize = perc, random_state=0)
X_train_200 = X_train.sample(n=200, random_state=40)
y_train_200 = y_train.sample(n=200, random_state=40)
X_train_100 = X_train.sample(n=100, random_state=42)
y_train_100 = y_train.sample(n=100, random_state=42)
print "Training set: {} samples".format(X_train.shape[0])
print "Test set: {} samples".format(X_test.shape[0])
# Note: If you need a validation set, extract it from within training da
ta
```

```
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:

- What are the general applications of this model? What are its strengths and weaknesses?
- Given what you know about the data so far, why did you choose this model to apply?
- 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.

Note: You need to produce 3 such tables - one for each model.

Answers:

- In this project, the goal is to find whether a student "Needs Intervention" or "Does NOT Need Intervention", constituting a binary classification problem. The common/general application of all the techniques used below are for classification problems.

  Model 1- Naive Bayes: This is a classification technique based on Bayes theorem of probability to predict the class of unknown data set.

  GENERAL APPLICATIONS
  - It can be applied to wide range of classification problems, such as text classification, word disambiguation and confidence measures for speech recognition etc.

  STRENGTHS
  - It is one of the fastest classification techniques.
  - It often provides good results at low cost in terms of model complexity.

  WEAKNESSES
  - One of the disadvantages of Naive Bayes is that it has strong feature independence assumption. But in reality, usually there's some relation between features, hence it's "Naive" assuming so.

  REASONS FOR CHOOSING
  - This technique has a advantage of being one of the fastest classification techniques and also it provides less complex models.
  - Given our small dataset of only a few hundred rows, it would be better to train with a high bias classifier.
  - The model also has the advantage of being interpretable to a non-technical audience like the school board.

  Model 2- Support Vector Machine:

  GENERAL APPLICATIONS
  - SVM is a powerful classification technique and is used in a variety of applications, e.g. text and hypertext categorization, classification of images, hand-written characters recognition and many more.

  STRENGTHS
  - Training is relatively easy with this (unlike neural networks it has no local optima).
  - It scales relatively well to high dimensional data.
  - Tradeoff between classifier complexity and error can be controlled explicitly.
  - Non-traditional data like strings and trees can be used as input to SVM, instead of feature vectors.

  WEAKNESSES
  - From a practical point of view perhaps the most serious problem with SVMs is the high algorithmic complexity and extensive memory requirements of the required

quadratic programming in large-scale tasks.
- It needs a “good” kernel function.

REASONS FOR CHOOSING
- This is very powerful, well known and widely used classification technique.
- It can learn complex models even with relatively decent dataset sizes.

Model 3- Logistic Regression:

GENERAL APPLICATIONS
- Logistic regression is used widely in many settings, including the medical and social sciences. For example - to predict whether a patient has a given disease like diabetes, coronary heart disease etc.
- It can be used in engineering to predict the probability of failure of a given process or a system.
- It's also widely used in marketing applications such as prediction of a customer's propensity to purchase a product.

STRENGTHS
- It is a well known technique, is quite easy to implement and fairly straight forward to understand.

WEAKNESSES
- Requires pre-processing in case of large number of predictors.

REASONS FOR CHOOSING
- It is a well known technique and easy to implement.
- It is reliable with the given small data sets of few hundreds.
- It can output probabilities of the outcome, i.e. be a soft-classifier - thus making the interpretation very easy and natural to understand to the student board.

Model 4- Decision Tree: In Decision Tree technique the model maps observations about an item to conclusions about the item's target value.

GENERAL APPLICATIONS
- The decision tree method is a powerful statistical tool for classification, prediction, interpretation, and data manipulation.
- It has several potential applications in medical research, in biomedical Engineering for identifying features to be used in implantable devices and in manufacturing and production for semiconductor manufacturing, for increasing productivity and for quality control etc.

STRENGTHS
- It is simple to understand and interpret.
- It requires little data preparation.
- It can handle both numerical and categorical data.
- It is robust, performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

- It performs well with large datasets.

WEAKNESSES

- Instability - even a small change in input data can at times, cause large changes in the tree.
- Complexity - decision tree learners can create over-complex trees that do not generalise well from the training data.
- Unwieldy - large trees are not intelligible, and pose presentation difficulties.
- Costs - decision tree analysis is an expensive option.

REASONS FOR CHOOSING

- It is simple to understand and interpret.
- It has an advantage of being flexible with both numerical and categorical data.
- It is robust.

- Just by understanding the problem statement we can conclude that the required output is binary. This narrows the selection of modeling techniques. Based on the given student intervention data, which has both categorical & continuous predictors, and considering the performance of the modeling techniques in terms of CPU utilization, the four modeling techniques chosen(Naive Bayes, SVM, Logistic Regression, Decision Tree) produce the desired outcome.

In [24]:
```
%%html
<style>
table {float:left}
</style>
```

Naive Bayes

Dataset	Training Time	Prediction Time	Training F1 Score	Test F1 Score
Training 300	0.002	0.001	0.80	0.75
Training 200	0.002	0.000	0.84	0.78
Training 100	0.002	0.001	0.55	0.46

SVM

Dataset	Training Time	Prediction Time	Training F1 Score	Test F1 Score
Training 300	0.011	0.008	0.86	0.75
Training 200	0.006	0.003	0.88	0.76
Training 100	0.002	0.002	0.89	0.78

Logistic Regression

Dataset	Training Time	Prediction Time	Training F1 Score	Test F1 Score
Training 300	0.12	0.040	0.83	0.79
Training 200	0.004	0.000	0.88	0.78
Training 100	0.003	0.000	0.90	0.78

Decision Trees

Dataset	Training Time	Prediction Time	Training F1 Score	Test F1 Score
Training 300	0.003	0.000	1.0	0.72
Training 200	0.003	0.001	1.0	0.73
Training 100	0.002	0.000	1.0	0.72

In [8]:
```
# Train a model
import time

def train_classifier(clf, X_train, y_train):
    print "Training {}...".format(clf.__class__.__name__)
    start = time.time()
    clf.fit(X_train, y_train)
    end = time.time()
    print "Done!\nTraining time (secs): {:.3f}".format(end - start)

# TODO: Choose a model, import it and instantiate an object
#Model using Naive Bayes
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
# Fit model to training data
train_classifier(clf, X_train, y_train)  # note: using entire training s
et here
#print clf  # you can inspect the learned model by printing it
```

```
Training GaussianNB...
Done!
Training time (secs): 0.002
```

In [9]:
```
# Predict on training set and compute F1 score
from sklearn.metrics import f1_score

def predict_labels(clf, features, target):
    print "Predicting labels using {}...".format(clf.__class__.__name__)
    start = time.time()
    y_pred = clf.predict(features)
    end = time.time()
    print "Done!\nPrediction time (secs): {:.3f}".format(end - start)
    return f1_score(target.values, y_pred, pos_label='yes')

train_f1_score = predict_labels(clf, X_train, y_train)
print "F1 score for training set: {}".format(train_f1_score)
```

```
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.002
F1 score for training set: 0.808823529412
```

In [10]:
```
# Predict on test data
print "F1 score for test set: {}".format(predict_labels(clf, X_test, y_t
est))
```

```
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.75
```

In [11]:
```
# Train and predict using different training set sizes
def train_predict(clf, X_train, y_train, X_test, y_test):
    print "------------------------------------------"
    print "Training set size: {}".format(len(X_train))
    train_classifier(clf, X_train, y_train)
    print "F1 score for training set: {}".format(predict_labels(clf, X_t
rain, y_train))
    print "F1 score for test set: {}".format(predict_labels(clf, X_test,
y_test))

# TODO: Run the helper function above for desired subsets of training da
ta
# Note: Keep the test set constant

#X_train1=X_train.sample(n=.75*num_train)
# Train and predict using training set size=200
train_predict(clf, X_train_200, y_train_200, X_test, y_test)
```

```
------------------------------------------
Training set size: 200
Training GaussianNB...
Done!
Training time (secs): 0.002
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.846666666667
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.788321167883
```

In [12]:
```
# Train and predict using training set size=100
train_predict(clf, X_train_100, y_train_100, X_test, y_test)
```

```
------------------------------------------
Training set size: 100
Training GaussianNB...
Done!
Training time (secs): 0.002
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.559139784946
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.46511627907
```

In [38]:
```
# TODO: Train and predict using two other models
```

In [13]:
```
# Model using Support Vector Machine
from sklearn.svm import SVC
clf2 = SVC()
# Fit model to training data
train_classifier(clf2, X_train, y_train)
#Predict on training set and compute F1 score
train_f1_score = predict_labels(clf2, X_train, y_train)
print "F1 score for training set: {}".format(train_f1_score)
# Predict on test data
print "F1 score for test set: {}".format(predict_labels(clf2, X_test, y_
test))
```

```
Training SVC...
Done!
Training time (secs): 0.011
Predicting labels using SVC...
Done!
Prediction time (secs): 0.008
F1 score for training set: 0.869198312236
Predicting labels using SVC...
Done!
Prediction time (secs): 0.002
F1 score for test set: 0.758620689655
```

In [14]:
```
# Train and predict using training set size=200
train_predict(clf2,X_train_200,y_train_200,X_test,y_test)
```

```
------------------------------------------
Training set size: 200
Training SVC...
Done!
Training time (secs): 0.006
Predicting labels using SVC...
Done!
Prediction time (secs): 0.003
F1 score for training set: 0.882352941176
Predicting labels using SVC...
Done!
Prediction time (secs): 0.002
F1 score for test set: 0.766233766234
```

In [15]:
```
# Train and predict using training set size=100
train_predict(clf2,X_train_100,y_train_100,X_test,y_test)
```

```
------------------------------------------
Training set size: 100
Training SVC...
Done!
Training time (secs): 0.002
Predicting labels using SVC...
Done!
Prediction time (secs): 0.002
F1 score for training set: 0.895104895105
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.785714285714
```

In [16]:
```
# Model using Logistic regression
from sklearn import linear_model
clf3 = linear_model.LogisticRegression()
# Fit model to training data
train_classifier(clf3, X_train, y_train)
#Predict on training set and compute F1 score
train_f1_score = predict_labels(clf3, X_train, y_train)
print "F1 score for training set: {}".format(train_f1_score)
# Predict on test data
print "F1 score for test set: {}".format(predict_labels(clf3, X_test, y_
test))
```

```
Training LogisticRegression...
Done!
Training time (secs): 0.120
Predicting labels using LogisticRegression...
Done!
Prediction time (secs): 0.040
F1 score for training set: 0.838137472284
Predicting labels using LogisticRegression...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.791044776119
```

In [17]:
```
# Train and predict using training set size=200
train_predict(clf3,X_train_200,y_train_200,X_test,y_test)
```

```
------------------------------------------
Training set size: 200
Training LogisticRegression...
Done!
Training time (secs): 0.004
Predicting labels using LogisticRegression...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.883435582822
Predicting labels using LogisticRegression...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.788321167883
```

In [18]:
```
# Train and predict using training set size=100
train_predict(clf3,X_train_100,y_train_100,X_test,y_test)
```

```
------------------------------------------
Training set size: 100
Training LogisticRegression...
Done!
Training time (secs): 0.003
Predicting labels using LogisticRegression...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.906474820144
Predicting labels using LogisticRegression...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.787401574803
```

In [19]:
```
# Model using Decision tree
from sklearn import tree
clf4 = tree.DecisionTreeClassifier()
# Fit model to training data
train_classifier(clf4, X_train, y_train)
#Predict on training set and compute F1 score
train_f1_score = predict_labels(clf4, X_train, y_train)
print "F1 score for training set: {}".format(train_f1_score)
# Predict on test data
print "F1 score for test set: {}".format(predict_labels(clf4, X_test, y_
test))
```

```
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.003
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for training set: 1.0
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.72131147541
```

In [20]:
```
# Train and predict using training set size=200
train_predict(clf4,X_train_200,y_train_200,X_test,y_test)
```

```
------------------------------------------
Training set size: 200
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.003
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for training set: 1.0
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.738461538462
```

In [21]:
```
# Train and predict using training set size=100
train_predict(clf4,X_train_100,y_train_100,X_test,y_test)
```

```
------------------------------------------
Training set size: 100
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.002
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for training set: 1.0
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.725663716814
```

5. Choosing the Best Model

- Based on the experiments you performed earlier, in 1-2 paragraphs explain to the board of supervisors what single model you chose as the best model. Which model is generally the most appropriate based on the available data, limited resources, cost, and performance?
- In 1-2 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 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 F_1 score?

Answer:

- Based on the experiments, all four models (Gaussian Naive Bayes, SVM, Logistic Regression and Decision Trees) take less than a second to perform classification and prediction. So timing wise, all models perform reasonably well and thus eliminate the need to compare models based on the computing time. Hence the error metric F1 score becomes the deciding factor to choose the best performing model.

  When we look at the F1 scores on training and the test set for various models, the **Logistic Regression model is the best** performing model of all.

  F1 score is a performance matric explained by a weighted average of precision and recall (F1 = 2 *(precision* recall) / (precision + recall)). It varies between 0 and 1. A higher F1 score indicates a better model. If we look at the F1 scores of the Logistic Regression models trained on various dataset sizes, the F1 score on training decreases as dataset size increases; but F1 score increases on the test set for the same models. This is inline with the learning curves theory, wherein the training error keeps on increasing as dataset size increases, and the corresponding error on test set decreases. This behavior is not seen with any of the models other than the Logistic Regression, thus Logistic Regression is considered the best in the given context.

  The F1 score on training dataset (300) for Logistic Regression and SVM models is almost same, but when we fit **SVM** on different training dataset sizes, F1 score increases for smaller training sizes (conversely, F1 decreases on larger training sets) and so does the F1 on corresponding test dataset. This shows that the model is perfoming better with fewer data (as opposed to working better with more data), which is possibly not correct. Hence this model is not chosen.

  On the **Decision Tree**, the F1 score is 1 for all the training dataset sizes and it is less than or equal to 0.75 for the test set. This shows that the model is overfitting on training data (classifying all the labels correctly), but fails to generalise on test. So this model is not chosen.

  For **Gaussian Naive Bayes** the F1 score is not consistent throughout the experiment between training sets and the test dataset. The incosistency is more so highlighted with the training set size of 100. This suggests that this algorithm probably needs more data, which is one of the disdvantages of Naive Bayes, to classify and generalise correctly.

- Based on the results from various techniques, I would propose the Logistic Regression model as the best model for student intervention system. Because if we look at the results, though the time taken by all four models is almost the same (less than a second), F1 scores are different - thus F1 becomes the deciding factor. The F1 scores yielded by Logistic Regression seem to be the most consistent and reliable. Other models such as Naive Bayes clearly do not work best with fewer data points. Looking at the pattern of decreased training and test F1 scores with increase in training data indicate that SVM is not performing better with more data. Finally, the Decision Tree F1 scores of training and test show that the model is inconsistent.

To explain in simple terms on how the Logistic Regression model works, it looks at each feature of the student e.g. age, gender, address, traveltime, studytime etc and learns to assign a weight to that feature which assesses the feature's importance in passing or failing the exam. Then it plugs in values into the model's equation by multiplying weights with corresponding feature values and summing them up, after which the results are input to the "Logistic" or "Logit" function. Finally the logit yeilds the probability that the student passes the exam and based on some threshold probability value (0.5 default), it classifies into Passed or Failed categories.

Thus the Logistic Regression model seems to perfom the best for the given student dataset. In simple terms, the Logistic Regression categorises the students into two groups: one group with students that have passed exams and another that have failed. Essentially the Logistic Regression algorithm learns how all the features (independent variables) contribute to student's performance (pass or fail the exam). The model built using the Logistic Regression on historical student data would predict the possibility of students' performance on future exams. So using this model we can predict how the student might perform in future exams and thus can decide whether or not a particular student needs intervention to reach the goal of 95% graduation rate.

Logistic Regression

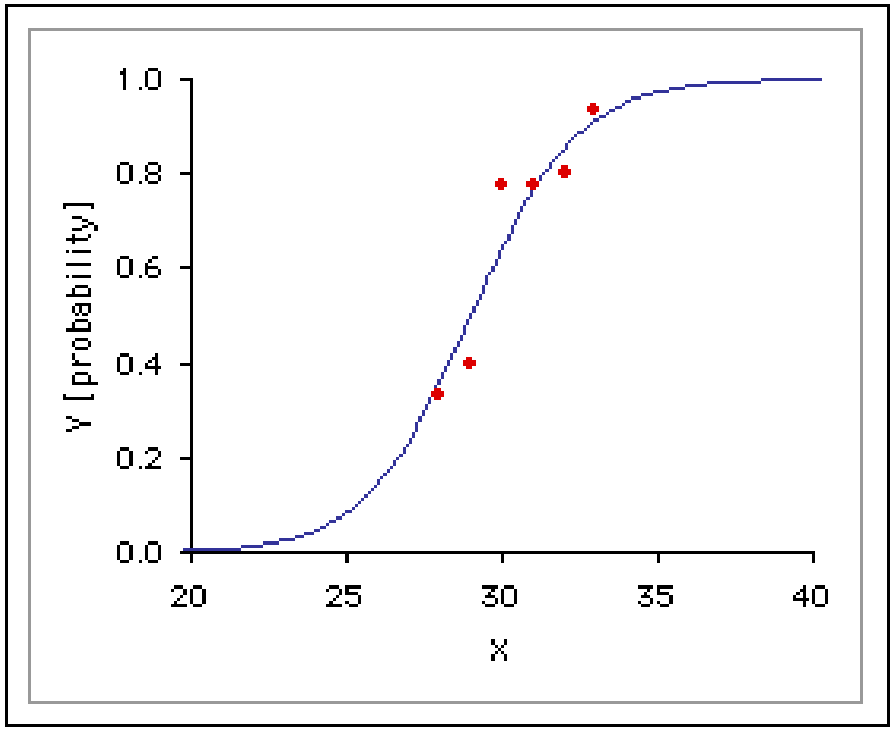

Fig: Logistic Regression Example Graph (at threshold 0.5)

In [22]:
```
# TODO: Fine-tune your model and report the best F1 score
```

In [31]:
```
from sklearn.metrics import make_scorer
from sklearn.grid_search import GridSearchCV
params={'C' :[.005,.05,.5,1.,10.,],
        'fit_intercept' : [False],
        'class_weight': ['balanced'],
        'random_state' : [45],
        'penalty': ['l1','l2'],
        'n_jobs': [-1]
       }
scoring_function = make_scorer(f1_score, pos_label="yes", greater_is_bet
ter=True)
Logistic_grid = GridSearchCV(clf3,param_grid=params, scoring=scoring_fun
ction, n_jobs=-1,cv=3)
train_classifier(Logistic_grid, X_train, y_train)
#Predict on training set and compute F1 score
train_f1_score = predict_labels(Logistic_grid, X_train, y_train)
print "Tuned F1 score for training set: {}".format(train_f1_score)
# Predict on test data
print "Tuned F1 score for test set: {}".format(predict_labels(Logistic_g
rid, X_test, y_test))
```

```
Training GridSearchCV...
Done!
Training time (secs): 7.884
Predicting labels using GridSearchCV...
Done!
Prediction time (secs): 0.001
Tuned F1 score for training set: 0.801033591731
Predicting labels using GridSearchCV...
Done!
Prediction time (secs): 0.000
Tuned F1 score for test set: 0.758064516129
```

- After fine tuning with gridsearchcv with a default of 3 folds, the final F1 score of the trained model is 0.8 and the test is 0.75