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?

To identify whether a student requires an early intervention is a classification. The idea is to use a large sum of data, and generalize whether the student will pass or or the high school exams. We require a generalization for our students to indicate which student needs assistance or not. This is clearly a classification case.

Regression is only used when the result can be linear, but the student intervention necessity only requires two types of output.

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 fea
```

```
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]:
```
n_students = len(student_data)
n_features = student_data.shape[1]
n_passed =  student_data[student_data['passed'] == 'yes'].shape[0]
n_failed = student_data[student_data['passed'] == 'no'].shape[0]
grad_rate = float(n_passed *1.0 / n_students*1.0)*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 "Number of features: {}".format(n_features)
print "Graduation rate of the class: {:.2f}%".format(grad_rate)
```

```
Total number of students: 395
Number of students who passed: 265
Number of students who failed: 130
Number of features: 31
Graduation rate of the class: 67.09%
```

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
target_col = student_data.columns[-1]  # last column is the target/labe
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', 'F
edu', 'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytim
e', 'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nurser
y', 'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout',
'Dalc', 'Walc', 'health', 'absences']
Target column: passed

Feature values:-
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      
Fjob  \
0     GP   F   18       U     GT3       A     4     4  at_home   tea
cher   
1     GP   F   17       U     GT3       T     1     1  at_home     o
ther   
2     GP   F   15       U     LE3       T     1     1  at_home     o
ther   
3     GP   F   15       U     GT3       T     4     2   health  serv
ices   
4     GP   F   16       U     GT3       T     3     3    other     o
ther   

    ...    higher internet  romantic  famrel  freetime goout Dalc Wa
lc 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)](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 e

    # Check each column
    for col, col_data in X.iteritems():
        # If data type is non-numeric, try to replace all yes/no values
        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

        # 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. 'sc

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

    return outX

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

```
Processed feature columns (48):-
['school_GP', 'school_MS', 'sex_F', 'sex_M', 'age', 'address_R', 'ad
dress_U', 'famsize_GT3', 'famsize_LE3', 'Pstatus_A', 'Pstatus_T', 'M
edu', 'Fedu', 'Mjob_at_home', 'Mjob_health', 'Mjob_other', 'Mjob_ser
vices', 'Mjob_teacher', 'Fjob_at_home', 'Fjob_health', 'Fjob_other',
'Fjob_services', 'Fjob_teacher', 'reason_course', 'reason_home', 're
ason_other', 'reason_reputation', 'guardian_father', 'guardian_mothe
r', 'guardian_other', 'traveltime', 'studytime', 'failures', 'school
sup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'interne
t', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Walc', 'heal
th', '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

from sklearn import cross_validation
# TODO: Then, select features (X) and corresponding labels (y) for the
# Note: Shuffle the data or randomly select samples to avoid any bias c
X_train, X_test, y_train, y_test = cross_validation.train_test_split(X_

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 c
```

```
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 is the theoretical O(n) time & space complexity in terms of input size?
- 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 F_1 score. Repeat this process with different training set sizes (100, 200, 300), keeping test set constant.

Produce a table showing training time, prediction time, F_1 score on training set and F_1 score on test set, for each training set size.

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

In [7]:

```
# 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 objectf
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()

# Fit model to training data
train_classifier(clf, X_train, y_train)  # note: using entire training
#print clf  # you can inspect the learned model by printing it
```

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

In [8]:

```
# 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): 0.001
F1 score for training set: 0.808823529412
```

In [9]:

```
# Predict on test data
print "F1 score for test set: {}".format(predict_labels(clf, X_test, y_
```

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

In [10]:
```
# 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_
    print "F1 score for test set: {}".format(predict_labels(clf, X_test

# TODO: Run the helper function above for desired subsets of training d
# Note: Keep the test set constant
train_predict(clf, X_train, y_train, X_test, y_test)
```

```
------------------------------------------
Training set size: 300
Training GaussianNB...
Done!
Training time (secs): 0.002
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.808823529412
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.75
```

In [11]:
```
from sklearn.svm import SVC
svm_clf = SVC()
train_predict(svm_clf, X_train, y_train, X_test, y_test)
```

```
------------------------------------------
Training set size: 300
Training SVC...
Done!
Training time (secs): 0.011
Predicting labels using SVC...
Done!
Prediction time (secs): 0.005
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 [12]:
```
from sklearn import tree
tree_clf = tree.DecisionTreeClassifier()
train_predict(tree_clf, X_train, y_train, X_test, y_test)
```

```
------------------------------------------
Training set size: 300
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.004
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.001
F1 score for training set: 1.0
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.704918032787
```

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?

Among all of the used models, the best model that is used is suppor vector machine. It has the highest number for the F1 score(0.758) for the testing set. The F1 score averages the mean between precision and recall. Precision is amount of clustered data, and recall is the ratio of the correct data. The highest F1 score indicates it is the most efficient model to use to provide the correct result for classification on the likelyhood the student will pass the highschool.

With the given availalble data, limited resource, cost, and performance, the best model that should be used is Gaussian Naive Bayes model. Although Naive Bayes Model and Decision tree have the same prediction time for predicting label, the trainign time for Naive Bayes is 0.002 seconds, which is 50% faster than the decisiontree time (0.004).

- How does Support Vector work?

Support Vector Machine Draws out a hyperplane that classified between the students that will pass and fail. What Support Vector Machine does specificially is it picks out the most optimal line between the two ends by having gray scale taht surface around it to make it write.

To prevent from having a hyperplane that is close to the data, and can be noise sensitive, it finds the largest minimum distaince to the training set to maximize the marginal length to the data set.

It makes prediciton by picking up the testing set, and the classification line gets adjusted whenever new data set has been added in.

- Tuning the Model for the most optimal SVM. Grab the full training set.

In [45]:

```
# TODO: Fine-tune your model and report the best F1 score
from sklearn.grid_search import GridSearchCV

def fit_model(X, y):
    """ Tunes a SVM model using GridSearchCV on the input data X
        and target labels y and returns this optimal model. """

    # Create a decision tree regressor object
    regressor = SVC()
    # Set up the parameters we wish to tune
    parameters = {'C':(1.001,1.0011,1.0012,1.0013,1.0014,1.0015,1.0016,

    # Make the GridSearchCV object
    reg = GridSearchCV(estimator=regressor,
                       param_grid=parameters,
                       )

    # Fit the learner to the data to obtain the optimal model with tune
    reg.fit(X, y)

    # Return the optimal model
    return reg.best_estimator_
try:
    reg = fit_model(X_train, y_train)
    print "Fit Completed!"
except:
    print "error happened."
```

```
Fit Completed!
```

In [46]:

```
print "Best preffered Value for C is ", reg.get_params()['C']
```

```
Best preffered Value for C is  1.001
```

In [48]:
```
# Running the test with a better C value.

svm_clf = SVC(C=1.001)
train_predict(svm_clf, X_train, y_train, X_test, y_test)
```

```
------------------------------------------
Training set size: 300
Training SVC...
Done!
Training time (secs): 0.007
Predicting labels using SVC...
Done!
Prediction time (secs): 0.005
F1 score for training set: 0.871035940803
Predicting labels using SVC...
Done!
Prediction time (secs): 0.002
F1 score for test set: 0.758620689655
```

Looking at the GridSearchSV result, it seems like when C= 1.001 is the best outcome for C to be the best estimater.