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?

Ans: Identifying students for early intervention is a classification problem. We are trying to find if can predicts student result and if student is going to fail then intervention can be happened ahead of time to assist them being failed. Here are few reason to clarify that.

1. Identifying students for early intervention needs descrete identification, yes or no. It don't require any other information to support that. For descrete output, classification is a best way to solve the problem.
2. Regression has continuous output which is not the case in this problem.
3. Regression has some sort of order in output but in this case there is not order, what we are really looking is yes or no.

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 feature 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 expression/function call
n_students = student_data.shape[0]
n_features = student_data.shape[1] - 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) / 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 "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: 30
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 are 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', '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

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

    ...    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)](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 empty

    # 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 int

        # 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. 'school' => 'school_GP', 'school_
MS'

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

    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', 'address_U', 'famsize_GT3', 'famsiz
e_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_teac
her', 'reason_course', 'reason_home', 'reason_other', 'reason_reputation', 'guardian_father', 'guard
ian_mother', 'guardian_other', 'traveltime', 'studytime', 'failures', 'schoolsup', 'famsup', 'paid',
'activities', 'nursery', 'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'W
alc', '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

# TODO: Then, select features (X) and corresponding labels (y) for the training and test sets
# Note: Shuffle the data or randomly select samples to avoid any bias due to ordering in the dataset
from sklearn.cross_validation import train_test_split

# X_train = ?
# y_train = ?
# X_test = ?
# y_test = ?
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_size=
                                        (float(num_test)/float(num_all)), train_size=(float(num_trai
n)/float(num_all)),
                                            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 data
```

```
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 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 object
from sklearn.naive_bayes import GaussianNB

clf = GaussianNB()

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

```
Training GaussianNB...
Done!
Training time (secs): 0.001
```

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): 0.002
F1 score for training set: 0.80378250591
```

In [9]:

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

```
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.763358778626
```

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_train, 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 data
# Note: Keep the test set constant
train_predict(clf, X_train[:100], y_train[:100], X_test, y_test)
train_predict(clf, X_train[:200], y_train[:200], X_test, y_test)
train_predict(clf, X_train[:250], y_train[:250], X_test, y_test)
train_predict(clf, X_train, y_train, X_test, y_test)
```

```
----------------------------------------
Training set size: 100
Training GaussianNB...
Done!
Training time (secs): 0.002
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.846715328467
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.802919708029
----------------------------------------
Training set size: 200
Training GaussianNB...
Done!
Training time (secs): 0.001
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.840579710145
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.724409448819
----------------------------------------
Training set size: 250
Training GaussianNB...
Done!
Training time (secs): 0.001
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.829545454545
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.77519379845
----------------------------------------
```

```
Training set size: 300
Training GaussianNB...
Done!
Training time (secs): 0.001
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.80378250591
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.763358778626
```

In [11]:
```
# TODO: Train and predict using two other models
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import BaggingClassifier
clf = BaggingClassifier(KNeighborsClassifier(n_neighbors=3),max_samples=0.5, max_features=0.5)
train_predict(clf, X_train[:100], y_train[:100], X_test, y_test)
train_predict(clf, X_train[:200], y_train[:200], X_test, y_test)
train_predict(clf, X_train[:250], y_train[:250], X_test, y_test)
train_predict(clf, X_train, y_train, X_test, y_test)
```

```
------------------------------------------
Training set size: 100
Training BaggingClassifier...
Done!
Training time (secs): 0.018
Predicting labels using BaggingClassifier...
Done!
Prediction time (secs): 0.005
F1 score for training set: 0.816901408451
Predicting labels using BaggingClassifier...
Done!
Prediction time (secs): 0.005
F1 score for test set: 0.741258741259
------------------------------------------
Training set size: 200
Training BaggingClassifier...
Done!
Training time (secs): 0.015
Predicting labels using BaggingClassifier...
Done!
Prediction time (secs): 0.011
F1 score for training set: 0.865573770492
Predicting labels using BaggingClassifier...
Done!
Prediction time (secs): 0.007
F1 score for test set: 0.760563380282
------------------------------------------
Training set size: 250
Training BaggingClassifier...
Done!
Training time (secs): 0.015
Predicting labels using BaggingClassifier...
Done!
Prediction time (secs): 0.019
F1 score for training set: 0.863636363636
Predicting labels using BaggingClassifier...
Done!
Prediction time (secs): 0.009
F1 score for test set: 0.77027027027
------------------------------------------
```

```
Training set size: 300
Training BaggingClassifier...
Done!
Training time (secs): 0.048
Predicting labels using BaggingClassifier...
Done!
Prediction time (secs): 0.022
F1 score for training set: 0.867841409692
Predicting labels using BaggingClassifier...
Done!
Prediction time (secs): 0.009
F1 score for test set: 0.769230769231
```

In [12]:
```
from sklearn.svm import SVC
clf = SVC()

train_predict(clf, X_train[:100], y_train[:100], X_test, y_test)
train_predict(clf, X_train[:200], y_train[:200], X_test, y_test)
train_predict(clf, X_train[:250], y_train[:250], X_test, y_test)
train_predict(clf, X_train, y_train, X_test, y_test)
```

```
------------------------------------------
Training set size: 100
Training SVC...
Done!
Training time (secs): 0.002
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.877697841727
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.774647887324
------------------------------------------
Training set size: 200
Training SVC...
Done!
Training time (secs): 0.004
Predicting labels using SVC...
Done!
Prediction time (secs): 0.003
F1 score for training set: 0.867924528302
Predicting labels using SVC...
Done!
Prediction time (secs): 0.002
F1 score for test set: 0.781456953642
------------------------------------------
Training set size: 250
Training SVC...
Done!
Training time (secs): 0.006
Predicting labels using SVC...
Done!
Prediction time (secs): 0.004
F1 score for training set: 0.871536523929
Predicting labels using SVC...
Done!
Prediction time (secs): 0.002
F1 score for test set: 0.773333333333
------------------------------------------
```

```
Training set size: 300
Training SVC...
Done!
Training time (secs): 0.009
Predicting labels using SVC...
Done!
Prediction time (secs): 0.008
F1 score for training set: 0.876068376068
Predicting labels using SVC...
Done!
Prediction time (secs): 0.003
F1 score for test set: 0.783783783784
```

In [13]:
```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()

train_predict(clf, X_train[:100], y_train[:100], X_test, y_test)
train_predict(clf, X_train[:200], y_train[:200], X_test, y_test)
train_predict(clf, X_train[:250], y_train[:250], X_test, y_test)
train_predict(clf, X_train, y_train, X_test, y_test)
```

```
------------------------------------------
Training set size: 100
Training RandomForestClassifier...
Done!
Training time (secs): 0.031
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.992125984252
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.003
F1 score for test set: 0.759689922481
------------------------------------------
Training set size: 200
Training RandomForestClassifier...
Done!
Training time (secs): 0.043
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.985714285714
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.785714285714
------------------------------------------
Training set size: 250
Training RandomForestClassifier...
Done!
Training time (secs): 0.021
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.99711815562
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.766917293233
------------------------------------------
```

```
Training set size: 300
Training RandomForestClassifier...
Done!
Training time (secs): 0.019
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.002
F1 score for training set: 0.99512195122
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.713178294574
```

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?

In [18]:
```
# TODO: Fine-tune your model and report the best F1 score
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.svm import SVC

parameters = {'kernel':('linear','rbf', 'poly','sigmoid'), 'C':[1, 50], 'degree':[3,6]}
custom_f1_scorer = make_scorer(f1_score, pos_label='yes')

clf = SVC()
regressor = GridSearchCV(clf, parameters, scoring=custom_f1_scorer)
regressor.fit(X_train, y_train)
reg = regressor.best_estimator_
print reg
train_f1_score = predict_labels(reg, X_train, y_train)
print "F1 score for training set: {}".format(train_f1_score)

print "F1 score for test set: {}".format(predict_labels(reg, X_test, y_test))
print "Best parameters for the final tuned SVM model is {}".format(regressor.best_params_)
```

```
SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
Predicting labels using SVC...
Done!
Prediction time (secs): 0.006
F1 score for training set: 0.876068376068
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.783783783784
Best parameters for the final tuned SVM model is {'kernel': 'rbf', 'C': 1, 'degree': 3}
```

In [ ]: