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?

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
from sklearn.cross_validation import train_test_split
```

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 colum
ns
```

```
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 = np.shape(student_data)[0]
n_features = np.shape(student_data)[1]
n_passed = np.sum(student_data.passed=="yes")
n_failed = np.sum(student_data.passed=="no")
grad_rate = float(n_passed) / n_students
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: 0.67%
```

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 featur
es
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', 'fam
rel', '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) 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_al
l.columns))
```

```
Processed feature columns (48):-
['school_GP', 'school_MS', 'sex_F', 'sex_M', 'age', 'address_R', 'address_U', 'fa
msize_GT3', 'famsize_LE3', 'Pstatus_A', 'Pstatus_T', 'Medu', 'Fedu', 'Mjob_at_hom
e', 'Mjob_health', 'Mjob_other', 'Mjob_services', 'Mjob_teacher', 'Fjob_at_home',
'Fjob_health', 'Fjob_other', 'Fjob_services', 'Fjob_teacher', 'reason_course', 'r
eason_home', 'reason_other', 'reason_reputation', 'guardian_father', 'guardian_mo
ther', 'guardian_other', 'traveltime', 'studytime', 'failures', 'schoolsup', 'fam
sup', 'paid', '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

# TODO: Then, select features (X) and corresponding labels (y) for the training a
nd test sets
# Note: Shuffle the data or randomly select samples to avoid any bias due to orde
ring in the dataset
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_size = num
_test, random_state = 0)
#X_train = ?
#y_train = ?
#X_test = ?
#y_test = ?
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()
    Ttime = end - start
    #print "Done!\nTraining time (secs): {:.3f}".format(end - start)
    return Ttime

# TODO: Choose a model, import it and instantiate an object
from sklearn.ensemble import GradientBoostingClassifier

clf = GradientBoostingClassifier(n_estimators=200, learning_rate=1.0,
                                 max_depth=3, random_state=0).fit(X_train, y_trai
n)

# 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
```

Out[7]: 0.1679999828338623

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__)
    y_pred = clf.predict(features)
    #print "Done!\nPrediction time (secs): {:.3f}".format(Ttime)
    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)
```

```
F1 score for training set: 1.0
```

In [9]:
```
# Predict on test data
print "F1 score for test set: {}".format(predict_labels(clf, X_test, y_test))
```

```
F1 score for test set: 0.785185185185
```

In [10]:
```
# Train and predict using different training set sizes
def train_predict(clf, X_train, y_train, X_test, y_test):
    #print "------------------------------------------"
    #trainsize = len(X_train)
    Ttime = train_classifier(clf, X_train, y_train)

    start = time.time()
    trainF1 = predict_labels(clf, X_train, y_train)
    testF1 = predict_labels(clf, X_test, y_test)
    end = time.time()
    Ptime = end - start

    #print "Training set size: {}".format(trainsize)
    #print "F1 score for training set: {}".format(trainF1)
    #print "F1 score for test set: {}".format(testF1)
    return Ttime, Ptime, trainF1, testF1

# TODO: Run the helper function above for desired subsets of training data
# Note: Keep the test set constant
traintime1, testtime1, trainF11, testF11 = train_predict(clf, X_train[0:100], y_t
rain[0:100], X_test, y_test)
traintime2, testtime2, trainF12, testF12 = train_predict(clf, X_train[0:200], y_t
rain[0:200], X_test, y_test)
traintime3, testtime3, trainF13, testF13 = train_predict(clf, X_train[0:300], y_t
rain[0:300], X_test, y_test)


rnames = ["Training time (sec)      ","Prediction time (secs)   ","F1 score for t
raining set","F1 score for test set    "]
cnames = ["100","200","300"]


data =    ([["{:.3f}".format(traintime1), "{:.3f}".format(traintime2), "{:.3f}".f
ormat(traintime3)],
            ["{:.3f}".format(testtime1),  "{:.3f}".format(testtime2),  "{:.3f}".f
ormat(testtime3)],
            ["{:.3f}".format(trainF11),   "{:.3f}".format(trainF12),   "{:.3f}".f
ormat(trainF13)],
            ["{:.3f}".format(testF11),    "{:.3f}".format(testF12),    "{:.3f}".f
ormat(testF13)]])

print ("")
print "Summary Table using {}:".format(clf.__class__.__name__)
col_format ="{:>18}"+"{:>21}"+"{:>15}"+"{:>15}"
row_format ="{:>15}" * (4)
header = "{:>44}"+"Training Set Size"
print header.format("")
print col_format.format("", *cnames)
for x, row in zip(rnames, data):
    print row_format.format(x, *row)
```

Summary Table using GradientBoostingClassifier:

	Training Set Size		
	100	200	300
Training time (sec)	0.091	0.122	0.168
Prediction time (secs)	0.002	0.002	0.003
F1 score for training set	1.000	1.000	1.000
F1 score for test set	0.693	0.752	0.785

In [11]:

```
#Train model 2-------------------------------------------------------------
---------
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=2000, random_state=0)

traintime1, testtime1, trainF11, testF11 = train_predict(clf,
                        X_train[0:100], y_train[0:100], X_test, y_test)
traintime2, testtime2, trainF12, testF12 = train_predict(clf,
                        X_train[0:200], y_train[0:200], X_test, y_test)
traintime3, testtime3, trainF13, testF13 = train_predict(clf,
                        X_train[0:300], y_train[0:300], X_test, y_test)

rnames = ["Training time (sec)      ","Prediction time (secs)   ","F1 score for t
raining set","F1 score for test set    "]
cnames = ["100","200","300"]

data =    ([["{:.3f}".format(traintime1), "{:.3f}".format(traintime2), "{:.3f}".f
ormat(traintime3)],
           ["{:.3f}".format(testtime1),  "{:.3f}".format(testtime2),  "{:.3f}".f
ormat(testtime3)],
           ["{:.3f}".format(trainF11),   "{:.3f}".format(trainF12),   "{:.3f}".f
ormat(trainF13)],
           ["{:.3f}".format(testF11),    "{:.3f}".format(testF12),    "{:.3f}".f
ormat(testF13)]])

print ("")
print "Summary Table using {}:".format(clf.__class__.__name__)
col_format ="{:>18}"+"{:>21}"+"{:>15}"+"{:>15}"
row_format ="{:>15}" * (4)
header = "{:>44}"+"Training Set Size"
print header.format("")
print col_format.format("", *cnames)
for x, row in zip(rnames, data):
    print row_format.format(x, *row)

#End Model 2---------------------------------------------------------------
---------

#Train model 3-------------------------------------------------------------
---------
from sklearn import svm

clf = svm.SVC(C=0.35, kernel='linear', max_iter=-1, random_state=0)

traintime1, testtime1, trainF11, testF11 = train_predict(clf,
                        X_train[0:100], y_train[0:100], X_test, y_test)
traintime2, testtime2, trainF12, testF12 = train_predict(clf,
                        X_train[0:200], y_train[0:200], X_test, y_test)
traintime3, testtime3, trainF13, testF13 = train_predict(clf,
                        X_train[0:300], y_train[0:300], X_test, y_test)

rnames = ["Training time (sec)      ","Prediction time (secs)   ","F1 score for t
raining set","F1 score for test set    "]
cnames = ["100","200","300"]

data =    ([["{:.3f}".format(traintime1), "{:.3f}".format(traintime2), "{:.3f}".f
ormat(traintime3)],
```

```
            ["{:.3f}".format(testtime1),  "{:.3f}".format(testtime2),  "{:.3f}".f
ormat(testtime3)],
            ["{:.3f}".format(trainF11),   "{:.3f}".format(trainF12),   "{:.3f}".f
ormat(trainF13)],
            ["{:.3f}".format(testF11),    "{:.3f}".format(testF12),    "{:.3f}".f
ormat(testF13)]])

print ("")
print "Summary Table using {}:".format(clf.__class__.__name__)
col_format ="{:>18}"+"{:>21}"+"{:>15}"+"{:>15}"
row_format ="{:>15}" * (4)
header = "{:>44}"+"Training Set Size"
print header.format("")
print col_format.format("", *cnames)
for x, row in zip(rnames, data):
    print row_format.format(x, *row)

#End Model 3-----------------------------------------------------------------
---------
```

```
Summary Table using RandomForestClassifier:
                                           Training Set Size
                                      100             200             300
Training time (sec)                 3.647           3.751           3.921
Prediction time (secs)              0.256           0.283           0.325
F1 score for training set           1.000           1.000           1.000
F1 score for test set               0.775           0.781           0.781

Summary Table using SVC:
                                           Training Set Size
                                      100             200             300
Training time (sec)                 0.005           0.007           0.017
Prediction time (secs)              0.002           0.003           0.004
F1 score for training set           0.870           0.847           0.843
F1 score for test set               0.746           0.770           0.794
```

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?

After splitting the data into a training set and a test set, three models were trained on the training set. The three models used were a gradient boosting model (GBM), random forest (RF), and a support vector machine (SVM). These model's performance was determined based on the how it performed at classifying the test set. The SVM outperformed the other two models on classifying the test set, training time, and prediction time. For these reason it is recommended to move forward with the SVM model with an F1 score 0.794.

With a two dimensional problem the SVM is a model which tries to draw a curve between the different features to separate the outcomes. This is a multidimensional problem so the SVM is going to try to make a surface, instead of a single curve, between all of those dimensions that best separates the students that graduated and those that did not. The best surface or curve is the one that maximizes the distance between the different points of that feature with the different outcomes.

This model will be fined tuned to try get the best performance out. This will be done by using a grid search to try several different input parameters.

In [12]:
```
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import make_scorer

parameters = {'C': (0.05,0.10,0.15,0.20,0.25,0.30,0.35,0.40,0.45,0.50), 'degree':
(1,2,3,4,5)}
clf = svm.SVC(random_state=0, max_iter=-1, kernel='linear', cache_size=1000)

f1_scorer = make_scorer(f1_score, pos_label="yes")
reg = GridSearchCV(clf, parameters, scoring = f1_scorer)
reg.fit(X_train, y_train)

print "Best model parameter:  " + str( reg.best_params_)
print "Best F1 Score: " + str(f1_score(reg.predict(X_test), y_test, pos_label='ye
s'))
```

```
Best model parameter:  {'C': 0.05, 'degree': 1}
Best F1 Score: 0.8
```

In [ ]: