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?

This is a classification problem as we are modeling whether or not a student will pass their final high school exam. They will either pass or fail, which is a binary outcome, not a continuous one.

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 appropria
te expression/function call
import math
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 "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', 'study
time', 'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'n
ursery', '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   t
eacher
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  se
rvices
4     GP   F   16       U     GT3       T     3     3    other
other

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

  health absences
0      3        6
1      3        4
2      3       10
3      5        2
4      5        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, initial
ly empty

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

    return outX

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

```
Processed feature columns (48):-
['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', 'F
job_other', 'Fjob_services', 'Fjob_teacher', 'reason_course', 'rea
son_home', 'reason_other', 'reason_reputation', 'guardian_father',
'guardian_mother', 'guardian_other', 'traveltime', 'studytime', 'f
ailures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery',
'higher', 'internet', 'romantic', 'famrel', '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 and test sets
# Note: Shuffle the data or randomly select samples to avoid any bi
as due to ordering in the dataset

from sklearn.cross_validation import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, t
est_size=num_test, random_state=0)

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 traini
ng 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]:
```
# Functions for training a model
import time
from sklearn.metrics import f1_score

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)


# Predict on training set and compute F1 score
def predict_labels(clf, features, target):
    print "Predicting labels using {}...".format(clf.__class__.__na
me__)
    start = time.time()
    y_pred = clf.predict(features)
    end = time.time()
    print "Done!\nPrediction time (secs): {:.3f}".format(end - star
t)
    return f1_score(target.values, y_pred, pos_label='yes')


# 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(cl
f, X_train, y_train))
    print "F1 score for test set: {}".format(predict_labels(clf, X_
test, y_test))
```

In [8]:
```
########## Model 1 - Decision Tree ##########
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(max_depth=4)

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

```
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.003
```

In [9]:
```
# Print F1 Score on Training Data
train_f1_score = predict_labels(clf, X_train, y_train)
print "F1 score for training set: {}".format(train_f1_score)
```

```
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.864745011086
```

In [10]:
```
# Predict on test data and Print F1 Score
print "F1 score for test set: {}".format(predict_labels(clf, X_tes
t, y_test))
```

```
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.773722627737
```

In [11]:
```
#Train and Predict using different training set sizes
# n_train = 100
train_predict(clf, X_train[0:100], y_train[0:100], X_test, y_test)
```

```
------------------------------------------
Training set size: 100
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.001
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.833333333333
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.727272727273
```

In [12]:
```
# n_train = 200
train_predict(clf, X_train[0:200], y_train[0:200], X_test, y_test)
```

```
------------------------------------------
Training set size: 200
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.001
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.858156028369
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.764705882353
```

In [13]:
```
########## Model 2 - SVM ##########
from sklearn.svm import SVC
clf = SVC()

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

```
Training SVC...
Done!
Training time (secs): 0.007
```

In [14]:
```
# Predict on training set and compute F1 score
train_f1_score = predict_labels(clf, X_train, y_train)
print "F1 score for training set: {}".format(train_f1_score)
```

```
Predicting labels using SVC...
Done!
Prediction time (secs): 0.005
F1 score for training set: 0.869198312236
```

In [15]:
```
# Predict on test data
print "F1 score for test set: {}".format(predict_labels(clf, X_tes
t, y_test))
```

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

In [16]:
```
#Train and Predict using different training set sizes
# n_train = 100
train_predict(clf, X_train[0:100], y_train[0:100], X_test, y_test)
```

```
------------------------------------------
Training set size: 100
Training SVC...
Done!
Training time (secs): 0.002
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.859060402685
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.783783783784
```

In [17]:
```
# n_train = 200
train_predict(clf, X_train[0:200], y_train[0:200], X_test, y_test)
```

```
------------------------------------------
Training set size: 200
Training SVC...
Done!
Training time (secs): 0.004
Predicting labels using SVC...
Done!
Prediction time (secs): 0.002
F1 score for training set: 0.869281045752
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.775510204082
```

In [18]:
```
########## Model 3 - Gaussian 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 train
ing set here
#print clf  # you can inspect the learned model by printing it
```

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

In [19]:
```
# Predict on training set and compute F1 score
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 [20]:
```
# Predict on test data
print "F1 score for test set: {}".format(predict_labels(clf, X_tes
t, y_test))
```

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

In [21]:
```
#Train and Predict using different training set sizes
# n_train = 100
train_predict(clf, X_train[0:100], y_train[0:100], X_test, y_test)
```

```
------------------------------------------
Training set size: 100
Training GaussianNB...
Done!
Training time (secs): 0.001
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.854961832061
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.748091603053
```

In [22]:
```
# n_train = 200
train_predict(clf, X_train[0:200], y_train[0:200], X_test, y_test)
```

```
------------------------------------------
Training set size: 200
Training GaussianNB...
Done!
Training time (secs): 0.001
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.832061068702
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.713178294574
```

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?

Decision Tree Model

Decision Trees can be used for both regression and classification which makes their application numerous and across many different types of data and problems. They are conceptually simple, but often pack a powerful punch in their ability to predict nearly as well as more complicated models.

The strength of decision trees are in their interpretabilitiy and ease of understanding. Data preparation for decision trees is not cumbersome, and since Decision Trees can handle both discrete and continuuous variables, feature selection is not limited. They are also relatively easier to compute than other methods.

Decision Trees, however, tend to overfit if not properly tuned. They can also be inconsistent on similar data sets as they are more prone to influence by small variations in the data.

Given that the school district in this case is concerned with computational cost and resources, Decision Trees are worth exploring as they may be able to provide the most value for the cost.

Decision Tree	Training set size		
	100	200	300
Training time (secs)	0.00100	0.00100	0.00300
Prediction time (secs)	0.00000	0.00000	0.00000
F1 score for training set	0.83333	0.85816	0.86475
F1 score for test set	0.72727	0.76471	0.77372

SVM Model

Support Vector Machines (SVM) are similar to Decision Trees in that they can perform both classification and regression. They are more complicated for the layperson to understand. In a classification scenario essentially find the hyperplanes in the data that best separate the data into the correct classificaiton.

SVMs are advantageous in scenarios where there are many features in the data set. As well, kernels allow SVMs to create non-linear separation in the data which makes it more flexible to model more complicated relationships.

Some disadvantages of SVMs are that they take longer to compute and do not handle large data sets very well. As well, selecting different kernels can yield very different results.

The data provided has many features and does not have a large volume (at least large enough to cause computational problems). Thus, SVMs are a viable option especially as they may be able to model more complicated relationships in the data.

SVM	Training set size		
	100	200	300
Training time (secs)	0.00100	0.00400	0.00700
Prediction time (secs)	0.00200	0.00200	0.00500
F1 score for training set	0.85906	0.86928	0.86920
F1 score for test set	0.78378	0.77551	0.75862

Gaussian Naive Bayes Model

Naive Bayes models are used for classificaiton and though they have simplified assumptions, have traditionally performed very well against more complex models. Naive Bayes assumes that every set of features is independent of one another, which is often a "naive" assumption. In the case of this data set, the Naive Bayes model would compare probabilities of a student failing or passing. Those probabilies are based on the frequencies and combinations of the feature values in the data set.

Naive Bayes are helpful in that they are not as taxing on computational time as other models. They perform fairly well and have little restrictions on data volume.

Since Naive Bayes models create probabilities based on the frequency and combination of features in the data set, it can be prone to error on feature combinations it has not seen before. As well, the probabilities created by the model are not necessarily useful for anything other than classification. This may make it harder to understand for the layman.

Naive Bayes is worth exploring for many of the same reasons Decision Trees are - they are easy to compute while also providing good classification performance. Given the school district's restraints on cost, this model should be considered.

Gaussian Naïve Bayes	Training set size		
	100	200	300
Training time (secs)	0.00100	0.00100	0.00100
Prediction time (secs)	0.00100	0.00100	0.00100
F1 score for training set	0.85496	0.83206	0.80882
F1 score for test set	0.74809	0.71318	0.75000

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?

  After training a Decision Tree, Support Vector Model, and Gaussian Naive Bayes Model, I would suggest using the Decision Tree model. The Decision Tree performed as well as any other model while also taking the least amount of time to compute. Based purely on model performance on the data given, I believe the Decision Tree is the right decision as it outperformed the other models. An added benefit of the model is that it takes less computational resources to run, and given our concerns around cost and computational time, the Decision Tree is the logical decision.

- 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).

  Decision Trees, while quite powerful, are beautiful in their simplicity. The Decision Tree algorithm looks at the input variables of a data set (or features) and looks for the optimal split variable. 'Optimal' in our circumstance would be the variable that best splits students who fail and students who pass. After splitting once, it follows both paths and continues to split. For example, if the algorithm finds that the gender of the students is the best first split, it will go to all female students and decide on the best split for that group. Likewise it will do the same for the male students. Once a 'tree' of optimal split variables is built, future predictions are created by following the tree for a given student.

- 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?

  0.785

In [23]:
```
# TODO: Fine-tune your model and report the best F1 score
from sklearn.metrics import make_scorer
from sklearn import grid_search

#y_train = y_train.replace(['yes', 'no'], [1, 0])
#y_test = y_test.replace(['yes', 'no'], [1, 0])

def fit_model(X, y):

    # Create a decision tree regressor object
    regressor = DecisionTreeClassifier()

    # Set up the parameters we wish to tune
    parameters = {'max_depth':(1,2,3,4,5,6,7,8,9,10)}

    # Make an appropriate scoring function
    scoring_function = make_scorer(f1_score, pos_label='yes')

    # Make the GridSearchCV object
    reg = grid_search.GridSearchCV(regressor, parameters, scoring=s
coring_function)

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

    # Return the optimal model
    return reg.best_estimator_

clf = fit_model(X_train, y_train)
```

In [24]:
```
# Predict on training set and compute F1 score
train_f1_score = predict_labels(clf, X_train, y_train)
print "F1 score for training set: {}".format(train_f1_score)
```

```
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.829493087558
```

In [25]:
```
# Predict on test data
print "F1 score for test set: {}".format(predict_labels(clf, X_tes
t, y_test))
```

```
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.785185185185
```

In [ ]:
```

```