In [ ]:
```
# 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 __future__ import division
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 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.index)
n_features = len(student_data[:30])
n_passed = student_data.groupby('passed').size()[1]
n_failed = student_data.groupby('passed').size()[0]
grad_rate = (n_passed / 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 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 [15]:
```
# 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 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_test, y_train, y_test = train_test_split(X_all, y_all, test_s
ize = 0.24, random_state = 2 )
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 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 [16]:
```
# 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): {:.6f}".format(end - start)

# TODO: Choose a model, import it and instantiate an object
from sklearn.svm import SVC
clf = SVC()

# 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
print "clf output: {} samples".format(clf)
```

```
Training SVC...
Done!
Training time (secs): 0.003000
clf output: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, de
gree=3, gamma=0.0,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False) samples
```

In [17]:

```
# 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): {:.6f}".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 SVC...
Done!
Prediction time (secs): 0.002000
F1 score for training set: 0.850427350427
```

In [18]:

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

```
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001000
F1 score for test set: 0.826666666667
```

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_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
print "train/test set 100: {}".format(train_predict(clf, X_train[:100],
y_train[:100], X_test[:100], y_test[:100]))
print "train/test set 200: {}".format(train_predict(clf, X_train[:200],
y_train[:200], X_test[:200], y_test[:200]))
print "train/test set 300: {}".format(train_predict(clf, X_train[:300],
y_train[:300], X_test[:100], y_test[:300]))
```

```
------------------------------------------
Training set size: 100
Training SVC...
Done!
Training time (secs): 0.001000
Predicting labels using SVC...
Done!
Prediction time (secs): 0.000000
F1 score for training set: 0.833333333333
Predicting labels using SVC...
Done!
Prediction time (secs): 0.000000
F1 score for test set: 0.825
train/test set 100: None
------------------------------------------
Training set size: 200
Training SVC...
Done!
Training time (secs): 0.002000
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001000
F1 score for training set: 0.842767295597
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001000
F1 score for test set: 0.846153846154
train/test set 200: None
------------------------------------------
Training set size: 300
Training SVC...
Done!
Training time (secs): 0.003000
Predicting labels using SVC...
Done!
Prediction time (secs): 0.003000
F1 score for training set: 0.847965738758
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001000
F1 score for test set: 0.855263157895
train/test set 300: None
```

In [19]:

```
# TODO: Train and predict using two other models

# train Decision tree
print"**********************-----Decision Tree Training-Start----******
*****************"

from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()

# 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
print "clf output: {} samples".format(clf)

# 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)

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

# TODO: Run the helper function above for desired subsets of training da
ta
# Note: Keep the test set constant
print "train/test set 100: {}".format(train_predict(clf, X_train[:100],
y_train[:100], X_test[:100], y_test[:100]))
print "train/test set 200: {}".format(train_predict(clf, X_train[:200],
y_train[:200], X_test[:200], y_test[:200]))
print "train/test set 300: {}".format(train_predict(clf, X_train[:300],
y_train[:300], X_test[:100], y_test[:300]))

print"**********************-----Decision Tree Training-End----********
***************"



# train KNN
print"**********************-----KNN Training-Start----***************
*******"
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier()

# 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
print "clf output: {} samples".format(clf)

# 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)
```

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

# TODO: Run the helper function above for desired subsets of training da
ta
# Note: Keep the test set constant
print "train/test set 100: {}".format(train_predict(clf, X_train[:100],
y_train[:100], X_test[:100], y_test[:100]))
print "train/test set 200: {}".format(train_predict(clf, X_train[:200],
y_train[:200], X_test[:200], y_test[:200]))
print "train/test set 300: {}".format(train_predict(clf, X_train[:300],
y_train[:300], X_test[:100], y_test[:300]))

print"**********************-----KNN Training-End----*****************
*****"
```

```
**********************-----Decision Tree Training-Start----*********
**************
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.001000
clf output: DecisionTreeClassifier(class_weight=None, criterion='gin
i', max_depth=None,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=
1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            random_state=None, splitter='best') samples
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000000
F1 score for training set: 1.0
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000000
F1 score for test set: 0.779411764706
------------------------------------------
Training set size: 100
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.000000
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000000
F1 score for training set: 1.0
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000000
F1 score for test set: 0.692913385827
train/test set 100: None
------------------------------------------
Training set size: 200
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.001000
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000000
F1 score for training set: 1.0
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000000
F1 score for test set: 0.698412698413
train/test set 200: None
------------------------------------------
Training set size: 300
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.001000
Predicting labels using DecisionTreeClassifier...
```

```
Done!
Prediction time (secs): 0.000000
F1 score for training set: 1.0
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000000
F1 score for test set: 0.785714285714
train/test set 300: None
**********************-----Decision Tree Training-End----***********
***********
**********************-----KNN Training-Start----******************
****
Training KNeighborsClassifier...
Done!
Training time (secs): 0.001000
clf output: KNeighborsClassifier(algorithm='auto', leaf_size=30, metr
ic='minkowski',
           metric_params=None, n_neighbors=5, p=2, weights='uniform')
samples
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.003000
F1 score for training set: 0.855835240275
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.001000
F1 score for test set: 0.814285714286
------------------------------------------
Training set size: 100
Training KNeighborsClassifier...
Done!
Training time (secs): 0.000000
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.001000
F1 score for training set: 0.850322580645
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.001000
F1 score for test set: 0.838461538462
train/test set 100: None
------------------------------------------
Training set size: 200
Training KNeighborsClassifier...
Done!
Training time (secs): 0.001000
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.001000
F1 score for training set: 0.785454545455
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.001000
```

```
F1 score for test set: 0.811594202899
train/test set 200: None
------------------------------------------
Training set size: 300
Training KNeighborsClassifier...
Done!
Training time (secs): 0.000000
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.004000
F1 score for training set: 0.855835240275
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.001000
F1 score for test set: 0.814285714286
train/test set 300: None
***********************-----KNN Training-End----*********************
**
```

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 F1 score?

In [ ]:

In [20]:
```
# TODO: Fine-tune your model and report the best F1 score

from sklearn import grid_search
from sklearn import cross_validation

from sklearn.metrics import make_scorer
# TODO: Choose a model, import it and instantiate an object
from sklearn.svm import SVC
knn = KNeighborsClassifier()
score_val = make_scorer(f1_score, pos_label="yes")
parameters = [{'weights': ['uniform','distance'], 'n_neighbors': [10, 2
0, 30, 40, 50, 60, 70, 80, 90, 100]
               ,'algorithm':('ball_tree','kd_tree','brute'),}]
clf = grid_search.GridSearchCV(knn, parameters, cv=10, scoring = score_v
al)
train_classifier(clf, X_train, y_train)
train_f1_score = predict_labels(clf, X_train, y_train)
#print clf.grid_scores_
print "F1 score for training set: {}".format(train_f1_score)
print "F1 score for test set: {}".format(predict_labels(clf, X_test, y_t
est))
```

```
Training GridSearchCV...
Done!
Training time (secs): 1.273000
Predicting labels using GridSearchCV...
Done!
Prediction time (secs): 0.002000
F1 score for training set: 1.0
Predicting labels using GridSearchCV...
Done!
Prediction time (secs): 0.001000
F1 score for test set: 0.828947368421
```

In [ ]: