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
```

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 = sum(student_data['passed'] == 'yes')
n_failed = sum(student_data['passed'] == 'no')
grad_rate = float(n_passed) / (n_passed + n_failed) * 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', 'nurs
ery', 'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Walc', 'health', 'abs
ences']
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', 'famsize
_LE3', 'Pstatus_A', 'Pstatus_T', 'Medu', 'Fedu', 'Mjob_at_home', 'Mjob_health', 'Mjob_other', 'Mjob_s
ervices', 'Mjob_teacher', 'Fjob_at_home', 'Fjob_health', 'Fjob_other', 'Fjob_services', 'Fjob_teache
r', 'reason_course', 'reason_home', 'reason_other', 'reason_reputation', 'guardian_father', 'guardian
_mother', 'guardian_other', 'traveltime', 'studytime', 'failures', 'schoolsup', 'famsup', 'paid', 'ac
tivities', 'nursery', 'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Wal
c', '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
indices = range(num_all)
import random
random.seed(1)
random.shuffle(indices)
X_train = pd.DataFrame(X_all, index=indices[:num_train])
y_train = pd.DataFrame(student_data['passed'], index=indices[:num_train])
X_test = pd.DataFrame(X_all, index=indices[-num_test:])
y_test = pd.DataFrame(student_data['passed'], index=indices[-num_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 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 object
from sklearn.neighbors import KNeighborsClassifier
import sklearn.svm as svm
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB

clf1_100 = KNeighborsClassifier() #clf1 with 100 inputs
clf1_200 = KNeighborsClassifier() #clf1 with 200 inputs
clf1_300 = KNeighborsClassifier() #clf1 with 300 inputs

clf2_100 = svm.SVC() #clf2 with 100 inputs
clf2_200 = svm.SVC() #clf2 with 200 inputs
clf2_300 = svm.SVC() #clf2 with 300 inputs

clf3_100 = DecisionTreeClassifier() #clf3 with 100 inputs
clf3_200 = DecisionTreeClassifier() #clf3 with 200 inputs
clf3_300 = DecisionTreeClassifier() #clf3 with 300 inputs

# Fit model to training data
train_classifier(clf1_100, X_train[:100], y_train[:100])  # note: using entire training set here
train_classifier(clf1_200, X_train[:200], y_train[:200])  # note: using entire training set here
train_classifier(clf1_300, X_train[:300], y_train[:300])  # note: using entire training set here

train_classifier(clf2_100, X_train[:100], y_train[:100])  # note: using entire training set here
train_classifier(clf2_200, X_train[:200], y_train[:200])  # note: using entire training set here
train_classifier(clf2_300, X_train[:300], y_train[:300])  # note: using entire training set here

train_classifier(clf3_100, X_train[:100], y_train[:100])  # note: using entire training set here
train_classifier(clf3_200, X_train[:200], y_train[:200])  # note: using entire training set here
train_classifier(clf3_300, X_train[:300], y_train[:300])  # note: using entire training set here

# train_classifier(clf, X_train, y_train)  # note: using entire training set here
print clf1_100  # you can inspect the learned model by printing it
```

```
as passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
/Library/Python/2.7/site-packages/ipykernel/__main__.py:7: DataConversionWarning: A column-vector y w
as passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
/Library/Python/2.7/site-packages/ipykernel/__main__.py:7: DataConversionWarning: A column-vector y w
as passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
/Library/Python/2.7/site-packages/sklearn/svm/base.py:514: DataConversionWarning: A column-vector y w
as passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  y_ = column_or_1d(y, warn=True)
/Library/Python/2.7/site-packages/sklearn/svm/base.py:514: DataConversionWarning: A column-vector y w
as passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  y_ = column_or_1d(y, warn=True)
/Library/Python/2.7/site-packages/sklearn/svm/base.py:514: DataConversionWarning: A column-vector y w
as passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  y_ = column_or_1d(y, warn=True)
```

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 "Prediction 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)

print "F1 score for clf1_100 training set: {}".format(predict_labels(clf1_100, X_train[:100], y_train[:10
print "F1 score for clf1_200 training set: {}".format(predict_labels(clf1_200, X_train[:200], y_train[:20
print "F1 score for clf1_300 training set: {}".format(predict_labels(clf1_300, X_train[:300], y_train[:30

print "F1 score for clf2_100 training set: {}".format(predict_labels(clf2_100, X_train[:100], y_train[:10
print "F1 score for clf2_200 training set: {}".format(predict_labels(clf2_200, X_train[:200], y_train[:20
print "F1 score for clf2_300 training set: {}".format(predict_labels(clf2_300, X_train[:300], y_train[:30

print "F1 score for clf3_100 training set: {}".format(predict_labels(clf3_100, X_train[:100], y_train[:10
print "F1 score for clf3_200 training set: {}".format(predict_labels(clf3_200, X_train[:200], y_train[:20
print "F1 score for clf3_300 training set: {}".format(predict_labels(clf3_300, X_train[:300], y_train[:30
```

```
Prediction time (secs): 0.003
F1 score for clf1_100 training set: 0.818791946309
Prediction time (secs): 0.005
F1 score for clf1_200 training set: 0.874213836478
Prediction time (secs): 0.010
F1 score for clf1_300 training set: 0.87032967033
Prediction time (secs): 0.001
F1 score for clf2_100 training set: 0.875816993464
Prediction time (secs): 0.004
F1 score for clf2_200 training set: 0.870090634441
Prediction time (secs): 0.007
F1 score for clf2_300 training set: 0.869022869023
Prediction time (secs): 0.001
F1 score for clf3_100 training set: 1.0
Prediction time (secs): 0.001
F1 score for clf3_200 training set: 1.0
Prediction time (secs): 0.001
F1 score for clf3_300 training set: 1.0
```

In [9]:
```
# Predict on test data
print "F1 score for clf1_100 test set: {}".format(predict_labels(clf1_100, X_test, y_test))
print "F1 score for clf1_200 test set: {}".format(predict_labels(clf1_200, X_test, y_test))
print "F1 score for clf1_300 test set: {}".format(predict_labels(clf1_300, X_test, y_test))

print "F1 score for clf2_100 test set: {}".format(predict_labels(clf2_100, X_test, y_test))
print "F1 score for clf2_200 test set: {}".format(predict_labels(clf2_200, X_test, y_test))
print "F1 score for clf2_300 test set: {}".format(predict_labels(clf2_300, X_test, y_test))

print "F1 score for clf3_100 test set: {}".format(predict_labels(clf3_100, X_test, y_test))
print "F1 score for clf3_200 test set: {}".format(predict_labels(clf3_200, X_test, y_test))
print "F1 score for clf3_300 test set: {}".format(predict_labels(clf3_300, X_test, y_test))
```

```
Prediction time (secs): 0.003
F1 score for clf1_100 test set: 0.762589928058
Prediction time (secs): 0.003
F1 score for clf1_200 test set: 0.735294117647
Prediction time (secs): 0.004
F1 score for clf1_300 test set: 0.77037037037
Prediction time (secs): 0.002
F1 score for clf2_100 test set: 0.743243243243
Prediction time (secs): 0.002
F1 score for clf2_200 test set: 0.748299319728
Prediction time (secs): 0.002
F1 score for clf2_300 test set: 0.780141843972
Prediction time (secs): 0.000
F1 score for clf3_100 test set: 0.672268907563
Prediction time (secs): 0.000
F1 score for clf3_200 test set: 0.727272727273
Prediction time (secs): 0.000
F1 score for clf3_300 test set: 0.754098360656
```

In [10]:
```
# Train and predict using different training set sizes
# Already done in previous sections so commenting this part out
```

In [11]:
```
# TODO: Train and predict using two other models
# Already done in previous sections so commenting this part out
```

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 [12]:
```
# TODO: Fine-tune your model and report the best F1 score
from sklearn.grid_search import GridSearchCV
tuned_parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10, 100], 'gamma':[0.001, 0.01] }

from sklearn.metrics import f1_score
from sklearn.metrics import make_scorer
f1_scorer = make_scorer(f1_score, pos_label="yes")
svc = svm.SVC()
grid_search = GridSearchCV(svc, tuned_parameters, scoring=f1_scorer)
grid_search.fit(X_train, y_train[:300].values.reshape(300))

# from array import array
print "Fine tuned CLF is : {}".format(grid_search)
print grid_search.get_params()
print "F1 score on test set for this CLF is: {}".format(predict_labels(grid_search, X_test, y_test))
```

```
Fine tuned CLF is : GridSearchCV(cv=None, error_score='raise',
       estimator=SVC(C=1.0, 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),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'kernel': ('linear', 'rbf'), 'C': [1, 10, 100], 'gamma': [0.001, 0.01]},
       pre_dispatch='2*n_jobs', refit=True,
       scoring=make_scorer(f1_score, pos_label=yes), verbose=0)
{'n_jobs': 1, 'verbose': 0, 'estimator__gamma': 'auto', 'estimator__decision_function_shape': None,
'estimator__probability': False, 'param_grid': {'kernel': ('linear', 'rbf'), 'C': [1, 10, 100], 'gamm
a': [0.001, 0.01]}, 'cv': None, 'scoring': make_scorer(f1_score, pos_label=yes), 'estimator__cache_si
ze': 200, 'estimator__verbose': False, 'pre_dispatch': '2*n_jobs', 'estimator__kernel': 'rbf', 'fit_p
arams': {}, 'estimator__max_iter': -1, 'refit': True, 'iid': True, 'estimator__shrinking': True, 'est
imator__degree': 3, 'estimator__class_weight': None, 'estimator__C': 1.0, 'estimator__random_state':
None, 'estimator': SVC(C=1.0, 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), 'estimator__coef0': 0.0, 'error_score': 'raise', 'estimator__tol': 0.00
1}
Prediction time (secs): 0.003
F1 score on test set for this CLF is: 0.738255033557
```