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: In this assignment, we have to predict whether a student will 'pass' or 'fail' from a given set of features. This is a classification problem because we have to classify students into distinct classes. If it is a regression problem, we have to predict continuous output. Therefore it could be a regression problem if we want to predict the score of final exam.

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 [109]:

```
# Import libraries
import numpy as np
import pandas as pd
from sklearn.cross_validation import train_test_split
from sklearn.cross_validation import StratifiedShuffleSplit
```

In [98]:

```
# Read student data
student_data = pd.read_csv("C:/Users/Mohammad/git/nanodegree/1603student
_intervention/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 [99]:
```
#student_data.describe()
#student_data.shape[0]
#print len(student_data)
x = student_data.passed.value_counts()
x['yes']
#student_data.head(1)
```

Out[99]: 265

In [101]:
```
# TODO: Compute desired values - replace each '?' with an appropriate ex
pression/function call
n_students = student_data.shape[0]
n_features = student_data.shape[1]

x = student_data.passed.value_counts()

n_passed = x['yes']
n_failed = x['no']
grad_rate = 100*float(n_passed)/float(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: 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 [110]:
```
# 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
print y_all.describe()
```

```
Feature column(s):-
['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fed
u', 'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime', 'fa
ilures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'highe
r', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Wal
c', '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 h
ealth  \
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]
count     395
unique      2
top       yes
freq      265
Name: passed, dtype: object
```

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 [124]:
```
# 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', 'addres
s_U', 'famsize_GT3', 'famsize_LE3', 'Pstatus_A', 'Pstatus_T', 'Medu', 'F
edu', 'Mjob_at_home', 'Mjob_health', 'Mjob_other', 'Mjob_services', 'Mjo
b_teacher', 'Fjob_at_home', 'Fjob_health', 'Fjob_other', 'Fjob_service
s', 'Fjob_teacher', 'reason_course', 'reason_home', 'reason_other', 'rea
son_reputation', 'guardian_father', 'guardian_mother', 'guardian_other',
'traveltime', 'studytime', 'failures', 'schoolsup', 'famsup', 'paid', 'a
ctivities', 'nursery', 'higher', 'internet', 'romantic', 'famrel', 'free
time', '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 [125]:
```
# 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
#student_data['passed'] = format(student_data['passed'])
y = student_data['passed']

def shuffle_split_data(X,y,num_train):
    s = StratifiedShuffleSplit(y, 1, train_size=num_train)

    # only one iteration
    for train_index, test_index in s:
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    return X_train, X_test, y_train, y_test

X_train, X_test, y_train, y_test = shuffle_split_data(X_all, y,num_train
)

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: 40 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.

Ans:

Data characteristics: The dataset contains data from 395 students. Every student data contain 31 features. Target label 'Passed' contains categorical variable: 'yes' and 'no'. Among the students, 265 passed and 135 failed. Graduation rate of the class is around 67.09%.

There are several issues with the dataset.

- The number of features is pretty large compare to number of students. In order to classify accurately, the number of training instances needed to increase exponentially as the number of features increase. Therefore, the prediction model will suffer from the curse of dimensionality [1] and the prediction/classification might be overfitted.
- The number students passed and failed are not equal. Some models might be performed poorly under this imbalanced condition. Furthermore, when we just simply split data into test and training sets, there are possibilities when the training set might contain very failed student data and as a result, the prediction model will perform poorly. In order overcome this issue, we applied Stratified Shuffle Split from scikit-learn. Stratified shuffle split ensures the constant ratio of passed and failed students in test and training datasets.

Accuracy measurement: Accuracy of this prediction model is measured by F1 score. F1 score takes into account both precision and recall scores.

In [126]:
```
# 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)
    return end-start

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

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

```
Training SVC...
Done!
Training time (secs): 0.009
```

Out[126]: 0.009000062942504883

In [127]:
```
# 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'), end-start

train_f1_score,train_time = predict_labels(svc_clf, X_train, y_train)
print "F1 score for training set: {}".format(train_f1_score)
```

```
Predicting labels using SVC...
Done!
Prediction time (secs): 0.006
F1 score for training set: 0.8410041841
```

In [81]:
```
# Predict on test data
test_f1_score,test_time = predict_labels(svc_clf, X_test, y_test)
print "F1 score for test set: {}".format(test_f1_score)
```

```
Predicting labels using SVC...
Done!
Prediction time (secs): 0.002
F1 score for test set: 0.87898089172
```

In [23]:
```
# 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))
    x=train_classifier(clf, X_train, y_train)
    f1_score_train,train_time= predict_labels(clf, X_train, y_train)
    print "F1 score for training set: {}".format(f1_score_train)
    f1_score_test,test_time= predict_labels(clf, X_test, y_test)
    print "F1 score for training set: {}".format(f1_score_test)
    return (f1_score_train, f1_score_test, x,test_time)

# TODO: Run the helper function above for desired subsets of training da
ta
# Note: Keep the test set constant
```

In [128]:
```
# TODO: Train and predict using two other models

from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(n_estimators=200)

# Fit model to training data
#train_classifier(rf_clf, X_train, y_train)  # note: using entire traini
ng set here
#print rf_clf
train_predict(rf_clf, X_train, y_train, X_test, y_test)
```

```
------------------------------------------
Training set size: 300
Training RandomForestClassifier...
Done!
Training time (secs): 0.176
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.021
F1 score for training set: 1.0
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.016
F1 score for training set: 0.847457627119
```

Out[128]:
```
(1.0, 0.8474576271186403, 0.17599987983703613, 0.016000032424926758)
```

In [129]:
```
from sklearn.naive_bayes import MultinomialNB
nb_clf = MultinomialNB(alpha=30)
print nb_clf
train_predict(nb_clf, X_train, y_train, X_test, y_test)
```

```
MultinomialNB(alpha=30, class_prior=None, fit_prior=True)
------------------------------------------
Training set size: 300
Training MultinomialNB...
Done!
Training time (secs): 0.003
Predicting labels using MultinomialNB...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.813417190776
Predicting labels using MultinomialNB...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.8125
```

Out[129]:
```
(0.81341719077568142, 0.8125, 0.003000020980834961, 0.0)
```

In [13]:
```
#X_train
#train_predict(nb_clf, x1, y1, X_test, y_test)
```

In [161]:

```
### Run the helper function above for desired subsets of training data
# Note: Keep the test set constant

def compare_algorithm(algorithm, size):
    for i in algorithm:
        training_time = []
        prediction_time = []
        f1_train_score = []
        f1_test_score = []
        z = ['train_size','train_time','f1_train','test_time','f1_test']
        for j in size:
            x1,x2,y1,y2 = shuffle_split_data(X_all, y,j )
            total_f1_train_score,total_f1_test_score,total_training_tim
e, total_prediction_time  =train_predict(i, x1, y1, X_test, y_test)
            training_time.append(total_training_time)
            prediction_time.append(total_prediction_time)
            f1_train_score.append(total_f1_train_score)
            f1_test_score.append(total_f1_test_score)

        tab= [size,training_time,f1_train_score,prediction_time,f1_test_
score]
        tab = pd.DataFrame(tab)
        tab[' ']=z
        tab = tab[[' ',0,1,2]]
        print "----------------------------------------"
        print "Summary Table"
        print "----------------------------------------"
        print tab

compare_algorithm([nb_clf,rf_clf,svc_clf],[100,200,300])
```

```
------------------------------------------
Training set size: 100
Training MultinomialNB...
Done!
Training time (secs): 0.001
Predicting labels using MultinomialNB...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.802395209581
Predicting labels using MultinomialNB...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.805970149254
------------------------------------------
Training set size: 200
Training MultinomialNB...
Done!
Training time (secs): 0.001
Predicting labels using MultinomialNB...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.798761609907
Predicting labels using MultinomialNB...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.787878787879
------------------------------------------
Training set size: 300
Training MultinomialNB...
Done!
Training time (secs): 0.002
Predicting labels using MultinomialNB...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.810126582278
Predicting labels using MultinomialNB...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.8125
------------------------------------------
Summary Table
------------------------------------------
                       0           1           2
0  train_size  100.000000  200.000000  300.000000
1  train_time    0.001000    0.001000    0.002000
2    f1_train    0.802395    0.798762    0.810127
3   test_time    0.000000    0.000000    0.000000
4     f1_test    0.805970    0.787879    0.812500
------------------------------------------
Training set size: 100
Training RandomForestClassifier...
Done!
Training time (secs): 0.151
```

```
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.017
F1 score for training set: 1.0
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.009
F1 score for training set: 0.813559322034
------------------------------------------
Training set size: 200
Training RandomForestClassifier...
Done!
Training time (secs): 0.151
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.028
F1 score for training set: 1.0
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.011
F1 score for training set: 0.866666666667
------------------------------------------
Training set size: 300
Training RandomForestClassifier...
Done!
Training time (secs): 0.133
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.017
F1 score for training set: 1.0
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.029
F1 score for training set: 0.896551724138
------------------------------------------
Summary Table
------------------------------------------
                        0           1           2
0  train_size  100.000000  200.000000  300.000000
1  train_time    0.151000    0.151000    0.133000
2    f1_train    1.000000    1.000000    1.000000
3   test_time    0.009000    0.011000    0.029000
4     f1_test    0.813559    0.866667    0.896552
------------------------------------------
Training set size: 100
Training SVC...
Done!
Training time (secs): 0.001
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.887417218543
Predicting labels using SVC...
```

```
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.830769230769
------------------------------------------
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: 0.863636363636
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.830769230769
------------------------------------------
Training set size: 300
Training SVC...
Done!
Training time (secs): 0.011
Predicting labels using SVC...
Done!
Prediction time (secs): 0.007
F1 score for training set: 0.90134529148
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.870967741935
------------------------------------------
Summary Table
------------------------------------------
                       0           1           2
0  train_size  100.000000  200.000000  300.000000
1  train_time    0.001000    0.004000    0.011000
2    f1_train    0.887417    0.863636    0.901345
3   test_time    0.000000    0.001000    0.001000
4     f1_test    0.830769    0.830769    0.870968
```

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 [164]:
```
# TODO: Fine-tune your model and report the best F1 score
y_train
```

Out[164]:
```
280     no
180     no
140     no
139    yes
14     yes
171    yes
68      no
93     yes
89      no
254    yes
350     no
78     yes
319    yes
256    yes
316     no
...
100     no
42     yes
13     yes
81     yes
264     no
53     yes
16     yes
366    yes
196    yes
92      no
204    yes
65     yes
121    yes
334     no
129    yes
Name: passed, Length: 300, dtype: object
```