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?

Answer:

Classification model is suitable to predict discrete labels and regression model is to predict continuous labels. In this case, the model uses student's features to predict 'Pass' or 'No Pass' and these are discrete lables.Therefore I choose classification model.

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 [43]:

```
# Import libraries
import numpy as np
import pandas as pd
```

In [44]:

```
# 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 [45]:

```
# TODO: Compute desired values - replace each '?' with an appropriate expression/function call
n_students = len(student_data)
n_features = student_data.shape[1]-1
n_passed = len(student_data.ix[(student_data['passed']=='yes')])
n_failed = len(student_data.ix[(student_data['passed']=='no')])
grad_rate = n_passed * 100.0 / 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: 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 [46]:

```
# 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', 'Fjo
b', 'reason', 'guardian', 'traveltime', 'studytime', 'failures', 'schoolsup', 'famsu
p', 'paid', 'activities', 'nursery', 'higher', 'internet', 'romantic', 'famrel', 'freetim
e', '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 [47]:

```
# 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', 'fa
msize_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_services', 'Fjob_teacher', 'reason_c
ourse', 'reason_home', 'reason_other', 'reason_reputation', 'guardian_father',
'guardian_mother', 'guardian_other', 'traveltime', 'studytime', 'failures', 'schoo
lsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet', 'romantic', 'fam
rel', '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 [48]:

```
# 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
from sklearn.cross_validation import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X_all,y_all,test_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 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 [49]:

```
# 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 "|TrainTime:{:.3f}|".format(end - start)
    return end-start
    #print "Done!\nTraining time (secs): {:.3f}".format(end - start)

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

# 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
```

```
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)
```

In [50]:

```
# 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)
    #print "|PredTime{:.3f}|".format(end - start)
    return [(end-start),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: [0.009914875030517578, 0.8691983122362869
6]
```

In [51]:

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

```
F1 score for test set: [0.001672983169555664, 0.75862068965517238]
```

In [52]:

```
# 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_time=train_classifier(clf, X_train, y_train)
    #print "F1 score for training set: {}".format(predict_labels(clf, X_train, y_train))
    #print "F1 score for test set: {}".format(predict_labels(clf, X_test, y_test))
    train_result=predict_labels(clf, X_train, y_train)
    test_result=predict_labels(clf, X_test, y_test)
    print "| {} | {:.3f} | {:.3f} | {:.3f}| {:.5f} | {:.5f} |".format(len(X_train) , train_time, train_result
[0],test_result[0],train_result[1],test_result[1] )



# TODO: Run the helper function above for desired subsets of training data
clf =SVC()
print " | {} | Training Time | Train Pred Time | Test Pred Time | F1_Train | F1_Test |".format(clf.
__class__.__name__)
print "  | --- | -- | -- | --| -- | -- |"
for x, i in enumerate([100,200,300]):
    train_predict(clf,X_train[:i],y_train[:i],X_test,y_test)
# Note: Keep the test set constant
```

```
 | SVC | Training Time | Train Pred Time | Test Pred Time | F1_Train | F1_Test |
  | --- | -- | -- | --| -- | -- |
| 100 | 0.001 | 0.001 | 0.001| 0.85906 | 0.78378 |
| 200 | 0.003 | 0.003 | 0.001| 0.86928 | 0.77551 |
| 300 | 0.007 | 0.007 | 0.002| 0.86920 | 0.75862 |
```

In [ ]:

In [53]:

```
# TODO: Train and predict using two other models
#GNB
#from sklearn.naive_bayes import GaussianNB
#clf2 = GaussianNB()
from sklearn.naive_bayes import MultinomialNB
clf2 = MultinomialNB()
print " | {} | Training Time | Train Pred Time | Test Pred Time | F1_Train | F1_Test |".format(clf
2.__class__.__name__)
print "  | --- | -- | -- | --| -- | -- |"
for x, i in enumerate([100,200,300]):
    train_predict(clf2,X_train[:i],y_train[:i],X_test,y_test)

#Neighbors
from sklearn import neighbors
clf3 = neighbors.KNeighborsClassifier()
print " | {} | Training Time | Train Pred Time | Test Pred Time | F1_Train | F1_Test |".format(clf
3.__class__.__name__)
print "  | --- | -- | -- | --| -- | -- |"
for x, i in enumerate([100,200,300]):
    train_predict(clf3,X_train[:i],y_train[:i],X_test,y_test)
```

```
 | MultinomialNB | Training Time | Train Pred Time | Test Pred Time | F1_Train | F
1_Test |
  | --- | -- | -- | --| -- | -- |
| 100 | 0.002 | 0.000 | 0.000| 0.78195 | 0.76471 |
| 200 | 0.002 | 0.001 | 0.000| 0.79004 | 0.77698 |
| 300 | 0.002 | 0.000 | 0.000| 0.78505 | 0.77698 |
 | KNeighborsClassifier | Training Time | Train Pred Time | Test Pred Time | F1_Tr
ain | F1_Test |
  | --- | -- | -- | --| -- | -- |
| 100 | 0.001 | 0.004 | 0.002| 0.79720 | 0.70677 |
| 200 | 0.001 | 0.005 | 0.004| 0.85714 | 0.71212 |
| 300 | 0.003 | 0.009 | 0.003| 0.87225 | 0.74820 |
```

Answer: The pros/cons of models

I have choosed SVC, K Neighborhood,and Gausian NaiveBaise. All models are classification models.

Model	Application	Pros	Cons
SVC	This model to find decidion boundary that maximizes the margin between closest opposite labels.	Compared to K Neighborhood, It doesn't require remembering whole data points.	It needs more training time to calculate the boundary.
K-Neighborhood	The model uses the label of the closest traing data point to input data.	It is less complex. And we don't need much training time.	It needs to remember whole data points. It is bad if the number of data points increase
Multinominal Naive Baise	Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes’ theorem with the “naive” assumption of independence between every pair of features.	It supports online update.	training time may cost bigger than Neighborhood.

Result:

(All time are in seconds)

SVC-TrainingSet	Training Time	Train Pred Time	Test Pred Time	F1_Train	F1_Test
100	0.002	0.001	0.001	0.85906	0.78378
200	0.004	0.003	0.001	0.86928	0.77551
300	0.007	0.005	0.002	0.86920	0.75862

KNeighborsClassifier	Training Time	Train Pred Time	Test Pred Time	F1_Train	F1_Test
100	0.001	0.002	0.001	0.79720	0.70677
200	0.001	0.003	0.002	0.85714	0.71212
300	0.001	0.006	0.003	0.87225	0.74820

MultinomialNB	Training Time	Train Pred Time	Test Pred Time	F1_Train	F1_Test
100	0.001	0.000	0.000	0.78195	0.76471
200	0.001	0.000	0.000	0.79004	0.77698
300	0.001	0.000	0.000	0.78505	0.77698

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?

Answer

What model chosen

- Comparing 3 models of SVC, KNeighborsClassifiler(KNC),and Multinominal NaiveBayse (MNB) and I recommend MNB Classifier because of 3 reasons.

1) Training Time. SVC consumes training time as number of training data increases but the other 2 models stays faster level.(0.001sec). Considering the number of students will increase, I remove SVC from my choise.

2) F1 score. Comparing F1_Test of MNB (=0.77698) and KNC(=0.74820), GNB has better F1_Test score.

3) Fast Prediction time. Comparing Test Pred Time of GNB and KNC, KNC is always very slower as the number of data increase. It is because KNC needs to see whole datapoints. Considering those facts, MNB is the best choise.

How the model works

- Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. Here is the step of prediction.

1)Calculate probabilities for each attribute, conditional on the class value.

2)Use the product rule to obtain a joint conditional probability for the attributes.

3)Use Bayes rule to derive conditional probabilities for the class variable.

4)Once this has been done for all class values, output the class with the highest probability.

In [54]:

```
# TODO: Fine-tune your model and report the best F1 score
```

In [61]:

```
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import f1_score, make_scorer
print clf2
clf4 = MultinomialNB()
parameters = {'alpha':(1,0.1,0.001,0.0001,0.00001),'fit_prior':(True,False),'class_prior':(No
ne,[0.5,0.5],[0.3,0.7],[0.7,0.3])}
sf_f1 = make_scorer(f1_score,pos_label='yes')
reg = GridSearchCV(clf4,parameters,scoring=sf_f1)

print " | {} | Training Time | Train Pred Time | Test Pred Time | F1_Train | F1_Test |".format(reg.
__class__.__name__)
print "  | --- | -- | -- | --| -- | -- |"
train_predict(reg,X_train,y_train,X_test,y_test)

print "Final model optimal parameters:", reg.best_params_
```

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
 | GridSearchCV | Training Time | Train Pred Time | Test Pred Time | F1_Train | F1
_Test |
  | --- | -- | -- | --| -- | -- |
| 300 | 0.492 | 0.000 | 0.000| 0.79070 | 0.76596 |
Final model optimal parameters: {'alpha': 0.1, 'fit_prior': True, 'class_prior': [0.
3, 0.7]}
```

Finetune result

GridSearchCV found the best parameters as {'alpha': 0.1, 'fit_prior': True, 'class_prior': [0.3, 0.7]} for MultinominalNB.

GridSearchCV	Training Time	Train Pred Time	Test Pred Time	F1_Train	F1_Test
300	0.485	0.000	0.000	0.79070	0.76596

Although F1_training score is 0.7970 which is slightly better than default setting achieved (0.78505),however F1_test is 0.76596 which is worse than default (0.77698). I assume some overfitting happened.

In [ ]: