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?

Classification. because classification is the concretely answer which is the answer is just yes or no. While regression answer will be the continuous number so student who might need early intervention in this supervised learning will be just "yes, he need early intervention" or "no, he don't need"

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 [5]:

```
# Import libraries
import numpy as np
import pandas as pd
from sklearn.metrics import f1_score
```

In [6]:

```
# 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 colu
mns
```

```
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 [7]:

```
# TODO: Compute desired values - replace each '?' with an appropriate expressio
n/function call
n_students = student_data.shape[0]
n_features = student_data.shape[1]-1
n_passed = student_data[student_data['passed'] == 'yes'].shape[0]
n_failed = student_data[student_data['passed'] == 'no'].shape[0]
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)
print "\nF1 score for all 'yes' on students: {:.4f}".format(
    f1_score(y_true = ['yes']*n_passed + ['no']*n_failed, y_pred = ['yes']*n_stu
dents,
             pos_label='yes', average='binary'))
```

```
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%

F1 score for all 'yes' on students: 0.8030
```

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 [8]:

```
# Extract feature (X) and target (y) columns
feature_cols = list(student_data.columns[:-1])  # all columns but last are featu
res
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', 'F
edu', 'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytim
e', 'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nurser
y', '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   tea
cher
1     GP   F   17       U     GT3       T     1     1  at_home     o
ther
2     GP   F   15       U     LE3       T     1     1  at_home     o
ther
3     GP   F   15       U     GT3       T     4     2   health  serv
ices
4     GP   F   16       U     GT3       T     3     3    other     o
ther

    ...    higher internet  romantic  famrel  freetime goout Dalc Wa
lc 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 [9]:

```
# 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_a
ll.columns))
```

```
Processed feature columns (48):-
['school_GP', 'school_MS', 'sex_F', 'sex_M', 'age', 'address_R', 'ad
dress_U', 'famsize_GT3', 'famsize_LE3', 'Pstatus_A', 'Pstatus_T', 'M
edu', 'Fedu', 'Mjob_at_home', 'Mjob_health', 'Mjob_other', 'Mjob_ser
vices', 'Mjob_teacher', 'Fjob_at_home', 'Fjob_health', 'Fjob_other',
 'Fjob_services', 'Fjob_teacher', 'reason_course', 'reason_home', 'r
eason_other', 'reason_reputation', 'guardian_father', 'guardian_moth
er', 'guardian_other', 'traveltime', 'studytime', 'failures', 'schoo
lsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'interne
t', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Walc', 'heal
th', '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 [11]:

```
# 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

y = student_data['passed']
# 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 ord
ering 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=42)

#X_train = ?
#y_train = ?
#X_test = ?
#y_test = ?
print "Training set: {} samples".format(X_train.shape[0])
print "Test set: {} samples".format(X_test.shape[0])
print "Grad rate of the training set: {:.2f}%".format(100 * (y_train == 'yes').m
ean())
print "Grad rate of the testing set: {:.2f}%".format(100 * (y_test == 'yes').mea
n())
# Note: If you need a validation set, extract it from within training data
```

```
Training set: 300 samples
Test set: 95 samples
Grad rate of the training set: 68.33%
Grad rate of the testing set: 63.16%
```

4. Training and Evaluating Models

Choose 3 supervised learning models that are available in scikit-learn, and appropriate for this problem. For each model:

I choose Decision Tree, Naive Bayes, and SVM of 3 supervised learning models

- What is the theoretical O(n) time & space complexity in terms of input size? ##### Decision Tree is O(mn . (log n)), Naive Bayes is O(N), SVM is O(N power 2)
- What are the general applications of this model? What are its strengths and weaknesses? ### Decision Tree #### Generation application
- Decision trees are normally used in operations research and operations management where the calculation is not too complex

Strengths

- white box model which can observer the model explanation by each step, easy to see what's going on behind the scene
- perform well with large datasets because of how fast while you could use just standard computing resources
- simple to understand

Weaknesses

- might get into local minimum but not global minimum
- too long time to train complexity data when compare to other algorithm

Naive Bayes

Generation application

- used for solving spam filtering, facial recognize, and segmentation each items into label categories

Strengths

- Naive Bayes models are fast and easy to train, and are fast to perform predictions.
- perform well with large datasets because of how fast while you could use just standard computing resources
- simple to understand

Weaknesses

- might get into local minimum but not global minimum
- too long time to train complexity data when compare to other algorithm

Support Vector Machine

Generation application

- text classification, data classification, face recognition

Strengths

- 100 % reaching Global minimum
- Memory Efficient
- effective when they have high dimension space

Weaknesses

- Not good for Big Data because consume too much time to train
- perform not well if data has more noise or doesn't have clear seperation

#

- Given what you know about the data so far, why did you choose this model to apply? ###### I picked 3 model which is Decision Tree, Naive Bayes, and SVM. Other that I didn't pick in this model is Neural Network and Boosting because those 2 later it's fit more for complex data like image processing.
- 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 [14]:

```
# 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
def train_withReturn(clf, X_train, y_train):
    start = time.time()
    clf.fit(X_train, y_train)
    end = time.time()
    return end - start

from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=5)

# 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
```

```
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.058
DecisionTreeClassifier(class_weight=None, criterion='gini', max_dept
h=None,
            max_features=None, max_leaf_nodes=None, min_samples_leaf
=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=5, splitter='best')
```

In [15]:

```
# 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')

def predict_withtime(clf, features, target):
    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') , float(end - start)

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): 0.008
F1 score for training set: 1.0
```

In [16]:

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

```
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.645161290323
```

In [17]:

```
# 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_train, y_
train))
    print "F1 score for test set: {}".format(predict_labels(clf, X_test,
y_test))

def train_predict_return(clf, X_train, y_train, X_test, y_test):
    print "------------------------------------------"
    print "Training set size return: {}".format(len(X_train))
    time_train = train_withReturn(clf, X_train, y_train)
    f1_train, time_predict_train = predict_withtime(clf, X_train, y_train)
    f1_test, time_predict_test = predict_withtime(clf, X_test, y_test)
    print "F1 score for training set return: {}".format(f1_train)
    print "F1 score for test set return: {}".format(f1_test)
    return  time_train,time_predict_train,time_predict_test,f1_train,f1_test



# TODO: Run the helper function above for desired subsets of training data
# Note: Keep the test set constant
time_train = list()
time_predict_train = list()
time_predict_test = list()
f1_train = list()
f1_test = list()
for x in (100,200,300):
    t1,t2,t3,t4,t5 =  train_predict_return(clf,X_train[:x], y_train[:x], X_test,
 y_test)
    time_train.append(t1)
    time_predict_train.append(t2)
    time_predict_test.append(t3)
    f1_train.append(t4)
    f1_test.append(t5)
```

```
------------------------------------------
Training set size return: 100
Done!
Prediction time (secs): 0.000
Done!
Prediction time (secs): 0.000
F1 score for training set return: 1.0
F1 score for test set return: 0.615384615385
------------------------------------------
Training set size return: 200
Done!
Prediction time (secs): 0.000
Done!
Prediction time (secs): 0.000
F1 score for training set return: 1.0
F1 score for test set return: 0.742424242424
------------------------------------------
Training set size return: 300
Done!
Prediction time (secs): 0.000
Done!
Prediction time (secs): 0.000
F1 score for training set return: 1.0
F1 score for test set return: 0.645161290323
```

In [18]:

```
# TODO: Train and predict using two other models
#Naive Bayes
print ("==============================")
print ("==============================")
print ("==============================")
print ("=========Naive Bayes==========")
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
train_classifier(gnb, X_train, y_train)
print gnb

train_f1_score = predict_labels(gnb, X_train, y_train)
print "F1 score for training set: {}".format(train_f1_score)
print "F1 score for test set: {}".format(predict_labels(gnb, X_test, y_test))

time_train2 = list()
time_predict_train2 = list()
time_predict_test2 = list()
f1_train2 = list()
f1_test2 = list()
for x in (100,200,300):
    t1,t2,t3,t4,t5 =  train_predict_return(gnb,X_train[:x], y_train[:x], X_test,
 y_test)
    time_train2.append(t1)
    time_predict_train2.append(t2)
    time_predict_test2.append(t3)
    f1_train2.append(t4)
    f1_test2.append(t5)

print ("==============================")
print ("==============================")
print ("==============================")
print ("====Support Vector Machine====")

#SVM
from sklearn import svm
svc = svm.SVC()
train_classifier(svc, X_train, y_train)
print svc

train_f1_score = predict_labels(svc, X_train, y_train)
print "F1 score for training set: {}".format(train_f1_score)
print "F1 score for test set: {}".format(predict_labels(clf, X_test, y_test))

time_train3 = list()
time_predict_train3 = list()
time_predict_test3 = list()
f1_train3 = list()
f1_test3 = list()
for x in (100,200,300):
    t1,t2,t3,t4,t5 =  train_predict_return(svc,X_train[:x], y_train[:x], X_test,
 y_test)
    time_train3.append(t1)
    time_predict_train3.append(t2)
    time_predict_test3.append(t3)
    f1_train3.append(t4)
    f1_test3.append(t5)

print("Table Comparison")
print("Comparison for Decision Tree")
print("No.    Training time    Prediction time(train)    Prediction time(test)
```

```
  F1 Score(training)    F1 Score(testing)")
print("100      %.5f            %.5f                    %.5f                %.5
f               %.5f" %
(time_train[0],time_predict_train[0],time_predict_test[0],f1_train[0],f1_test[0])

print("200      %.5f            %.5f                    %.5f                %.5
f               %.5f" %
(time_train[1],time_predict_train[1],time_predict_test[1],f1_train[1],f1_test[1])

print("300      %.5f            %.5f                    %.5f                %.5
f               %.5f" %
(time_train[2],time_predict_train[2],time_predict_test[2],f1_train[2],f1_test[2])

print("====================================")
print("Comparison for Naive Bayes")
print("No.    Training time    Prediction time(train)    Prediction time(test)
  F1 Score(training)    F1 Score(testing)")
print("100      %.5f            %.5f                    %.5f                %.5
f               %.5f" %
(time_train2[0],time_predict_train2[0],time_predict_test2[0],f1_train2[0],f1_tes
t2[0]))
print("200      %.5f            %.5f                    %.5f                %.5
f               %.5f" %
(time_train2[1],time_predict_train2[1],time_predict_test2[1],f1_train2[1],f1_tes
t2[1]))
print("300      %.5f            %.5f                    %.5f                %.5
f               %.5f" %
(time_train2[2],time_predict_train2[2],time_predict_test2[2],f1_train2[2],f1_tes
t2[2]))
print("====================================")
print("Comparison for SVM")
print("No.    Training time    Prediction time(train)    Prediction time(test)
  F1 Score(training)    F1 Score(testing)")
print("100      %.5f            %.5f                    %.5f                %.5
f               %.5f" %
(time_train3[0],time_predict_train3[0],time_predict_test3[0],f1_train3[0],f1_tes
t3[0]))
print("200      %.5f            %.5f                    %.5f                %.5
f               %.5f" %
(time_train3[1],time_predict_train3[1],time_predict_test3[1],f1_train3[1],f1_tes
t3[1]))
print("300      %.5f            %.5f                    %.5f                %.5
f               %.5f" %
(time_train3[2],time_predict_train3[2],time_predict_test3[2],f1_train3[2],f1_tes
t3[2]))
#training time, prediction time, F1 score on training set and F1 score on test s
et, for each training set size.
#import pandas
#pandas.DataFrame(train_f1_score, train_f1_score, train_f1_score)
```

```
==============================
==============================
==============================
=========Naive Bayes==========
Training GaussianNB...
Done!
Training time (secs): 0.003
GaussianNB()
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.002
F1 score for training set: 0.80378250591
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.763358778626
------------------------------------------
Training set size return: 100
Done!
Prediction time (secs): 0.001
Done!
Prediction time (secs): 0.001
F1 score for training set return: 0.846715328467
F1 score for test set return: 0.802919708029
------------------------------------------
Training set size return: 200
Done!
Prediction time (secs): 0.000
Done!
Prediction time (secs): 0.000
F1 score for training set return: 0.840579710145
F1 score for test set return: 0.724409448819
------------------------------------------
Training set size return: 300
Done!
Prediction time (secs): 0.000
Done!
Prediction time (secs): 0.000
F1 score for training set return: 0.80378250591
F1 score for test set return: 0.763358778626
==============================
==============================
==============================
====Support Vector Machine====
Training SVC...
Done!
Training time (secs): 0.011
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rb
f',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
Predicting labels using SVC...
Done!
Prediction time (secs): 0.005
F1 score for training set: 0.876068376068
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.645161290323
------------------------------------------
```

```
Training set size return: 100
Done!
Prediction time (secs): 0.001
Done!
Prediction time (secs): 0.001
F1 score for training set return: 0.877697841727
F1 score for test set return: 0.774647887324
------------------------------------------
Training set size return: 200
Done!
Prediction time (secs): 0.002
Done!
Prediction time (secs): 0.001
F1 score for training set return: 0.867924528302
F1 score for test set return: 0.781456953642
------------------------------------------
Training set size return: 300
Done!
Prediction time (secs): 0.007
Done!
Prediction time (secs): 0.002
F1 score for training set return: 0.876068376068
F1 score for test set return: 0.783783783784
Table Comparison
Comparison for Decision Tree
No.    Training time    Prediction time(train)    Prediction time(te
st)    F1 Score(training)    F1 Score(testing)
100      0.00177          0.00033                      0.00018
           1.00000              0.61538
200      0.00134          0.00020                      0.00016
           1.00000              0.74242
300      0.00183          0.00022                      0.00018
           1.00000              0.64516
====================================
Comparison for Naive Bayes
No.    Training time    Prediction time(train)    Prediction time(te
st)    F1 Score(training)    F1 Score(testing)
100      0.00135          0.00063                      0.00081
           0.84672              0.80292
200      0.00159          0.00042                      0.00026
           0.84058              0.72441
300      0.00078          0.00037                      0.00025
           0.80378              0.76336
====================================
Comparison for SVM
No.    Training time    Prediction time(train)    Prediction time(te
st)    F1 Score(training)    F1 Score(testing)
100      0.00115          0.00074                      0.00086
           0.87770              0.77465
200      0.00314          0.00218                      0.00112
           0.86792              0.78146
300      0.00776          0.00733                      0.00172
           0.87607              0.78378
```

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? ###### It's seem that from the table above, Decision tree is the fastest in term of training and testing but F1 score is the lowest. I can conculde that Decision tree has high variance because while it's make perfect score in training but the f1 score in testing in the lowest. As lecturer's explained that Decision tree is not suitable for complex data. ###### Naive Bayes is the second in terms of fastest in traning and testing while f1 score for testing has done a similar result when compare to SVC. SVC is the best in prediction accuracy but it's take longest time to train. I will pick Naive Bayes for the most appropriate model if we're consider about limited resource, cost compare to f1 score.
- 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). ###### To explained in layman's terms with 1-2 paragraphs will be, We are using Naive Bayes method to be our appropriate model. How Naive Bayes work? We've devided into 2 parts Naive rules and Bayes rules. If you have 10 candies in total, 8 of that candies come from box A, 2 come from box B. If you random pick 1 out of 10, we can guess that a candy that you pick is come from Box A because it has higher probability. This is called Naive rules because we pick it naively. For Bayes rules, if you are interesting in study crime rate, and you could see the data that there is a connection or related to education of people around there, economic, and how crowded. which you can use Bayes Theorem for getting probability of those informations. ###### So Naive Bayes are looking into features or the data that you provided and calculate the relation between those featuers to get probability. They will use "Naive" to guess that if student is far from school they will have higher probability to failed at school and after that algorithm will use "Bayes" to see the connection of other featuers to calculate other "Naive" to get the final probability of the process.
- 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. ###### Naive Bayes doesn't need Fine-tune because they are't any parameters for Naive Bayes but I show how Fine-tune is working below by using Support Vector Machine
- What is the model's final F_1 score? ###### F1 score is 0.794520547945 which I could improved from 0.78378 and the speed from 0.00721 go down to 0.00665402412415

In [19]:

```
# TODO: Fine-tune your model and report the best F1 score
#due to GaussianNB are not accept any paramater so I use SVM instead to see how
 much it could improve
from sklearn.grid_search import GridSearchCV
from sklearn.cross_validation import KFold
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import make_scorer
from sklearn import svm

def fine_tune(classifier, params, X_train, y_train):
    kfcv = KFold(n=len(y_train), n_folds=10, shuffle=True)
    grid = GridSearchCV(classifier, params, cv=kfcv, scoring='f1')
    grid.fit(X_train, y_train)
    return grid.best_estimator_, grid.best_params_

y_new_all = y_all.replace(['yes', 'no'], [1, 0]) # grid search is error because
 can't accept string
X_train, X_test, y_train, y_test = train_test_split(X_all, y_new_all,
test_size=num_test, random_state=42)
params = {'degree':(1,2,3), 'kernel':('linear','poly','rbf', 'sigmoid'), 'C':(1.
0,3.0,5.0,10.0) }
svc = svm.SVC()
#NB = GaussianNB()
#cfs = DecisionTreeClassifier()
model, best_value = fine_tune(svc, params, X_train, y_train)

print "best choice : {}".format(best_value)
```

```
best choice : {'kernel': 'poly', 'C': 1.0, 'degree': 1}
```

In [20]:

```
start = time.time()
model.fit(X_train, y_train)
end = time.time()
f1 = f1_score(model.predict(X_test), y_test)
print "Number of Training : ", len(X_train)
print "Time : {}" .format(end - start)
print "F1 score : {}".format(f1)
```

```
Number of Training :  300
Time : 0.00693011283875
F1 score : 0.794520547945
```

In [ ]:

```

```

In [ ]:

```

```