experimenting2_intervention

April 3, 2016

```
In [76]: # Project 2: Supervised Learning
         ### Building a Student Intervention System
```

0.1 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?

```
In [77]: # Import libraries
         import numpy as np
         import pandas as pd
```

```
In [78]: # 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 [79]: # TODO: Compute desired values - replace each '?' with an appropriate expression/function call

         n_students = len(student_data.index)
         n_features = len(student_data.columns)
         n_passed = sum([1 for y in student_data['passed'] if y == 'yes'])
         n_failed = sum([1 for n in student_data['passed'] if n == 'no'])
         grad_rate = 100.*n_passed/(n_passed + n_failed)

         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%
```

0.2 3. Preparing the Data

In this section, we will prepare the data for modeling, training and testing.

0.2.1 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 [80]:
```
# 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', 'gua
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]
```

0.2.2 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()` function to perform this transformation.

```
In [81]: # 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', '

                 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'
```

0.2.3 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 [82]: # 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 d
         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 = .24, 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
```

0.3 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 F1 score. Repeat this process with different training set sizes (100, 200, 300), keeping test set constant.

Produce a table showing training time, prediction time, F1 score on training set and F1 score on test set, for each training set size.

Note: You need to produce 3 such tables - one for each model.

In [83]:
```
# Helper Functions
import time
from sklearn.metrics import f1_score

# Return the classifier's training time
def timeTraining(clf, X_train, y_train):
    start = time.time()
    clf.fit(X_train, y_train)
    end = time.time()
    return "{:.3f}".format(end - start)

# Return the classifier's predictions and prediction time
def predictAndTime(clf, features):
    start = time.time()
    y_pred = clf.predict(features)
    end = time.time()
    return y_pred, "{:.3f}".format(end - start)

# Return the f1 score for the target values and predictions
def F1(target, prediction):
    return f1_score(target.values, prediction, pos_label='yes')
```

In [84]:
```
# To get my bearings I wanted to try most of the classifiers seen in class out of the box.
# I excluded neural net since its recommended use requires the features to be scaled

from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier

# Array of classifiers
clfs = [DecisionTreeClassifier(criterion = "entropy"),
        SVC(C = 1.0, kernel="rbf"),
        GaussianNB(),
        AdaBoostClassifier(),
        KNeighborsClassifier(n_neighbors = 3)]

#Gathering Table column and index labels
classifier_names = [clf.__class__.__name__ for clf in clfs]
benchmarks = ["Training time",  "F1 score training set","Prediction time", "F1 score test set"
table = pd.DataFrame(columns = classifier_names, index = benchmarks)

# Fit Classifiers
for clf in clfs:
    classifier   = clf.__class__.__name__
```

```
        t_train    = timeTraining(clf, X_train, y_train)
        pred_train_set = predictAndTime(clf, X_train)[0]
        pred_test_set, t_test = predictAndTime(clf,X_test)

        table[classifier]['Training time']     = t_train
        table[classifier]['F1 score training set'] = F1(y_train, pred_train_set)
        table[classifier]['Prediction time']   = t_test
        table[classifier]['F1 score test set'] = F1(y_test, pred_test_set)
```

In [85]:
```
from IPython.display import display, HTML
display(table)
#HTML(table.to_html())
#Out of the box, with no tuning, it seems hard to differentiate their performance. Aside from
#slower training times seen in SVC and Adaboost, they pretty much all give similar f1 test sco
#SVC and Adaboost slightly ahead of the rest.
```

```
                      DecisionTreeClassifier       SVC GaussianNB  \
Training time                          0.006     0.016      0.002
F1 score training set                      1  0.869198   0.808824
Prediction time                        0.000     0.004      0.001
F1 score test set                   0.757576  0.758621       0.75

                      AdaBoostClassifier KNeighborsClassifier
Training time                      0.136                0.001
F1 score training set           0.868778             0.886878
Prediction time                    0.006                0.002
F1 score test set               0.779412             0.721805
```

In [86]:
```
# Helper function makeTable
def makeTable(clf, training_sizes, X_tr, X_t, y_tr, y_t):

    #Gathering column and row labels for the table
    benchmarks = ["Training time",  "F1 score training set","Prediction time", "F1 score test s
    size_labels = ["Training samples: {}".format(s) for s in training_sizes]
    table = pd.DataFrame(columns = benchmarks, index = size_labels)

    for i, size in enumerate(training_sizes):
        #Use only the first "size" number of samples
        X_train, X_test, y_train, y_test = [df.iloc[:size] for df in [X_tr, X_t, y_tr, y_t]]

        #Compute benchmarks
        t_train    = timeTraining(clf, X_train, y_train)
        pred_train_set = predictAndTime(clf, X_train)[0]
        pred_test_set, t_test = predictAndTime(clf,X_test)

        #fill table
        table['Training time'][i]     = t_train
        table['F1 score training set'][i] = F1(y_train, pred_train_set)
        table['Prediction time'][i]   = t_test
        table['F1 score test set'][i] = F1(y_test, pred_test_set)

    return table
```

In [87]:
```
# Test Classifiers with increasing data set size
training_sizes = [50,100,150,200,250,300]
```

```
for clf in clfs:
    print clf.__class__.__name__
    table = makeTable(clf, training_sizes, X_train, X_test, y_train, y_test)
    display(table)
```

```
DecisionTreeClassifier

                         Training time F1 score training set Prediction time  \
Training samples: 50             0.002                     1           0.001
Training samples: 100            0.003                     1           0.000
Training samples: 150            0.004                     1           0.000
Training samples: 200            0.004                     1           0.000
Training samples: 250            0.005                     1           0.000
Training samples: 300            0.005                     1           0.000

                         F1 score test set
Training samples: 50              0.677966
Training samples: 100             0.694915
Training samples: 150             0.703125
Training samples: 200             0.753846
Training samples: 250             0.710744
Training samples: 300             0.772727


SVC

                         Training time F1 score training set Prediction time  \
Training samples: 50             0.001               0.90625           0.001
Training samples: 100            0.003               0.85906           0.002
Training samples: 150            0.003              0.870813           0.002
Training samples: 200            0.004              0.869281           0.002
Training samples: 250            0.006              0.879177           0.002
Training samples: 300            0.008              0.869198           0.002

                         F1 score test set
Training samples: 50              0.738462
Training samples: 100             0.783784
Training samples: 150             0.771429
Training samples: 200              0.77551
Training samples: 250             0.758621
Training samples: 300             0.758621


GaussianNB

                         Training time F1 score training set Prediction time  \
Training samples: 50             0.001              0.666667           0.000
Training samples: 100            0.001              0.854962           0.000
Training samples: 150            0.001              0.808743           0.000
Training samples: 200            0.001              0.832061           0.001
Training samples: 250            0.001              0.817647           0.000
Training samples: 300            0.001              0.808824           0.000

                         F1 score test set
Training samples: 50              0.468085
```

```
Training samples: 100         0.748092
Training samples: 150         0.736842
Training samples: 200         0.713178
Training samples: 250         0.746269
Training samples: 300             0.75


AdaBoostClassifier

                      Training time F1 score training set Prediction time  \
Training samples: 50          0.099                      1           0.005
Training samples: 100         0.093               0.953846           0.006
Training samples: 150         0.097               0.912821           0.006
Training samples: 200         0.117               0.882562           0.006
Training samples: 250         0.153               0.886427           0.007
Training samples: 300         0.158               0.868778           0.008

                      F1 score test set
Training samples: 50           0.645161
Training samples: 100              0.72
Training samples: 150          0.757576
Training samples: 200          0.805755
Training samples: 250          0.776978
Training samples: 300          0.779412


KNeighborsClassifier

                      Training time F1 score training set Prediction time  \
Training samples: 50          0.001                    0.8           0.001
Training samples: 100         0.001               0.823529           0.002
Training samples: 150         0.001               0.816327           0.002
Training samples: 200         0.001                0.86121           0.003
Training samples: 250         0.001               0.889503           0.003
Training samples: 300         0.001               0.886878           0.003

                      F1 score test set
Training samples: 50           0.761905
Training samples: 100          0.666667
Training samples: 150          0.677419
Training samples: 200          0.666667
Training samples: 250          0.711111
Training samples: 300          0.721805
```

```
In [88]: # 3 chosen classifiers: DecissionTree, SVC, and AdaBoost

         # The models were relatively stable as training size increased. KNN had a strange u-shaped beh
         # highish score decreasing, and then rising again. The SVC seemed the most stable acrosss diff
         # training data set sizes.

         chosen_clfs = [ KNeighborsClassifier()]
```

```
In [89]: # tree.export_graphviz(clfs[0].fit(X_train,y_train), out_file='tree.dot')
```

0.4 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 [90]: # TODO: Fine-tune your model and report the best F1 score
         from sklearn import grid_search
         from sklearn.metrics import make_scorer

         r = np.arange
         scorer = make_scorer(F1)

         neigh_param = {'n_neighbors' : [10,20,25,30,40], 'weights' : ['uniform', 'distance'], 'p':[1,2


         #Perform grid Search
         def gridIt(clf, params):
             grid_clf = grid_search.GridSearchCV(clf, params, scorer)
             final_clf = grid_clf.fit(X_train, y_train).best_estimator_
             print final_clf
             y_pred, predict_t = predictAndTime(final_clf, X_test)
             print F1(y_test, y_pred), predict_t
             print '------------------\n'

         gridIt(chosen_clfs[0], neigh_param)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=30, p=2,
           weights='uniform')
0.77027027027 0.003
------------------
```

```
In [ ]:
```