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/fu
nction call
nO, nF = student_data.shape
col_nm = student_data.columns.tolist()
pss = student_data[col_nm[-1]]

isPass = np.array(pss=='yes')
nPass = sum(isPass==True)
nFail = sum(isPass==False)
grad = float(nPass) / float(nO)

n_students = nO
n_features = nF
n_passed = nPass
n_failed = nFail
grad_rate = grad*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: 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 [3]:
```
# 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_org = student_data[feature_cols]  # feature values for all students
y_org = student_data[target_col]  # corresponding targets/labels
print "\nFeature values:-"
print X_org.head()  # print the first 5 rows
```

```
Feature column(s):-
['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu', 'Mjob'
, 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime', 'failures', 'schoolsu
p', 'famsup', 'paid', 'activities', 'nursery', '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   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 [4]:
```
# 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' => 'sc
hool_GP', 'school_MS'

        outX = outX.join(col_data)  # collect column(s) in output dataframe

    return outX

X_org_aug = preprocess_features(X_org)
print "Processed feature columns ({}):-\n{}".format(len(X_org_aug.columns), list(X_
org_aug.columns))
```

```
Processed feature columns (48):-
['school_GP', 'school_MS', 'sex_F', 'sex_M', 'age', 'address_R', 'address_U', 'f
amsize_GT3', 'famsize_LE3', 'Pstatus_A', 'Pstatus_T', 'Medu', 'Fedu', 'Mjob_at_h
ome', 'Mjob_health', 'Mjob_other', 'Mjob_services', 'Mjob_teacher', 'Fjob_at_hom
e', 'Fjob_health', 'Fjob_other', 'Fjob_services', 'Fjob_teacher', 'reason_course
', 'reason_home', 'reason_other', 'reason_reputation', 'guardian_father', 'guard
ian_mother', 'guardian_other', 'traveltime', 'studytime', 'failures', 'schoolsup
', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet', 'romantic',
'famrel', '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 [5]:
```
# 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 orderi
ng in the dataset
from sklearn.utils import shuffle
ixs = shuffle(X_org_aug.index, random_state=0)
X_tmp_aug, y_tmp = X_org_aug.reindex(ixs), y_org.reindex(ixs)

X_train, y_train = X_tmp_aug.ix[ixs[:num_train],:], y_tmp[ixs[:num_train]]
X_test, y_test = X_tmp_aug.ix[ixs[num_train:],:], y_tmp[ixs[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 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.

Overall review of 3 supervised learning algos

Given that the problem is one of classification (digital outcome) I will choose 3 algorithms that are more tuned to that: SVM, k-NN, and decision tree.

O(n)

As far as I gather: SVM is the most challenging being of O(n) ~ d n^2 *to d* n^3

k-NN is O(n) ~ d *n* log(n) or d * n^2

Decition tree can be reduced to log(n)

Strengths

SVM: high dim problems, uses a subset of data thus robust, versatile through choices of kernel functions

k-NN: non-parametric uses a subset of data thus more robust, good for complex decision boundaries

Decision-tree: trees can be visualized, robust

Weaknesses

SVM: can be challenging if features larger than # of samples (This should be a bad state of affairs in general, because the problem in under-determined: one should be able to drive the training error down to zero, if the problem has a well-defined solution)

k-NN: will not generalize globally

Decision tree: prone to overfitting; certain problems (XOR) are hard for trees

Justifying choices

The problem is certainly complex: we turned it into 48 features with a sample of size of 300. It is unlikely that the data samples the space properly. So local approximators like k-NN are probably good. SVM should be able to hand complex boundaries and decision tree can be visualized so that it might help checking our intuition (although I was not able to use pydot to do that).

I left out Naive Bayes, in part because of the conditional independence assumption. Note that the preprocess method will convert {'yes','no'} features into {0,1}, however, other variables that assume two values, e.g. sex = {'M','F'} are converted into 2 features, which are actually exactly 100% correlated (one is the opposite of the other). Also other variables that contain 3 values are dependent (if you have a feature 'a' \in {a_1, a_2, a_3} and know the value of a_1 (= 0 or 1), a_2 (= 0 or 1), you'll know the value of a_3).

In [6]:
```
# 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 import svm
clf = svm.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
```

```
Training SVC...
Done!
Training time (secs): 0.021
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 [7]:
```
# 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__)
    t_start = time.time()
    y_pred = clf.predict(features)
    t_end = time.time()
    dt = t_end - t_start
    print "Done!\nPrediction time (secs): {:.3f}".format(dt)
    return (f1_score(target.values, y_pred, pos_label='yes'), dt)

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

```
Predicting labels using SVC...
Done!
Prediction time (secs): 0.016
F1 score for training set: 0.859688195991
```

In [8]:
```
# Predict on test data
f1_tmp, dt_tmp = predict_labels(clf, X_test, y_test)
print "F1 score for test set: {}".format(f1_tmp)
```

```
Predicting labels using SVC...
Done!
Prediction time (secs): 0.006
F1 score for test set: 0.853503184713
```

In [9]:
```
# 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))
    dt_train = train_classifier(clf, X_train, y_train)
    f1_train, dt_train_prd = predict_labels(clf, X_train, y_train)
    f1_test, dt_test_prd = predict_labels(clf, X_test, y_test)
    print "Training time (secs): {:.3f}".format(dt_train)
    print "F1 score for training set: {}".format(f1_train)
    print "F1 score for test set: {}".format(f1_test)
    return (f1_train, f1_test, dt_train, dt_test_prd)

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

X1, y1 = X_train.ix[ixs[0:100],:], y_train[ixs[0:100]]
X2, y2 = X_train.ix[ixs[0:200],:], y_train[ixs[0:200]]
X3, y3 = X_train, y_train # memory inefficient, but improving legibility
f11_train, f11_test, dt1_train, dt1_test = train_predict(clf, X1, y1, X_test, y_tes
t)
f12_train, f12_test, dt2_train, dt2_test = train_predict(clf, X2, y2, X_test, y_tes
t)
f13_train, f13_test, dt3_train, dt3_test = train_predict(clf, X3, y3, X_test, y_tes
t)

from pandas import DataFrame
df_SVM = DataFrame(columns={'train', 'test', 'train time', 'test time'})
df_SVM['train'] = [f11_train, f12_train, f13_train]
df_SVM['test'] = [f11_test, f12_test, f13_test]
df_SVM['train time'] = [dt1_train, dt2_train, dt3_train]
df_SVM['test time'] = [dt1_test, dt2_test, dt3_test]
df_SVM = df_SVM[['train', 'test', 'train time', 'test time']]


from IPython.display import display, HTML
print "SVM"
display(df_SVM)
```

```
------------------------------------------
Training set size: 100
Training SVC...
Done!
Training time (secs): 0.005
Predicting labels using SVC...
Done!
Prediction time (secs): 0.009
Predicting labels using SVC...
Done!
Prediction time (secs): 0.003
Training time (secs): 0.005
F1 score for training set: 0.857142857143
F1 score for test set: 0.857142857143
------------------------------------------
Training set size: 200
Training SVC...
Done!
Training time (secs): 0.012
Predicting labels using SVC...
Done!
Prediction time (secs): 0.010
Predicting labels using SVC...
Done!
Prediction time (secs): 0.004
Training time (secs): 0.012
F1 score for training set: 0.851211072664
F1 score for test set: 0.842767295597
------------------------------------------
Training set size: 300
Training SVC...
Done!
Training time (secs): 0.025
Predicting labels using SVC...
Done!
Prediction time (secs): 0.017
Predicting labels using SVC...
Done!
Prediction time (secs): 0.005
Training time (secs): 0.025
F1 score for training set: 0.859688195991
F1 score for test set: 0.853503184713
SVM
```

	train	test	train time	test time
0	0.857143	0.857143	0.005	0.003
1	0.851211	0.842767	0.012	0.004
2	0.859688	0.853503	0.025	0.005

In [67]:
```
# TODO: Train and predict using two other models
# Nearest Neighbor
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3, algorithm='ball_tree')

f11_train, f11_test, dt1_train, dt1_test = train_predict(clf, X1, y1, X_test, y_tes
t)
f12_train, f12_test, dt2_train, dt2_test = train_predict(clf, X2, y2, X_test, y_tes
t)
f13_train, f13_test, dt3_train, dt3_test = train_predict(clf, X3, y3, X_test, y_tes
t)

df_kNN = DataFrame(columns={'train', 'test', 'train time', 'test time'})
df_kNN['train'] = [f11_train, f12_train, f13_train]
df_kNN['test'] = [f11_test, f12_test, f13_test]
df_kNN['train time'] = [dt1_train, dt2_train, dt3_train]
df_kNN['test time'] = [dt1_test, dt2_test, dt3_test]
df_kNN = df_kNN[['train', 'test', 'train time', 'test time']]



# Decision tree
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(max_depth=5)

f11_train, f11_test, dt1_train, dt1_test = train_predict(clf, X1, y1, X_test, y_tes
t)
f12_train, f12_test, dt2_train, dt2_test = train_predict(clf, X2, y2, X_test, y_tes
t)
f13_train, f13_test, dt3_train, dt3_test = train_predict(clf, X3, y3, X_test, y_tes
t)

df_DT = DataFrame(columns={'train', 'test', 'train time', 'test time'})
df_DT['train'] = [f11_train, f12_train, f13_train]
df_DT['test'] = [f11_test, f12_test, f13_test]
df_DT['train time'] = [dt1_train, dt2_train, dt3_train]
df_DT['test time'] = [dt1_test, dt2_test, dt3_test]
df_DT = df_DT[['train', 'test', 'train time', 'test time']]

print "\n\n"

print "SVM"
display(df_SVM)
print "\n"

print "kNN"
display(df_kNN)
print "\n"

print "DT"
display(df_DT)
print "\n"
```

```
------------------------------------------
Training set size: 100
Training KNeighborsClassifier...
Done!
Training time (secs): 0.000
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.015
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.000
Training time (secs): 0.000
F1 score for training set: 0.885496183206
F1 score for test set: 0.75
------------------------------------------
Training set size: 200
Training KNeighborsClassifier...
Done!
Training time (secs): 0.016
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.000
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.000
Training time (secs): 0.016
F1 score for training set: 0.862595419847
F1 score for test set: 0.785714285714
------------------------------------------
Training set size: 300
Training KNeighborsClassifier...
Done!
Training time (secs): 0.000
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.015
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.000
Training time (secs): 0.000
F1 score for training set: 0.883054892601
F1 score for test set: 0.777777777778
------------------------------------------
Training set size: 100
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.000
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
Training time (secs): 0.000
F1 score for training set: 0.921875
F1 score for test set: 0.732824427481
------------------------------------------
Training set size: 200
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.000
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
```

	train	test	train time	test time
0	0.930233	0.776978	0.015	0.016
1	0.902724	0.825175	0.000	0.000
2	0.886199	0.813793	0.000	0.000

kNN

	train	test	train time	test time
0	0.885496	0.750000	0.000	0
1	0.862595	0.785714	0.016	0
2	0.883055	0.777778	0.000	0

DT

	train	test	train time	test time
0	0.921875	0.732824	0.000	0
1	0.902724	0.825175	0.000	0
2	0.886199	0.813793	0.016	0

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?

Choice of algorithm

I chose to use SVM because it is adaptable and well suited for classification problems (e.g. yes/no answers) and complex decisions, i.e. when there are many variables (features) that can affect the answer. In addition it provides flexibility in terms of controlling overfitting (more technically: the balance of high bias vs. high variance) and is versatile in that it allows different ways to express similarity (more technically the kernel function). It is a computationally more involved algorithm but for the limited size of this problem it is not a relevant constraint. Training and testing is achieved in well below 1 second.

As a more technical caveat, and not for the board, it might be challenging to pick a model without making broader checks. I mostly used default parameters to train the different models. There is no guarantee that the default parameter values (or any specific set of parameters) will make the algorithms directly comparable. For example, if the classifiers were regressors whereby one had polynomial functions, another exponential functions, and the third trigonometric functions, by fixing the number of degrees of freedom in each to be the same we could make them directly comparable. It does not seem to be the case in this context where algorithms are sunstantially different and there might not be an obvious intrinsic measure of 'complexity' to make them compete on an even field.

SVM

As other machine learning algorithms SVM works be learning the relationship between inputs, gender, family size, travel time, etc. to the output: graduation. SVM solves the problem by trying to create regions of input features that will 'map' into graduation or not. Once these regions have been learned one can query the ML classifier and ask the question: if I know all this attributes for this student will it graduate? The algorithm will locate the point in this multi-dimensional space that corresponds to the candidate. If that point belongs to a region of 'will graduate' then it will classify the student as such.

I fine-tuned the SVM by optimizing over the parameter C with 3-fold CV. The optimal value is 1.47. To remain a bit truer to the spirit of out-of-sample testing I still optimized on just the training set and obtained a final F1 score of 0.85. Using the full data set the F1 score goes to 0.87.

In [13]:
```
# TODO: Fine-tune your model and report the best F1 score

# Gridsearch
from sklearn.metrics import make_scorer, f1_score
sc_fnc = make_scorer(f1_score, greater_is_better=True, pos_label='yes')
prm = {'C':np.linspace(0.2,4,10)}

from sklearn.grid_search import GridSearchCV
clf = svm.SVC()
clf_gs = GridSearchCV(clf,prm,scoring=sc_fnc,cv=3)
clf_gs.fit(X_train,y_train)
clf_gs.best_estimator_
clf_gs.predict(X_test)

f1_no_cheat = f1_score(y_test.values, clf_gs.predict(X_test), pos_label='yes')
print "f1 with optimized parameter (out-of-sample) = ", f1_no_cheat

# Entire data set
clf_gs = GridSearchCV(clf,prm,scoring=sc_fnc,cv=3)
clf_gs.fit(X_org_aug,y_org)
clf_gs.best_estimator_
clf_gs.predict(X_test)

f1_cheat = f1_score(y_test.values, clf_gs.predict(X_test), pos_label='yes')
print "f1 with optimized parameter (in-sample) = ", f1_cheat
```

```
f1 with optimized parameter (out-of-sample) =  0.851612903226
f1 with optimized parameter (in-sample) =  0.869565217391
```

In [ ]: