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?

This is classification problem. Here we need to divide our data in pass and failure. Regression is suitable for continuous output type data as dependant variable like score in exams. But here we have outcome as categorical variable i.e pass and failure.

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 feat
ure columns
```

```
Student data read successfully!
```

In [3]:
```
#temporary - to delete
student_data.columns
```

Out[3]:
```
Index([u'school', u'sex', u'age', u'address', u'famsize', u'Pstatus',
u'Medu',
       u'Fedu', u'Mjob', u'Fjob', u'reason', u'guardian', u'traveltim
e',
       u'studytime', u'failures', u'schoolsup', u'famsup', u'paid',
       u'activities', u'nursery', u'higher', u'internet', u'romantic',
       u'famrel', u'freetime', u'goout', u'Dalc', u'Walc', u'health',
       u'absences', u'passed'],
      dtype='object')
```

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 [4]:
```
# TODO: Compute desired values - replace each '?' with an appropriate ex
pression/function call
n_students = len(student_data.index)
n_features = len(student_data.columns)
n_passed = len(student_data[student_data['passed'] == 'yes'])
n_failed = len(student_data[student_data['passed'] == 'no'])
grad_rate = (n_passed*100)/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: 67.00%
```

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 [5]:
```
# 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
```

```
Feature column(s):-
['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fed
u', 'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime', 'f
ailures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'high
er', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Wa
lc', 'health', 'absences']
Target column: passed

Feature values:-
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjo
b  \
0     GP   F   18       U     GT3       A     4     4  at_home   teache
r
1     GP   F   17       U     GT3       T     1     1  at_home     othe
r
2     GP   F   15       U     LE3       T     1     1  at_home     othe
r
3     GP   F   15       U     GT3       T     4     2   health  service
s
4     GP   F   16       U     GT3       T     3     3    other     othe
r

    ...    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 [6]:
```
# 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', 'addre
ss_U', 'famsize_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_serv
ices', 'Fjob_teacher', 'reason_course', 'reason_home', 'reason_other',
'reason_reputation', 'guardian_father', 'guardian_mother', 'guardian_ot
her', 'traveltime', 'studytime', 'failures', 'schoolsup', 'famsup', 'pa
id', 'activities', 'nursery', 'higher', 'internet', 'romantic', 'famre
l', '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 [7]:
```
# First, decide how many training vs test samples you want
from sklearn import cross_validation
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
X_train, X_test, y_train, y_test = cross_validation.train_test_split(X_a
ll, y_all, train_size=num_train)
print "Training set: {} samples".format(X_train.shape[0])
print "Test set: {} samples".format(X_test.shape[0])
```

```
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 [8]:
```
import time
# Trains the classifier model and returns training 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()
    return (end-start)
#    print "Done!\nTraining time (secs): {:.3f}".format(end - start)

# Predict on training set and compute F1 score
# returns [prediction_time, 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 [end-start, f1_score(target.values, y_pred, pos_label='yes')]

# splits the data in training and testing samples. Trains and predicts o
n training/testing data respectively
# returns [training_time, prediction_time, f1_score_training, f1_score_t
esting_sample]
def splitTrainAndValidatePredictor(clf, _train_size):
    # As test size needs to be kept same and only train size needs to va
ried. X_train is already randomly
    # distributed so continuous data around it can be taken
    _X_train = X_train[:_train_size]
    _y_train = y_train[:_train_size]
    train_time = train_classifier(clf, _X_train, _y_train)
    [time_training_predict, f1_train] =  predict_labels(clf, _X_train, _
y_train)
    [time_testing_predict, f1_test] =  predict_labels(clf, X_test, y_tes
t)
    return [train_time, time_testing_predict, f1_train, f1_test]

# wrapper to iterate on different sample sizes
# return dataframe with data on performance/scores corresponding to algo
rithm passed.
def createLearningTable(clf):
    sample_sizes = [50, 100, 150, 200, 250, 300]
    rows = []
    for s in sample_sizes:
        rows.append([s] + splitTrainAndValidatePredictor(clf, s))
    df = pd.DataFrame(rows, columns = ['size', 'train_time', 'prediction
_time', 'train_score', 'test_score'])
    return df
```

Now we would try above learning framework on different types of algorithms.

DECISION TREE

Decision Trees(DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.
Advantages of Decision Tree are:-

- **Simple** to understand and interpret. Trees can be visualized.
- **Cost** of using the tree is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data.
- Able to handle multi-output problems.

Disadvantages of decision trees include:

- **Overfitting** Decision Tree learners can create over-complex trees that do not generalise the data well.
- **NP-complete** Problem of learning an optimal decision tree is NP-complete. so most algorithms are based on heuristic like greedy algorithm.

Reasons for choosing decision tree for this student intervention problem are-

- As Decison Tree is available as a module in sklearn, it could be quickly used.
- Advantages of decision tree mentioned above satisfy our requirements. As our resources are limited, so we want efficient algorithm. Also our output data is categorical ('pass' or 'failed').
- As node of decision tree splits the data based on value of particular attribute, it helps building intuition to intervene in a particular feature for better success rate.

In [9]:
```
from sklearn import tree
clf = tree.DecisionTreeClassifier()
df_tree = createLearningTable(clf)
df_tree
```

Out[9]:

	size	train_time	prediction_time	train_score	test_score
0	50	0.016	0	1	0.783217
1	100	0.000	0	1	0.672131
2	150	0.015	0	1	0.776119
3	200	0.000	0	1	0.683333
4	250	0.000	0	1	0.730159
5	300	0.000	0	1	0.772727

SUPPORT VECTOR MACHINE

Support Vector Machine (SVM) model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that sampe space and predicted to belong to category based on which side of the gap they fall on.

Advantages are

- Effective in **high dimensional** spaces
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function(called support vectors), so that it is also **memory efficient**.
- **versatile**: different kernel functions can be specified for the decision function.

Disadvantages are

- If the number of features is much greater than the number of samples, the method is likely to give poor performances.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross validation.

We chose SVM to use for this student intervention problem as:

- SVM support is available in sklearn, so it is easier to use.
- our number of features are large, which is handled by SVM better.
- Our problems is classification, which is optimized and solved by SVM algorithm.
- We would have number of different parameters, kernels to finetune our model.

In [10]:
```
from sklearn import svm
clf = svm.SVC()
df_svm = createLearningTable(clf)
df_svm
```

Out[10]:

	size	train_time	prediction_time	train_score	test_score
0	50	0.047	0.000	0.921053	0.845638
1	100	0.000	0.000	0.875000	0.861111
2	150	0.000	0.000	0.859574	0.842767
3	200	0.016	0.000	0.836013	0.851613
4	250	0.015	0.000	0.837696	0.849673
5	300	0.015	0.016	0.862832	0.842105

LOGISTIC REGRESSION

Logistic regression is a linear model for classification rather than regression. Advantages of Logistic Regression are:

- Parametric analytic tool
- Provides probability outcome
- Controls confounding

Disadvantages of Logistic regression are:

- suffers from overfitting.
- it is not robust to outliers

We chose logistic regression as:-

- It is available in sklearn, so would be easier to use.
- Logistic regression is binary classification algorithm suitable for our purpose here.

In [11]:
```
from sklearn import linear_model
clf = linear_model.LogisticRegression()
df_logistic = createLearningTable(clf)
df_logistic
```

Out[11]:

	size	train_time	prediction_time	train_score	test_score
0	50	0.160	0.031	0.972222	0.787879
1	100	0.000	0.000	0.848485	0.775194
2	150	0.000	0.000	0.859813	0.770370
3	200	0.000	0.000	0.850000	0.764706
4	250	0.015	0.000	0.839080	0.776119
5	300	0.016	0.000	0.838863	0.797101

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?

We have tried following models on our data and calculated both computational and learning perfomance on training/testing data.

- Decision Tree
- SVM
- Logistic Regression

We found that Decision Tree perfectly fits the data as its f1 score on training set is perfect 1.0, which seems to be overfit as checked by score on testing set. We found that training and testing time of Decision Tree algorithm is very low when compared to other algorithms like SVM, logistic regression. Although SVM has best generalized the model and has highest testing score, but its performance cost in terms of runtime is not as efficient as compared to decision tree. Logistic regression has number of inherent limitations. Taking account of all these factors, we have chosed Decision Tree model to make prediction. Though Decision Tree is overfitting in current model, but we could fine-tune it by limiting depth of tree to generalize the model.

Decision Tree splits the the data hierarchically based on range of feature attributes. It create a tree like graph of decisions and their possible consequences. All the terminating nodes (i.e. without children) are final outcomes. While creating a model, decision tree algorithm at each node tries to split the data in way that best separates negative and positive outcomes.
After decision tree is created, to get category of a variable, it is just traversed through tree starting from root to bottom and choosing a children at each node as directed by node per its feature value.

In [12]:
```
from sklearn import grid_search
from sklearn.tree import DecisionTreeRegressor
regressor = tree.DecisionTreeClassifier()
parameters = {'max_depth':(1,2,3,4,5,6,7,8,9,10)}
clf = grid_search.GridSearchCV(regressor, parameters)
clf.fit(X_train, y_train)
print "F1 score for train set with fine tuned Decision Tree : {}".format
(predict_labels(clf.best_estimator_, X_train, y_train)[1])
print "F1 score for test set with fine tuned Decision Tree: {}".format(p
redict_labels(clf.best_estimator_, X_test, y_test)[1])
```

```
F1 score for train set with fine tuned Decision Tree : 0.806378132118
F1 score for test set with fine tuned Decision Tree: 0.826086956522
```