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: We want to know whether or not a student is likely to pass or fail. This would be classified as a discrete problem and therefore a classification problem.

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
from tabulate import tabulate

import warnings
warnings.filterwarnings('ignore')
```

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/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 = n_passed / float(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*100))
```

```
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 [4]:

```
# 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', 'reas
on', 'guardian', 'traveltime', 'studytime', 'failures', 'schoolsup', 'famsup', 'paid', 'activit
ies', '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)](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html?highlight=get_dummies#pandas.get_dummies) function to perform this transformation.

In [5]:

```
# 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', 'famsize_GT3', 'f
amsize_LE3', 'Pstatus_A', 'Pstatus_T', 'Medu', 'Fedu', 'Mjob_at_home', 'Mjob_health', 'Mjob_oth
er', 'Mjob_services', 'Mjob_teacher', 'Fjob_at_home', 'Fjob_health', 'Fjob_other', 'Fjob_servic
es', 'Fjob_teacher', 'reason_course', 'reason_home', 'reason_other', 'reason_reputation', 'guar
dian_father', 'guardian_mother', 'guardian_other', 'traveltime', 'studytime', 'failures', 'scho
olsup', '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 [6]:

```
# 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 import cross_validation

X_train, X_test, y_train, y_test = cross_validation.train_test_split(X_all, y_all,
                                                                     train_size=num_train, random_state=100)


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.

Answer: The three supervised learing models I used were the Gaussian Naive Bayes, the Support Vector Machine, and the random forest model.

Gaussian Naive Bayes: This model has a space complexity of O(dc) and a training time complexity of O(nd+cd). The general application of this model includes spam detection, real time prediction, and text classification. Naive Bayes strengths would include that the learning speed is fast and when assumption of independence holds, a Naive Bayes classifier performs better compared to other models like logistic regression and you need less training data. A weakness would be If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as “Zero Frequency”. I chose this model because of its speed.

Support Vector Machine: This model has a space complexity between O(n^2) and O(n^3). This algorithm can be used for text and hypertext catergorization, classification of images, and bioinformatics classification. Some strengths would be that it is effective in cases where number of dimensions is greater than the number of samples and ot works really well with clear margin of separation. A weakness could be that it doesn’t perform very well, when the data set has more noise i.e. target classes are overlapping and It doesn’t perform well, when we have large data. I chose this algorithm because the use of the kernel trick, which transform the data and then based on these transformations it finds an optimal boundary between the possible outputs

Random Forest: A Random Forest has a space complexity of O(√f N log N) and a training complexity of O(M√f N log N). Random Forest can be used for data mining. One of it's strengths is that it can handle large data sets with high dimensionality. It also has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing. A flaw could be that it doesnt do to well dealing with regression problems. It doesn’t predict beyond the range in the training data, and that they may over-fit data sets that are particularly noisy. I chose this algorithm because it grows multiple decision trees that vote and chooses the tree with the most votes.

In [7]:

```
# 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
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()

# 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 GaussianNB...
Done!
Training time (secs): 0.001
GaussianNB()
```

In [8]:

```
# 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')

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

```
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.7855421686746987
```

In [9]:

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

```
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.7910447761194029
```

In [10]:

```
# 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)))

# TODO: Run the helper function above for desired subsets of training data
# Note: Keep the test set constant
train_predict(clf, X_train[:100], y_train[:100], X_test, y_test)
train_predict(clf, X_train[:200], y_train[:200], X_test, y_test)
train_predict(clf, X_train, y_train, X_test, y_test)
```

```
------------------------------------------
Training set size: 100
Training GaussianNB...
Done!
Training time (secs): 0.001
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.026
F1 score for training set: 0.49411764705882355
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.40860215053763443
------------------------------------------
Training set size: 200
Training GaussianNB...
Done!
Training time (secs): 0.001
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.8106060606060607
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.75
------------------------------------------
Training set size: 300
Training GaussianNB...
Done!
Training time (secs): 0.001
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.7855421686746987
Predicting labels using GaussianNB...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.7910447761194029
```

In [11]:

```
header = ['Training Set Size', '100', '200', '300']
table = [['Training time (secs)', '0.001', '0.001', '0.001'],['Prediction time (secs)','0.000','0.000','0.00
0'],
        ['F1 score for training set','0.494','0.810','0.785'],['F1 score for test set','0.408','0.750','0.79
1']]

print(tabulate(table, header, tablefmt='fancy_grid'))
```

Training Set Size	100	200	300
Training time (secs)	0.001	0.001	0.001
Prediction time (secs)	0	0	0
F1 score for training set	0.494	0.81	0.785
F1 score for test set	0.408	0.75	0.791

In [12]:

```
# TODO: Train and predict using two other models
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier


svc = svm.SVC()

train_predict(svc, X_train[:100], y_train[:100], X_test, y_test)
train_predict(svc, X_train[:200], y_train[:200], X_test, y_test)
train_predict(svc, X_train, y_train, X_test, y_test)

header = ['Training Set Size', '100', '200', '300']
table1 = [['Training time (secs)', '0.001', '0.003', '0.006'],['Prediction time (secs)','0.001','0.003','0.0
05'],
        ['F1 score for training set','0.882','0.881','0.845'],['F1 score for test set','0.774','0.783','0.83
3']]

print(tabulate(table1, header, tablefmt='fancy_grid'))
```

```
------------------------------------------
Training set size: 100
Training SVC...
Done!
Training time (secs): 0.001
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.8823529411764706
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.7746478873239437
------------------------------------------
Training set size: 200
Training SVC...
Done!
Training time (secs): 0.003
Predicting labels using SVC...
Done!
Prediction time (secs): 0.002
F1 score for training set: 0.8813559322033899
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.7837837837837838
------------------------------------------
Training set size: 300
Training SVC...
Done!
Training time (secs): 0.006
Predicting labels using SVC...
Done!
Prediction time (secs): 0.005
F1 score for training set: 0.8459869848156182
Predicting labels using SVC...
Done!
Prediction time (secs): 0.002
F1 score for test set: 0.8333333333333334
```

Training Set Size	100	200	300
Training time (secs)	0.001	0.003	0.006
Prediction time (secs)	0.001	0.003	0.005
F1 score for training set	0.882	0.881	0.845
F1 score for test set	0.774	0.783	0.833

In [ ]:

```
rfc = RandomForestClassifier(n_estimators=25)

train_predict(rfc, X_train[:100], y_train[:100], X_test, y_test)
train_predict(rfc, X_train[:200], y_train[:200], X_test, y_test)
train_predict(rfc, X_train, y_train, X_test, y_test)

header = ['Training Set Size', '100', '200', '300']
table1 = [['Training time (secs)', '0.015', '0.021', '0.022'],['Prediction time (secs)','0.001','0.002','0.0
02'],
        ['F1 score for training set','1.0','1.0','1.0'],['F1 score for test set','0.755','0.720','0.829']]

print(tabulate(table1, header, tablefmt='fancy_grid'))
```

```
------------------------------------------
Training set size: 100
Training RandomForestClassifier...
Done!
Training time (secs): 0.020
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.002
F1 score for training set: 1.0
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.7819548872180452
------------------------------------------
Training set size: 200
Training RandomForestClassifier...
Done!
Training time (secs): 0.018
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.002
F1 score for training set: 1.0
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.002
F1 score for test set: 0.786206896551724
------------------------------------------
Training set size: 300
Training RandomForestClassifier...
Done!
Training time (secs): 0.017
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.002
F1 score for training set: 0.9974554707379135
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.7916666666666667
```

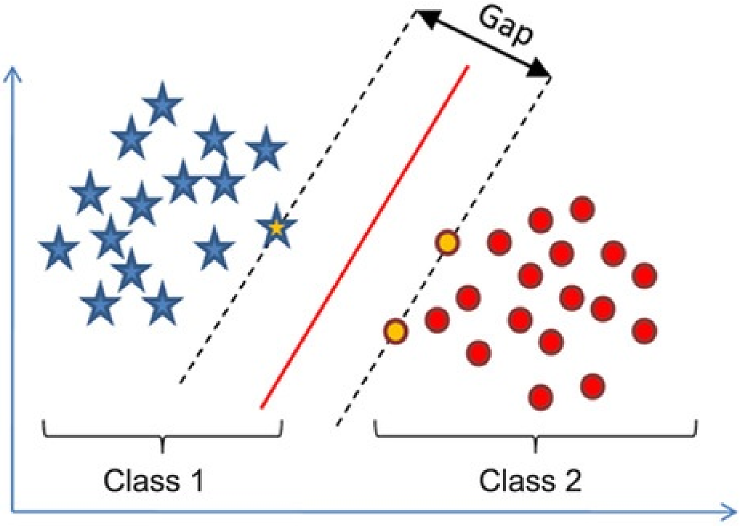
Training Set Size	100	200	300
Training time (secs)	0.015	0.021	0.022
Prediction time (secs)	0.001	0.002	0.002
F1 score for training set	1	1	1
F1 score for test set	0.755	0.72	0.829

5. Choosing the Best Model

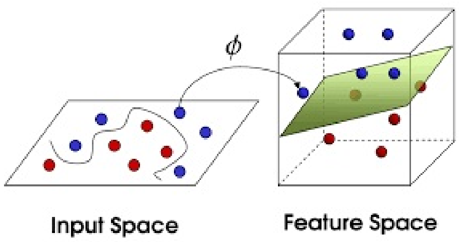
- Based on the experiments you performed earlier, in 1-2 paragraphs explain to the board of supervisors what single model you chose as the best model. Which model is generally the most appropriate based on the available data, limited resources, cost, and performance?
- In 1-2 paragraphs explain to the 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: The best model to choose from would be the Support Vector Machine (SVM). First reason would be that as the data continued to increase, SVM performed the best with Random Forest coming in at a close second. Guassian Naive Bayes had a fast traininig and predicting time, but had a lower F1 score compared to the other algorithms. It took random forest much longer to train compared to SVM. SVM would give a high performance that could handle large datasets without having as high as a training time than Random Forest.

A simple support vector machine (SVM) is a type of algorithm that attempts to separate the students who passed (blue stars) and those who failed (red circles) with an optimized line. SVM would like to separate the the two categories by a maximum margin (gap). This is so that unseen student examples are more likely to be classified correctly.



Now given a set of training examples, each marked for belonging to either pass or fail, an SVM training algorithm builds a model that assigns new examples into one category or the other. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on. SVM also has a technique called the kernel trick. These are functions which takes a non separable problem and transforms it into a separable one. It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then find out the process to separate the data based on the labels or outputs thats defined.



In [ ]:

```
# TODO: Fine-tune your model and report the best F1 score
from sklearn.grid_search import GridSearchCV
from sklearn.cross_validation import StratifiedShuffleSplit
from sklearn.metrics import make_scorer

parameters = {'C': [1, 10, 100, 200, 300, 400, 500, 600, 700, 800, 700, 800, 900, 1000],
              'gamma': [.0001,.001, .01, .1, 1, 10, 100, 200, 300, 400]}


f1_scorer = make_scorer(f1_score, pos_label="yes")

clf_gs = GridSearchCV(svm.SVC(), parameters, scoring=f1_scorer)
clf_gs.fit(X_train, y_train)

# Select the best settings for classifier
best_clf = clf_gs.best_estimator_

# Fit the algorithm to the training data
print ("Fine-tuned Model: ")
print (best_clf)
print ('\n')
train_classifier(best_clf, X_train, y_train)

# Test algorithm's performance
print ("F1 score for training set: {}".format(predict_labels(best_clf, X_train, y_train)))
print ("F1 score for test set: {}".format(predict_labels(best_clf, X_test, y_test)))
```