Machine Learning Engineer Nanodegree

Supervised Learning

Project 2: Building a Student Intervention System

Welcome to the second project of the Machine Learning Engineer Nanodegree! In this notebook, some template code has already been provided for you, and it will be your job to implement the additional functionality necessary to successfully complete this project. Sections that begin with **'Implementation'** in the header indicate that the following block of code will require additional functionality which you must provide. Instructions will be provided for each section and the specifics of the implementation are marked in the code block with a `'TODO'` statement. Please be sure to read the instructions carefully!

In addition to implementing code, there will be questions that you must answer which relate to the project and your implementation. Each section where you will answer a question is preceded by a **'Question X'** header. Carefully read each question and provide thorough answers in the following text boxes that begin with **'Answer:'**. Your project submission will be evaluated based on your answers to each of the questions and the implementation you provide.

> **Note:** Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. In addition, Markdown cells can be edited by typically double-clicking the cell to enter edit mode.

Question 1 - Classification vs. Regression

Your goal for this project is to identify students who might need early intervention before they fail to graduate. Which type of supervised learning problem is this, classification or regression? Why?

Answer: Classification, as once a student is identified as positive they need early intervention, whereas regression implies there would be a continuous response rather than a discrete yes or no.

In addition the labels in the data set are "passed" and "failed", categorical data better suiting classification.

A likelihood that a student would fail to graduate would be a regression problem, whereas this one is to identify whether a student requires early intervention or not.

Exploring the Data

Run the code cell below to load necessary Python libraries and load the student data. Note that the last column from this dataset, `'passed'`, will be our target label (whether the student graduated or didn't graduate). All other columns are features about each student.

In [1]:

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

# Read student data
student_data = pd.read_csv("student-data.csv")
print "Student data read successfully!"
```

```
Student data read successfully!
```

Implementation: Data Exploration

Let's begin by investigating the dataset to determine how many students we have information on, and learn about the graduation rate among these students. In the code cell below, you will need to compute the following:

- The total number of students, `n_students`.
- The total number of features for each student, `n_features`.
- The number of those students who passed, `n_passed`.
- The number of those students who failed, `n_failed`.
- The graduation rate of the class, `grad_rate`, in percent (%).

In [2]:

```
# TODO: Calculate number of students
n_students = len(student_data)

# TODO: Calculate number of features
n_features = len(student_data.columns) - 1

# TODO: Calculate passing students
n_passed = len(student_data[student_data['passed'] == 'yes'])

# TODO: Calculate failing students
n_failed = len(student_data[student_data['passed'] == 'no'])

# TODO: Calculate graduation rate
grad_rate = (float(n_passed) / float(n_students))*100

# Print the results
print "Total number of students: {}".format(n_students)
print "Number of features: {}".format(n_features)
print "Number of students who passed: {}".format(n_passed)
print "Number of students who failed: {}".format(n_failed)
print "Graduation rate of the class: {:.2f}%".format(grad_rate)
```

```
Total number of students: 395
Number of features: 30
Number of students who passed: 265
Number of students who failed: 130
Graduation rate of the class: 67.09%
```

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.

Run the code cell below to separate the student data into feature and target columns to see if any features are non-numeric.

In [3]:

```
# Extract feature columns
feature_cols = list(student_data.columns[:-1])

# Extract target column 'passed'
target_col = student_data.columns[-1]

# Show the list of columns
print "Feature columns:\n{}".format(feature_cols)
print "\nTarget column: {}".format(target_col)

# Separate the data into feature data and target data (X_all and y_all, respectiv
```

```
# Separate the data into feature data and target data (X_all and y_all, respectiv
X_all = student_data[feature_cols]
y_all = student_data[target_col]

# Show the feature information by printing the first five rows
print "\nFeature values:"
print X_all.head()
```

```
Feature columns:
['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'F
edu', 'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime
', 'failures', 'schoolsup', '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   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. Run the code cell below to perform the preprocessing routine discussed in this section.

In [4]:

```
def preprocess_features(X):
    ''' Preprocesses the student data and converts non-numeric binary variables i
        binary (0/1) variables. Converts categorical variables into dummy variabl

    # Initialize new output DataFrame
    output = pd.DataFrame(index = X.index)

    # Investigate each feature column for the data
    for col, col_data in X.iteritems():

        # If data type is non-numeric, replace all yes/no values with 1/0
        if col_data.dtype == object:
            col_data = col_data.replace(['yes', 'no'], [1, 0])

        # If data type is categorical, convert to dummy variables
        if col_data.dtype == object:
            # Example: 'school' => 'school_GP' and 'school_MS'
            col_data = pd.get_dummies(col_data, prefix = col)

        # Collect the revised columns
        output = output.join(col_data)

    return output

X_all = preprocess_features(X_all)
print "Processed feature columns ({} total features):\n{}".format(len(X_all.colum
```

```
Processed feature columns (48 total features):
['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', 're
ason_other', 'reason_reputation', 'guardian_father', 'guardian_mothe
r', 'guardian_other', 'traveltime', 'studytime', 'failures', 'school
sup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet
', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Walc', 'healt
h', 'absences']
```

Implementation: Training and Testing Data Split

Implementation: Training and Testing Data Split

So far, we have converted all *categorical* features into numeric values. For the next step, we split the data (both features and corresponding labels) into training and test sets. In the following code cell below, you will need to implement the following:

- Randomly shuffle and split the data (`X_all`, `y_all`) into training and testing subsets.
  - Use 300 training points (approximately 75%) and 95 testing points (approximately 25%).
  - Set a `random_state` for the function(s) you use, if provided.
  - Store the results in `X_train`, `X_test`, `y_train`, and `y_test`.

In [5]:

```
# TODO: Import any additional functionality you may need here
from sklearn import cross_validation

# TODO: Set the number of training points
num_train = 300

# Set the number of testing points
num_test = X_all.shape[0] - num_train

# TODO: Shuffle and split the dataset into the number of training and testing poi
X_train, X_test, y_train, y_test = cross_validation.train_test_split(X_all, y_all
                                                                     /float(X



# Show the results of the split
print "Training set has {} samples.".format(X_train.shape[0])
print "Testing set has {} samples.".format(X_test.shape[0])
```

```
Training set has 300 samples.
Testing set has 95 samples.
```

Training and Evaluating Models

In this section, you will choose 3 supervised learning models that are appropriate for this problem and available in `scikit-learn`. You will first discuss the reasoning behind choosing these three models by considering what you know about the data and each model's strengths and weaknesses. You will then fit the model to varying sizes of training data (100 data points, 200 data points, and 300 data points) and measure the F_1 score. You will need to produce three tables (one for each model) that shows the training set size, training time, prediction time, F_1 score on the training set, and F_1 score on the testing set.

Question 2 - Model Application

List three supervised learning models that are appropriate for this problem. What are the general applications of each model? What are their strengths and weaknesses? Given what you know about the data, why did you choose these models to be applied?

Answer: Random Forest Classifier - This algorithm is an ensemble method based on decision trees. It models many decision trees on subsets of the data to help reduce overfitting.
http://www.nickgillian.com/wiki/pmwiki.php/GRT/RandomForests

(http://www.nickgillian.com/wiki/pmwiki.php/GRT/RandomForests) Decision trees are useful here as our problem has a high number of features relative to our number of data points meaning dimensionality will heavily affect our learning, and decision trees are effective at identifying relevant features and focusing on these. They are also easy and quick to use due to needing little data preparation and being able to handle categorical and numerical data equally.

However, they are still prone to overfitting and are liable to not find the optimal decision tree due to the algorithm splitting using the best local split at each node. http://scikit-learn.org/stable/modules/tree.html (http://scikit-learn.org/stable/modules/tree.html)

The ensemble method's advantage is that it is still easily interpretable and counters the disadvantages stated due to creation of multiple trees. These additional trees add in the random factor that counters local optimum, as well as reducing the effect of overfitting of any one tree. The disadvantage being the additional prediction time having to process each of these trees.

Support Vector Classifier - These maximize the margin and so reduce overfitting error.

They are better with smaller training sets due to a polynomially increasing training time, so would be useful here with the limited resources. They are also effective if the number of features is slightly greater than the number of training samples as they still give good performance, a large amount more features though will give poor performance. They are versatile thanks to the kernel trick and are memory efficient as they only use a subset of the training samples called support vectors.

Another disadvantage is they don't deal well with noise, which is a possibility with the problem here (it's a complex problem with a multitude of factors that won't be all captured in the features). They also do not directly give probability estimates. http://scikit-learn.org/stable/modules/svm.html (http://scikit-learn.org/stable/modules/svm.html)

K Nearest Neighbour - KNN works to work out a similarity metric which separates out the different classes. So more similar data points will be grouped under a single class.

An instance based and non-parametric model such as this one may work better in this scenario as it is possible that the data won't behave in a conventional way or a way suited to parametric methods. However, similarity in certain features is likely to occur for each class, and be a strong indicator. KNN is effective with large datasets and is simple to implement. Additionally, the method allows more data to be added easily should more pupils drop out/fail.

The weaknesses of this algorithm is its variability as different values of K can lead to different classifications. In addition, the versatility given by the distance metrics can make it hard to find a high performance one. It also gives a high prediction time due to the need to compute the distance of each query for all training samples. Additionally, as the model doesn't learn from data it may not generalise well and can be vulnerable to noisy data (as it keeps the data exactly).
http://people.revoledu.com/kardi/tutorial/KNN/Strength%20and%20Weakness.htm (http://people.revoledu.com/kardi/tutorial/KNN/Strength%20and%20Weakness.htm)

Setup

Run the code cell below to initialize three helper functions which you can use for training and testing the three supervised learning models you've chosen above. The functions are as follows:

- `train_classifier` - takes as input a classifier and training data and fits the classifier to the data.
- `predict_labels` - takes as input a fit classifier, features, and a target labeling and makes predictions using the F_1 score.
- `train_predict` - takes as input a classifier, and the training and testing data, and performs `train_clasifier` and `predict_labels`.
  - This function will report the F_1 score for both the training and testing data separately.

In [6]:

```
def train_classifier(clf, X_train, y_train):
    ''' Fits a classifier to the training data. '''

    # Start the clock, train the classifier, then stop the clock
    start = time()
    clf.fit(X_train, y_train)
    end = time()

    # Print the results
    print "Trained model in {:.4f} seconds".format(end - start)


def predict_labels(clf, features, target):
    ''' Makes predictions using a fit classifier based on F1 score. '''

    # Start the clock, make predictions, then stop the clock
    start = time()
    y_pred = clf.predict(features)
    end = time()

    # Print and return results
    print "Made predictions in {:.4f} seconds.".format(end - start)
    return f1_score(target.values, y_pred, pos_label='yes')


def train_predict(clf, X_train, y_train, X_test, y_test):
    ''' Train and predict using a classifer based on F1 score. '''

    # Indicate the classifier and the training set size
    print "Training a {} using a training set size of {}. . .".format(clf.__class

    # Train the classifier
    train_classifier(clf, X_train, y_train)

    # Print the results of prediction for both training and testing
    print "F1 score for training set: {:.4f}.".format(predict_labels(clf, X_train
    print "F1 score for test set: {:.4f}.".format(predict_labels(clf, X_test, y_t
```

Implementation: Model Performance Metrics

With the predefined functions above, you will now import the three supervised learning models of your choice and run the `train_predict` function for each one. Remember that you will need to train and predict on each classifier for three different training set sizes: 100, 200, and 300. Hence, you should expect

to have 9 different outputs below — 3 for each model using the varying training set sizes. In the following code cell, you will need to implement the following:

- Import the three supervised learning models you've discussed in the previous section.
- Initialize the three models and store them in `clf_A`, `clf_B`, and `clf_C`.
  - Use a `random_state` for each model you use, if provided.
- Create the different training set sizes to be used to train each model.
  - *Do not reshuffle and resplit the data! The new training points should be drawn from `X_train` and `y_train`.*
- Fit each model with each training set size and make predictions on the test set (9 in total).
  Note: Three tables are provided after the following code cell which can be used to store your results.

In [7]:

```
# TODO: Import the three supervised learning models from sklearn
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

# TODO: Initialize the three models
clf_A = RandomForestClassifier(random_state = 0)
clf_B = SVC(random_state = 0)
clf_C = KNeighborsClassifier()

# TODO: Set up the training set sizes
X_train_100 = X_train[0:100]
y_train_100 = y_train[0:100]

X_train_200 = X_train[0:200]
y_train_200 = y_train[0:200]

X_train_300 = X_train[0:300]
y_train_300 = y_train[0:300]

# TODO: Execute the 'train_predict' function for each classifier and each trainin
# train_predict(clf, X_train, y_train, X_test, y_test)

for i in [clf_A, clf_B, clf_C]:
    train_predict(i, X_train_100, y_train_100, X_test, y_test)
    train_predict(i, X_train_200, y_train_200, X_test, y_test)
    train_predict(i, X_train_300, y_train_300, X_test, y_test)
```

```
Training a RandomForestClassifier using a training set size of 100.
. .
Trained model in 0.0212 seconds
Made predictions in 0.0011 seconds.
F1 score for training set: 0.9853.
Made predictions in 0.0009 seconds.
F1 score for test set: 0.7808.
Training a RandomForestClassifier using a training set size of 200.
. .
Trained model in 0.0195 seconds
Made predictions in 0.0009 seconds.
F1 score for training set: 0.9960.
```

```
Made predictions in 0.0007 seconds.
F1 score for test set: 0.7424.
Training a RandomForestClassifier using a training set size of 300.
. .
Trained model in 0.0177 seconds
Made predictions in 0.0021 seconds.
F1 score for training set: 0.9896.
Made predictions in 0.0010 seconds.
F1 score for test set: 0.8108.
Training a SVC using a training set size of 100. . .
Trained model in 0.0016 seconds
Made predictions in 0.0012 seconds.
F1 score for training set: 0.8428.
Made predictions in 0.0013 seconds.
F1 score for test set: 0.8810.
Training a SVC using a training set size of 200. . .
Trained model in 0.0047 seconds
Made predictions in 0.0031 seconds.
F1 score for training set: 0.8610.
Made predictions in 0.0015 seconds.
F1 score for test set: 0.8805.
Training a SVC using a training set size of 300. . .
Trained model in 0.0064 seconds
Made predictions in 0.0047 seconds.
F1 score for training set: 0.8442.
Made predictions in 0.0015 seconds.
F1 score for test set: 0.8889.
Training a KNeighborsClassifier using a training set size of 100. .
.
Trained model in 0.0005 seconds
Made predictions in 0.0011 seconds.
F1 score for training set: 0.8105.
Made predictions in 0.0010 seconds.
F1 score for test set: 0.8447.
Training a KNeighborsClassifier using a training set size of 200. .
.
Trained model in 0.0005 seconds
Made predictions in 0.0025 seconds.
F1 score for training set: 0.8261.
Made predictions in 0.0014 seconds.
F1 score for test set: 0.8442.
Training a KNeighborsClassifier using a training set size of 300. .
.
Trained model in 0.0006 seconds
Made predictions in 0.0051 seconds.
F1 score for training set: 0.8509.
Made predictions in 0.0024 seconds.
F1 score for test set: 0.8553.
```

Tabular Results

Edit the cell below to see how a table can be designed in Markdown (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#tables). You can record your results from above in the tables provided.

Classifer 1 - Random Forest

Training Set Size	Prediction Time (train)	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0009	0.0008	0.9853	0.7808
200	0.0022	0.0017	0.9960	0.7424
300	0.0019	0.0017	0.9896	0.8108

Classifer 2 - Support Vector Machine

Training Set Size	Prediction Time (train)	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0014	0.0014	0.8428	0.8810
200	0.0035	0.0012	0.8610	0.8805
300	0.0048	0.0015	0.8442	0.8889

Classifer 3 - K-Nearest-Neighbour

Training Set Size	Prediction Time (train)	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0040	0.0015	0.8105	0.8447
200	0.0059	0.0064	0.8261	0.8442
300	0.0054	0.0031	0.8509	0.8553

Choosing the Best Model

In this final section, you will choose from the three supervised learning models the *best* model to use on the student data. You will then perform a grid search optimization for the model over the entire training set (`X_train` and `y_train`) by tuning at least one parameter to improve upon the untuned model's F_1 score.

Question 3 - Chosing the Best Model

Based on the experiments you performed earlier, in one to two 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?

Answer: I believe the SVC offers the best choice of model out of the three analysed here. It has offered the best F1 Score on the test set for all 3 data sets, and with the score varying little between the three different sizes it has proved it is not reliant on a large amount of data, important given the limited amount. In addition, it consistently offers a low prediction time on the test set, beaten only by the RandomForestClassifier which suffers with lower performance.

While prediction time (train) is high for the largest training set (relevant for the training and optimisation needed), this is negated by the smaller data sets performing as well, so less data can be used to counter limited resources. Another factor is that KNN is much faster in training as it does much of its computation

here, and while this is likely not important here as the dataset isn't treated as online, this could be a factor should this change.

Overall though, I believe the school would value accuracy, as despite the limited resources this is a problem which is likely noisy but important, and so should be calculated effectively. This is offset slightly by the human input still needed to interpret results of a problem like this, that involves people so intrinsically.

Question 4 - Model in Layman's Terms

In one to two paragraphs, explain to the board of directors in layman's terms how the final model chosen is supposed to work. For example if you've chosen to use a decision tree or a support vector machine, how does the model go about making a prediction?

In [ ]:

```
**Answer: ** The model works by finding the best linear separator between the cla

|x    x    x   /->-   - |
|    x    x   /->-   -  |
|x       x   /          |
|     x  x<-/    - - -  |
|  x    x  /->-         |

For datasets where a linear separator does not exist with the current set of feat

An example of this is if there are two continuous features x and y plotted to a t
      y              y
      |              |
    + | +            |     +
  +   |   +          |     +
    - | -            | -
 ------------ x     z--------------
 +   - | -   +      | -
   +   |   +        |     +
     + | +          |     +
       |            |                  '''
Once this line is found new points can be mapped to these additional dimensions (
      y              y
      |              |
    + | +            |     +
  +   |F  +          | F   +
    - | -            | -
------------ x     z--------------
+   - | -   +      | -
  +   |   +        |     +
    + | +          |     +
      |            |
Here "F" is likely in the class "-".
```

Implementation: Model Tuning

Fine tune the chosen model. Use grid search (`GridSearchCV`) with at least one important parameter tuned with at least 3 different values. You will need to use the entire training set for this. In the code cell below, you will need to implement the following:

- Import `sklearn.grid_search.gridSearchCV` (http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html) and `sklearn.metrics.make_scorer` (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.make_scorer.html).
- Create a dictionary of parameters you wish to tune for the chosen model.
  - Example: `parameters = {'parameter' : [list of values]}`.
- Initialize the classifier you've chosen and store it in `clf`.
- Create the F_1 scoring function using `make_scorer` and store it in `f1_scorer`.
  - Set the `pos_label` parameter to the correct value!
- Perform grid search on the classifier `clf` using `f1_scorer` as the scoring method, and store it in `grid_obj`.
- Fit the grid search object to the training data (`X_train`, `y_train`), and store it in `grid_obj`.

In [8]:

```
# TODO: Import 'gridSearchCV' and 'make_scorer'
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import make_scorer

# TODO: Create the parameters list you wish to tune
parameters = {'kernel': ('rbf','linear','poly'), 'C': [0.5,1,1.5], 'gamma': [0.5,

# TODO: Initialize the classifier
clf = SVC(random_state = 0)

# TODO: Make an f1 scoring function using 'make_scorer'
f1_scorer = make_scorer(f1_score, pos_label = "yes")

# TODO: Perform grid search on the classifier using the f1_scorer as the scoring
grid_obj = GridSearchCV(clf, parameters, scoring = f1_scorer)

# TODO: Fit the grid search object to the training data and find the optimal para
grid_obj = grid_obj.fit(X_train, y_train)

# Get the estimator
clf = grid_obj.best_estimator_

# Report the final F1 score for training and testing after parameter tuning
print "Tuned model has a training F1 score of {:.4f}.".format(predict_labels(clf,
print "Tuned model has a testing F1 score of {:.4f}.".format(predict_labels(clf,
```

```
Made predictions in 0.0056 seconds.
Tuned model has a training F1 score of 0.7755.
Made predictions in 0.0019 seconds.
Tuned model has a testing F1 score of 0.8824.
```

Question 5 - Final F_1 Score

What is the final model's F_1 score for training and testing? How does that score compare to the untuned model?

Answer: The tuned model performance of 0.8824 varies minimally from the untuned version (and is in fact slightly worse), implying the defaults were among the best performing parameters for this model.

The training score did however rise a lot, showing the extra degrees of freedom found in the new parameter settings only contributed to overfitting rather than ability to generalise.

> **Note**: Once you have completed all of the code implementations and successfully answered each question above, you may finalize your work by exporting the iPython Notebook as an HTML document. You can do this by using the menu above and navigating to
> **File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.