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:
It is a classification problem.
Because the output result is whether the student fail or not. This is a discrete output.

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!"

#from IPython.display import display
#display(student_data.head())
```

```
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.index)

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

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

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

# TODO: Calculate graduation rate
# grad_rate = n_passed/n_students #### need to turn into float number
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, respectively)
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', 'Fedu', 'M
job', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime', 'failures',
'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'interne
t', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Walc', 'health', 'ab
sences']

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 healt
h  \
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 into
        binary (0/1) variables. Converts categorical variables into dummy variables. '''

    # 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.columns), lis
t(X_all.columns))
```

```
Processed feature columns (48 total features):
['school_GP', 'school_MS', 'sex_F', 'sex_M', 'age', 'address_R', 'address_
U', 'famsize_GT3', 'famsize_LE3', 'Pstatus_A', 'Pstatus_T', 'Medu', 'Fedu',
'Mjob_at_home', 'Mjob_health', 'Mjob_other', 'Mjob_services', 'Mjob_teache
r', 'Fjob_at_home', 'Fjob_health', 'Fjob_other', 'Fjob_services', 'Fjob_teac
her', 'reason_course', 'reason_home', 'reason_other', 'reason_reputation',
'guardian_father', 'guardian_mother', 'guardian_other', 'traveltime', 'study
time', 'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery',
'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Wa
lc', 'health', 'absences']
```

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.cross_validation import train_test_split
# 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 points above
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_size = 95, random_s
tate=42)



# 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:

1. I will choose k-NN, support vector machines, and naive Bayes.
2. All of these three models can be applied to classification problem. For example, k-NN can be used in hand writting recognition, SVM has been applied to text categorization, naive Bayes can be used in spam e-mail classification.
3. k-NN:
   Advantage: Easy to use. Don't need to take addtional time to train the model when new data comes in.
   Disadvantage: Has to use a lot of space to store original data. The amount of calculation is relatively large when predicting a test example.
   SVM:
   Advantage: The classification hypeplane SVM find is robust. SVM can solve high-dimensional problem. With kernel method, SVM can also applyed to non-linear classification problem.
   Disadvantage: SVM is sensitive to data noise. The training process of SVM is complex.
   naive Bayes:
   Advantage: NB has reliable mathemetics foundation, and it's easy to explain. Parameter set in NB is relatively small.
   Disadvantage: NB assumes features are indepedent of each other. However this is often not true.
4. Since this problem is high dimensional, I think the classification boundary could be rather complex. So I use k-NN, a lazy learning method. It can provide a strange shape boundary than a simple linear model, such as Logistic regression.
   For similar reason, I will also try SVM with Gaussian kernel.
   Given the fact there are a lot of boolean features in the data set, it is quite similar to the spam e-mail classification in the lecture video. So I will also check if naive Bayes works.

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__.__name
__, len(X_train))

    # 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, y_trai
n))
    print "F1 score for test set: {:.4f}.".format(predict_labels(clf, X_test, y_test))
```

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.
  - **Note:** Use the default settings for each model — you will tune one specific model in a later section.
- 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 [9]:

```
# TODO: Import the three supervised learning models from sklearn
# from sklearn import model_A
# from sklearn import model_B
# from skearln import model_C
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB

# TODO: Initialize the three models
clf_A = KNeighborsClassifier(n_neighbors = 5) # KNN model with k = 5
clf_B = SVC(C = 2, kernel = 'rbf') # SVM model with Gaussian kernel, regularization parame
ter C = 2
clf_C = MultinomialNB(alpha = 1) # naive Bayes model, use MultinomialNB rather than Gaussi
anNB

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

# print len(y_train_100)

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

# print len(y_train_200)

X_train_300 = X_train
y_train_300 = y_train

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

"""
# model A, training set size = 100,200,300
train_predict(clf_A, X_train_100, y_train_100, X_test, y_test)
print '\n'
train_predict(clf_A, X_train_200, y_train_200, X_test, y_test)
print '\n'
train_predict(clf_A, X_train_300, y_train_300, X_test, y_test)
print '\n'

# model B, training set size = 100,200,300
train_predict(clf_B, X_train_100, y_train_100, X_test, y_test)
print '\n'
train_predict(clf_B, X_train_200, y_train_200, X_test, y_test)
print '\n'
train_predict(clf_B, X_train_300, y_train_300, X_test, y_test)
print '\n'

# model C, training set size = 100,200,300
train_predict(clf_C, X_train_100, y_train_100, X_test, y_test)
print '\n'
train_predict(clf_C, X_train_200, y_train_200, X_test, y_test)
print '\n'
train_predict(clf_C, X_train_300, y_train_300, X_test, y_test)
print '\n'
```

```
"""

# use for loop
for clf in [clf_A, clf_B, clf_C]:
  for size in [100, 200, 300]:
      train_predict(clf, X_train[:size], y_train[:size], X_test, y_test)
      print '\n'
```

```
Training a KNeighborsClassifier using a training set size of 100. . .
Trained model in 0.0010 seconds
Made predictions in 0.0020 seconds.
F1 score for training set: 0.8060.
Made predictions in 0.0020 seconds.
F1 score for test set: 0.7246.


Training a KNeighborsClassifier using a training set size of 200. . .
Trained model in 0.0000 seconds
Made predictions in 0.0030 seconds.
F1 score for training set: 0.8800.
Made predictions in 0.0030 seconds.
F1 score for test set: 0.7692.


Training a KNeighborsClassifier using a training set size of 300. . .
Trained model in 0.0010 seconds
Made predictions in 0.0080 seconds.
F1 score for training set: 0.8809.
Made predictions in 0.0040 seconds.
F1 score for test set: 0.7801.


Training a SVC using a training set size of 100. . .
Trained model in 0.0020 seconds
Made predictions in 0.0010 seconds.
F1 score for training set: 0.9078.
Made predictions in 0.0010 seconds.
F1 score for test set: 0.7536.


Training a SVC using a training set size of 200. . .
Trained model in 0.0040 seconds
Made predictions in 0.0030 seconds.
F1 score for training set: 0.9262.
Made predictions in 0.0010 seconds.
F1 score for test set: 0.7639.


Training a SVC using a training set size of 300. . .
Trained model in 0.0080 seconds
Made predictions in 0.0060 seconds.
F1 score for training set: 0.9172.
Made predictions in 0.0020 seconds.
F1 score for test set: 0.7972.


Training a MultinomialNB using a training set size of 100. . .
Trained model in 0.0010 seconds
Made predictions in 0.0000 seconds.
F1 score for training set: 0.8000.
Made predictions in 0.0000 seconds.
F1 score for test set: 0.7538.


Training a MultinomialNB using a training set size of 200. . .
```

```
Trained model in 0.0010 seconds
Made predictions in 0.0000 seconds.
F1 score for training set: 0.8057.
Made predictions in 0.0000 seconds.
F1 score for test set: 0.7941.


Training a MultinomialNB using a training set size of 300. . .
Trained model in 0.0020 seconds
Made predictions in 0.0000 seconds.
F1 score for training set: 0.7954.
Made predictions in 0.0000 seconds.
F1 score for test set: 0.7941.
```

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 - k-NN with k = 5

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0010s	0.0020s	0.8060	0.7246
200	0.0010s	0.0020s	0.8800	0.7692
300	0.0010s	0.0030s	0.8809	0.7801

Classifer 2 - SVM with Gaussian kernel

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0010s	0.0010s	0.9078	0.7536
200	0.0050s	0.0010s	0.9262	0.7639
300	0.0100s	0.0020s	0.9172	0.7972

Classifer 3 - Multinomial naive Bayes

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0010s	0.0000s	0.8000	0.7538
200	0.0010s	0.0000s	0.8057	0.7941
300	0.0010s	0.0000s	0.7954	0.7941

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:
Based on the experiment result, I will choose Multinomial naive Bayes as my final model.
The training time that NB takes is just as quick as the lazy learning method k-NN, and obviously it is much faster than training a SVM. Its prediction time is the shortest of the three. So NB will need the least computaion resourses.
In addition, the F1 score of NB is also high, only a little lower than SVM with training set size = 300. The F1 score on training set is similar to that on the test set, which means we don't suffer from overfitting problem when using naive Bayes method.

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?

Answer:
Given a new example, the naive Bayes model will first calculate the probability that this student will pass based on his or her features, P(y_i = 1 | x_i), and the probability that he or she will fail, P(y_i = 0 | x_i). Then the model will predict the result who has a higher probability.

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)](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)](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 [10]:

```
# TODO: Import 'GridSearchCV' and 'make_scorer'

from sklearn import grid_search
from sklearn.metrics import make_scorer
# TODO: Create the parameters list you wish to tune
alphaList = np.linspace(0.1,2,10)

parameters = {'alpha':alphaList,'fit_prior': (True,False)}

# TODO: Initialize the classifier
clf = MultinomialNB()

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

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

# TODO: Fit the grid search object to the training data and find the optimal parameters
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, X_train, y_train))
print "Tuned model has a testing F1 score of {:.4f}.".format(predict_labels(clf, X_test, y_test))
```

```
Made predictions in 0.0000 seconds.
Tuned model has a training F1 score of 0.7954.
Made predictions in 0.0010 seconds.
Tuned model has a testing F1 score of 0.7941.
```

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 Final F1 score for training set is 0.7954. F1 score for test set is 0.7941.
They are the same as untuned model, since naive Bayes model don't need adjust paremeter actually.

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.