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 classification problem. Because the output is discrete labels: either 'yes' or 'no'. If the output is continuous numeric values, it is regression problem.

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 [115]:
```
# TODO: Calculate number of students
n_students = student_data.shape[0]

# TODO: Calculate number of features
n_features = student_data.shape[1]-1

# TODO: Calculate passing students
n_passed = student_data[student_data['passed']=='yes'].shape[0]

# TODO: Calculate failing students
n_failed = student_data[student_data['passed']=='no'].shape[0]

# TODO: Calculate graduation rate
grad_rate = float(n_passed)/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 [26]:
```
# 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, respec
tively)
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', 'Mj
ob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime', 'failures', 'sc
hoolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet', 'r
omantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Walc', 'health', 'absence
s']

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. Run the code cell below to perform the preprocessing routine discussed in this section.

In [27]:
```
def preprocess_features(X):
    ''' Preprocesses the student data and converts non-numeric binary variable
s into
        binary (0/1) variables. Converts categorical variables into dummy vari
ables. '''

    # 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.co
lumns), list(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', 'Mjo
b_at_home', 'Mjob_health', 'Mjob_other', 'Mjob_services', 'Mjob_teacher', 'Fj
ob_at_home', 'Fjob_health', 'Fjob_other', 'Fjob_services', 'Fjob_teacher', 'r
eason_course', 'reason_home', 'reason_other', 'reason_reputation', 'guardian_
father', 'guardian_mother', 'guardian_other', 'traveltime', 'studytime', 'fai
lures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'in
ternet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Walc', '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 [33]:
```
# TODO: Import any additional functionality you may need here
from sklearn import cross_validation as cv
# TODO: Set the number of training points
num_train = 300

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

X_train, X_test, y_train, y_test = cv.train_test_split(X_all,y_all,test_size=n
um_test, random_state=1)

# TODO: Shuffle and split the dataset into the number of training and testing
 points above
#X_train = None
#X_test = None
#y_train = None
#y_test = None

# 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: The three supervised learning models that I chose are logistic regression, decision tree, and naive bayes. All three algos are capabale of any classification problem, but they do have their strengths and weekness. Specifically, if you are trying to predic sentiment from product reviews, logistic regression is good choice. if you are not sure about whether to give a loan to someone given personal information, decision tree is the choice. And naive bayes is very useful natural language processing.

Logistic regression is easy to use and computationally economy, and once the estimator is trained, we don't need to store the old example data to predict the new data. But it's prone to underfitting.

Decision tree is also very easy to use, and can help us to understanding data better, since it is actually same with the human decision process. But it is prone to overfitting if given lots of features.

Naive bayes is not sensitive to several errors/noise in the trainning data, because it depends on the overall possibility of the data, and it can give accurate result if the inter-independence condition is satified. But inter-independence condition between features is actually hard to be fully satified. Most features are related with some other features to some extent. Also, it is not suitbal for very high dimensional settings.

Looking at our dataset, some features are continuous numeric, this can be best handeled by logistic regression. Some features are labels, that is best suitble for decision trees. And for naive bayes, since most of our features are discrete values, it is more convenient for naive bayes to make accurate predictions.

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 [34]:
```
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.__cl
ass__.__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_tr
ain, y_train))
    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 [117]:
```
# TODO: Import the three supervised learning models from sklearn
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier


# TODO: Initialize the three models
clf_A = GaussianNB()
clf_B = LogisticRegression(random_state=1)
clf_C = DecisionTreeClassifier(random_state=1)

# 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 trai
ning set size
# train_predict(clf, X_train, y_train, X_test, y_test)
clf = [clf_A, clf_B, clf_C]
train_data = [(X_train_100,y_train_100),(X_train_200,y_train_200),
(X_train_300,y_train_300)]
for clfn in clf:
    for xtrain, ytrain in train_data:
        train_predict(clfn, xtrain, ytrain, X_test, y_test)
```

```
Training a GaussianNB using a training set size of 100. . .
Trained model in 0.0050 seconds
Made predictions in 0.0000 seconds.
F1 score for training set: 0.8346.
Made predictions in 0.0000 seconds.
F1 score for test set: 0.7402.
Training a GaussianNB using a training set size of 200. . .
Trained model in 0.0030 seconds
Made predictions in 0.0010 seconds.
F1 score for training set: 0.7879.
Made predictions in 0.0000 seconds.
F1 score for test set: 0.6446.
Training a GaussianNB using a training set size of 300. . .
Trained model in 0.0020 seconds
Made predictions in 0.0010 seconds.
F1 score for training set: 0.7921.
Made predictions in 0.0010 seconds.
F1 score for test set: 0.6720.
Training a LogisticRegression using a training set size of 100. . .
Trained model in 0.0030 seconds
Made predictions in 0.0050 seconds.
F1 score for training set: 0.8529.
Made predictions in 0.0020 seconds.
F1 score for test set: 0.7737.
Training a LogisticRegression using a training set size of 200. . .
Trained model in 0.0030 seconds
Made predictions in 0.0010 seconds.
F1 score for training set: 0.8269.
Made predictions in 0.0010 seconds.
F1 score for test set: 0.7857.
Training a LogisticRegression using a training set size of 300. . .
Trained model in 0.0050 seconds
Made predictions in 0.0000 seconds.
F1 score for training set: 0.8337.
Made predictions in 0.0000 seconds.
F1 score for test set: 0.7368.
Training a DecisionTreeClassifier using a training set size of 100. . .
Trained model in 0.0020 seconds
Made predictions in 0.0000 seconds.
F1 score for training set: 1.0000.
Made predictions in 0.0010 seconds.
F1 score for test set: 0.6829.
Training a DecisionTreeClassifier using a training set size of 200. . .
Trained model in 0.0020 seconds
Made predictions in 0.0000 seconds.
F1 score for training set: 1.0000.
Made predictions in 0.0000 seconds.
F1 score for test set: 0.7023.
Training a DecisionTreeClassifier using a training set size of 300. . .
Trained model in 0.0020 seconds
Made predictions in 0.0000 seconds.
F1 score for training set: 1.0000.
Made predictions in 0.0000 seconds.
F1 score for test set: 0.6984.
```

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)](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#tables). You can record your results from above in the tables provided.

Classifer 1 - GaussianNB

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.005	0.000	0.8346	0.7402
200	0.003	0.000	0.7879	0.6446
300	0.002	0.001	0.7921	0.6720

Classifer 2 - LogisticRegression

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.003	0.005	0.8529	0.7737
200	0.005	0.001	0.8269	0.7857
300	0.002	0	0.8337	0.7368

Classifer 3 - DecisionTreeClassifier

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0010	0	1	0.6829
200	0.0010	0	1	0.7023
300	0.0030	0	1	0.6984

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: Base on the test result, the logistic regression should be chose as the best model. Although the training F1 score of decision tree is 1, which means it successfully seperate all different labels. But its poor testing results indicate a serious overfitting. This can be improved by limit on maximum tree depth. For naive bayes, it testing score is also smaller than that of logistic regression. From the train/test computation time, as far as we get from this results, decision tree is the fastest algo with lowest training and testing time. Logistic regression is relatively more computationally costly in both trainning and testing phase. Naive bayes is in the middle. But in general, all three algos didn't show much difference in terms of computation cost. Thus, we pick logistic regression as our best model because it has highest testing F1 score

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: When we try to predict whether a student will passing or not passing the exam, we will look at his/her performance in school. For example, if the student have multiple absence during the last semester, we may have more confidence in saying that he/she will not pass the exam. Logistic regression is doing such things by helping to us to find the exact threshold number of absence that will lead to not passing exam.

(backup for personal use) Logistic regression is very similar to linear regression, except for it outputs discrete labels. For example, suppose we have only one feature, absence, as the input. We may know that absence is a negative indicator to passing the exam. For linear regression, we give a weight to the feature of absence, and fomulate the estimator as y=wx+b, w is a negative value, and b is some constant. So we now have a numeric value y as our prediction: if y is a small number, it tends to predict not passing exam, if y is a large number, it tends to predict passing the exam. However, we want our estimator to predict labels, other than continuous numeric values. So we feed y into a sigmoid function that maps y into [0,1], and require if the output is less than 0.5, then it predict not passing exam, and if output is larger than 0.5, then it predicts passing the exam. This is basically the process of logistic regression. If more features are added, simply add more weights to our equations, and the algorithms will find the optimal solution to separate different labels.

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 [112]:
```
# TODO: Import 'GridSearchCV' and 'make_scorer'
from sklearn import grid_search
from sklearn.metrics import make_scorer,fbeta_score
# TODO: Create the parameters list you wish to tune
parameters = {'C':[0.001,0.01,0.1,1,10]}


# TODO: Initialize the classifier
clf = LogisticRegression()

# TODO: Make an f1 scoring function using 'make_scorer'
f1_scorer = make_scorer(fbeta_score, beta=1, pos_label = 'yes')

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

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

```
Made predictions in 0.0000 seconds.
Tuned model has a training F1 score of 0.7968.
Made predictions in 0.0010 seconds.
Tuned model has a testing F1 score of 0.8272.
```

In [113]:
```
grid_obj.best_estimator_
```

Out[113]:
```
LogisticRegression(C=0.001, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [114]:
```
grid_obj.grid_scores_
```

Out[114]:
```
[mean: 0.79518, std: 0.00000, params: {'C': 0.001},
 mean: 0.79251, std: 0.00553, params: {'C': 0.01},
 mean: 0.78259, std: 0.01718, params: {'C': 0.1},
 mean: 0.77025, std: 0.00792, params: {'C': 1},
 mean: 0.76742, std: 0.00697, params: {'C': 10}]
```

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 and testing are 0.7968 and 0.8272 respectively. Compared to the untuned ones, the F1 scores improved over 4%. From the results, we may say the untuned logistic models is overfitted. Since we increased our regularization term (decreasing parameter C in LogisticRegression), we get better testing results.

> **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.