student_intervention

July 8, 2016

1 Machine Learning Engineer Nanodegree

1.1 Supervised Learning

1.2 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.

1.2.1 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: This supervised learning belongs to classification as by the definition of classification "given a known relationship, identify the class that the data belongs to".So the following project distinguish for whether the student will graduate or not.

1.3 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!
```

1.3.1 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 = student_data.passed.str.contains('yes').sum()

# TODO: Calculate failing students
n_failed = student_data.passed.str.contains('no').sum() #or we can do n_failed=n_students-n_pas

# 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%
```

1.4 Preparing the Data

In this section, we will prepare the data for modeling, training and testing.

1.4.1 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', 'Mjob', 'Fjob', 'reason', 'gua

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]
```

1.4.2 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()` 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), list(X_al
```

```
Processed feature columns (48 total features):
['school_GP', 'school_MS', 'sex_F', 'sex_M', 'age', 'address_R', 'address_U', 'famsize_GT3', 'famsize_LE3'
```

1.4.3 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, train_size=num_train, test_si

        # 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.
```

1.5 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 F1 score. You will need to produce three tables (one for each model) that shows the training set size, training time, prediction time, F1 score on the training set, and F1 score on the testing set.

1.5.1 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 models that I found suitable for the problem were: Support Vector Machines (SVM), Decision Trees(DT) and Naive Bayes (NB).The choice of models was based on the success of these algorithms in studies in the scientific literature as being trustworthy for classification problems and also for small data sets with good performance.

1) Support Vector Machines

- General Applications: SVMs have demonstrated highly competitive performance in numerous real-world applications, such as bioinformatics, text mining, face recognition, and image processing.
- Strengths: Some of its advantages are that SVM is very effective in high-dimensional spaces, and in situations when we have a non-linear separation problem. With SVM we have the possibility to apply new kernels that allows flexibility for our decision boundaries, leading to a better classification performance.
- Weaknesses: One major disadvantage of the SVM is the choice of the kernel and also it is kind of slower when compared to some other models like Decision trees or Naive bayes.
- Why I chose this model: I chose this model as given a small number of data samples SVMs work well and applying different kernels provide flexibility for ourdecision boundaries thus leading to a better classification performance.

2) Decision Trees

- General Applications: Decision trees have long been important areas of application in the field of medical research and practice. Recent uses of automatic induction of decision trees can be found in diagnosis, cardiology, psychiatry, gastroenterology, for detecting microcalcifications in mammography, to analyze Sudden Infant Death(SID) syndrome and for diagnosing thyroid disorders.
- Strengths: The advantages of a decision trees are that nonlinear relationships between parameters do not affect our performance metrics and they give us faster prediction as compared to some other models like SVMs.
- Weaknesses: Decision Trees do not work well if we have smooth boundaries. i.e they work best when we have discontinuous piece wise constant model. If we truly have a linear target function decision trees are not the best.
- Why I chose this model: I chose this model as decision tree provides a good performance metric for non-linear data. So, if our student data would come out to be non-linear then decision trees would be a better option.

3) Naive Bayes

- General Applications: Naive Bayes methods can be used to mark an email as spam or not spam, to check a piece of text expressing positive emotions, or negative emotions and also in face recognition softwares.
- Strengths: The advantages of naive bayes is that a NB classifier will converge quicker than discriminative models like logistic regression, so we need less training data.

- Weaknesses: The same conditional independence assumption can be a disadvantage when we have no occurrences of a class label and a feature value together, what will give us a zero frequency-based value probability that affects any posterior probability estimate.
- Why I chose this model: I chose this model because if the NB conditional independence assumption actually holds, the Naive Bayes classifier will converge quicker than discriminative models like logistic regression, so we'll need less training data which is in accordance with the number of data samples given to us.

1.5.2 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 F1 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 F1 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__, le

    # 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_train))
```

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

1.5.3 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 [8]:
```
# TODO: Import the three supervised learning models from sklearn
from sklearn import tree
from sklearn import svm
from sklearn.naive_bayes import GaussianNB

# TODO: Initialize the three models
clf_A = tree.DecisionTreeClassifier(random_state=42)
clf_B = svm.SVC(random_state=42)
clf_C = GaussianNB()

# TODO: Set up the training set sizes

# TODO: Execute the 'train_predict' function for each classifier and each training set size
for size in [100, 200, 300]:
    train_predict(clf_A, X_train[:size], y_train[:size], X_test, y_test)
for size in [100, 200, 300]:
    train_predict(clf_B, X_train[:size], y_train[:size], X_test, y_test)
for size in [100, 200, 300]:
    train_predict(clf_C, X_train[:size], y_train[:size], X_test, y_test)
```

```
Training a DecisionTreeClassifier using a training set size of 100. . .
Trained model in 0.0010 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: 0.6552.


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: 0.7500.


Training a DecisionTreeClassifier using a training set size of 300. . .
Trained model in 0.0040 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: 0.6613.


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: 0.8777.
Made predictions in 0.0010 seconds.
F1 score for test set: 0.7746.


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: 0.8679.
Made predictions in 0.0010 seconds.
F1 score for test set: 0.7815.


Training a SVC using a training set size of 300. . .
Trained model in 0.0100 seconds
Made predictions in 0.0060 seconds.
F1 score for training set: 0.8761.
Made predictions in 0.0030 seconds.
F1 score for test set: 0.7838.


Training a GaussianNB using a training set size of 100. . .
Trained model in 0.0010 seconds
Made predictions in 0.0010 seconds.
F1 score for training set: 0.8467.
Made predictions in 0.0000 seconds.
F1 score for test set: 0.8029.


Training a GaussianNB using a training set size of 200. . .
Trained model in 0.0010 seconds
Made predictions in 0.0010 seconds.
F1 score for training set: 0.8406.
Made predictions in 0.0000 seconds.
F1 score for test set: 0.7244.


Training a GaussianNB using a training set size of 300. . .
Trained model in 0.0010 seconds
Made predictions in 0.0010 seconds.
F1 score for training set: 0.8038.
Made predictions in 0.0000 seconds.
F1 score for test set: 0.7634.
```

1.5.4 Tabular Results

Edit the cell below to see how a table can be designed in Markdown. You can record your results from above in the tables provided.

** Classifer 1 - Decision Trees**

Training Set Size	Prediction Time (train)	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0010 seconds	0.0000 seconds	1.0000	0.6552
200	0.0000 seconds	0.0000 seconds	1.0000	0.7500
300	0.0000 seconds	0.0000 seconds	1.0000	0.6613

** Classifer 2 - SVMs**

Training Set Size	Prediction Time (train)	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0010 seconds	0.0010 seconds	0.8777	0.7746
200	0.0040 seconds	0.0020 seconds	0.8679	0.7815
300	0.0090 seconds	0.0030 seconds	0.8761	0.7838

** Classifer 3 - Naive Bayes**

Training Set Size	Prediction Time (train)	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0010 seconds	0.0010 seconds	0.8467	0.8029
200	0.0010 seconds	0.0000 seconds	0.8406	0.7244
300	0.0010 seconds	0.0010 seconds	0.8038	0.7634

1.6 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 F1 score.

1.6.1 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: From the results of the three models evaluated, and from the mean of their metrics results, for the F1 score, the method that had the highest prediction rate on the test set was the Support Vector Machines with an average F1 score of (0.7799) when compared with the average of the other two candidate models Decision Trees (0.6589) and Naive Bayes (0.7635).Although the training and prediction performance is a bit slower as compared to other methods,we can choose SVM as the best model for this case as with a better selection of features SVM compensates for its slower performance.A reassessment of the features can reduce their number to only those features that are really important to the prediction process, thus increasing it's performance.We can see from the performance metrics that even for 200 data samples the SVM model gives better performance metric on test data(0.7815) as compared to the rest two thus revealing that SVM is a better choice when we have a limited number of available data.

1.6.2 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: The model chosen to classify students likely to suffer intervention or not, is the Support Vector Machines(SVMs). The model uses the data from the previous students (age, sex, family, etc.) to make predictions on the new students' data. We can understand the working of SVMs as let's suppose we try to fit a sheet of paper in between two clouds of points, the margin could then be described as us trying to place this sheet of paper as dead center between the two clouds of points as possible (so as to avoid touching either cloud).Also we can use different kernel trick(to deal with non-linearity) which can be described as the similarity measure by which we determine whether two points are close or far.So,this model will undergo a training process with the data of these students and will create a margin between a set of students who failed and those who managed to pass.After establishing this margin that separates our class well for the model, we introduce the model to the new data from the newly arrived students. So, the model will be able to predict depending on the side of the margin the student falls if he or she would pass or not.

1.6.3 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` and `sklearn.metrics.make_scorer`. - 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 F1 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 [19]:
```
# TODO: Import 'gridSearchCV' and 'make_scorer'
from sklearn import svm, grid_search, datasets
from sklearn.metrics import make_scorer
from sklearn.svm import SVC
# TODO: Create the parameters list you wish to tune
parameters = [{'C': [1, 10, 1000], 'kernel': ['linear']},
  {'C': [1, 10, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']}]
# TODO: Initialize the classifier
clf = SVC()

# 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 method
grid_obj = grid_search.GridSearchCV(clf,param_grid = 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_t
print "Tuned model has a testing F1 score of {:.4f}.".format(predict_labels(clf, X_test, y_tes
```

```
Made predictions in 0.0000 seconds.
Tuned model has a training F1 score of 0.8423.
Made predictions in 0.0000 seconds.
Tuned model has a testing F1 score of 0.7838.
```

1.6.4 Question 5 - Final F1 Score

What is the final model's F1 score for training and testing? How does that score compare to the untuned model?

Answer: The final model's F1 score for training set is 0.8423 and for testing set is 0.7838 which is better as compared to the mean of the F1 score for the testing set(0.7799) and training set(0.8739) on untuned model.Therefore, tuning improved the F1 score, but by only 0.008 for testing set and 0.06 for training set. This small improvement is a little disappointing, but may indicate that the default parameters are well chosen.

> **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.