student_intervention

February 11, 2016

1 Project 2: Supervised Learning

1.0.1 Building a Student Intervention System

1.1 1. Classification vs Regression

Your goal is to identify students who might need early intervention - which type of supervised machine learning problem is this, classification or regression? Why? A: Identifying students for early intervention is a classification problem. Students can be divided into two groups: students who passed and students who failed. Binary label of Success naturally lead to a classification problem. We could get the probability of Success, but even in that case one given student should be labeld into success or failure because the goal of this system is to decide to intervene.

1.2 2. Exploring the Data

Let's go ahead and read in the student dataset first.

To execute a code cell, click inside it and press **Shift+Enter**.

```
In [1]: # Import libraries
        import numpy as np
        import pandas as pd
```

```
In [2]: # Read student data
        student_data = pd.read_csv("student-data.csv")
        print "Student data read successfully!"
        # Note: The last column 'passed' is the target/label, all other are feature columns
```

```
Student data read successfully!
```

Now, can you find out the following facts about the dataset? - Total number of students - Number of students who passed - Number of students who failed - Graduation rate of the class (%) - Number of features

Use the code block below to compute these values. Instructions/steps are marked using **TODO**s.

```
In [3]: n_students = student_data.shape[0]
        n_features = student_data.shape[1]

        n_passed = student_data[student_data['passed']=='yes'].shape[0]
        n_failed = student_data[student_data['passed']=='no'].shape[0]
        grad_rate = float(n_passed*100)/(n_passed+n_failed)
        print "Total number of students: {}".format(n_students)
        print "Number of students who passed: {}".format(n_passed)
        print "Number of students who failed: {}".format(n_failed)
        print "Number of features: {}".format(n_features)
        print "Graduation rate of the class: {:.2f}%".format(grad_rate)
```

```
Total number of students: 395
Number of students who passed: 265
Number of students who failed: 130
Number of features: 31
Graduation rate of the class: 67.09%
```

1.3 3. Preparing the Data

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

1.3.1 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.

Let's first separate our data into feature and target columns, and see if any features are non-numeric.
Note: For this dataset, the last column (`passed`) is the target or label we are trying to predict.

```
In [4]: # Extract feature (X) and target (y) columns
        feature_cols = list(student_data.columns[:-1])  # all columns but last are features
        target_col = student_data.columns[-1]  # last column is the target/label
        print "Feature column(s):-\n{}".format(feature_cols)
        print "Target column: {}".format(target_col)

        X_all = student_data[feature_cols]  # feature values for all students
        y_all = student_data[target_col]  # corresponding targets/labels
        print "\nFeature values:-"
        print X_all.head()  # print the first 5 rows
```

```
Feature column(s):-
['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu', 'Mjob', 'Fjob', 'reason', 'gu
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]
```

1.3.2 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.

```
In [7]: # Preprocess feature columns
        def preprocess_features(X):
            outX = pd.DataFrame(index=X.index)  # output dataframe, initially empty

            # Check each column
            for col, col_data in X.iteritems():
                # If data type is non-numeric, try to replace all yes/no values with 1/0
                if col_data.dtype == object:
                    col_data = col_data.replace(['yes', 'no'], [1, 0])
                # Note: This should change the data type for yes/no columns to int

                # If still non-numeric, convert to one or more dummy variables
                if col_data.dtype == object:
                    col_data = pd.get_dummies(col_data, prefix=col)  # e.g. 'school' => 'school_GP', 's

                outX = outX.join(col_data)  # collect column(s) in output dataframe

            return outX

        X_all = preprocess_features(X_all)
        print "Processed feature columns ({}):-\n{}".format(len(X_all.columns), list(X_all.columns))
```

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

1.3.3 Split data into training and test sets

So far, we have converted all categorical features into numeric values. In this next step, we split the data (both features and corresponding labels) into training and test sets.

```
In [8]: # First, decide how many training vs test samples you want
        num_all = student_data.shape[0]  # same as len(student_data)
        num_train = 300  # about 75% of the data
        num_test = num_all - num_train

        import random
        train_random_index = random.sample(range(0,num_all), num_train)
        test_random_index = set(range(0,num_all))-set(train_random_index)
        # Then, select features (X) and corresponding labels (y) for the training and test sets
        # Note: Shuffle the data or randomly select samples to avoid any bias due to ordering in the da
        X_train = X_all.ix[train_random_index]
        y_train = y_all.ix[train_random_index]
        X_test = X_all.ix[test_random_index]
        y_test = y_all.ix[test_random_index]
        print "Training set: {} samples".format(X_train.shape[0])
```

```
print "Test set: {} samples".format(X_test.shape[0])
# Note: If you need a validation set, extract it from within training data
```

```
Training set: 300 samples
Test set: 95 samples
```

1.4 4. Training and Evaluating Models

Choose 3 supervised learning models that are available in scikit-learn, and appropriate for this problem. For each model:

- What are the general applications of this model? What are its strengths and weaknesses? (Q1)
- Given what you know about the data so far, why did you choose this model to apply? (Q2)
- Fit this model to the training data, try to predict labels (for both training and test sets), and measure the F1 score. Repeat this process with different training set sizes (100, 200, 300), keeping test set constant.

Produce a table showing training time, prediction time, F1 score on training set and F1 score on test set, for each training set size.

1. Logistic Regression
   - (A1)This model is mainly utilized for binary classification. Logistic Regression is simple but powerful enough to train simple data and yield reasonable performances. By using this model, we can easily understand the effects of predictors under controlling the effects of other variables. Due to the its simplicity, it takes relatively short time to train underlying relationships of data. On the other hand, its simplicity can give high bias when data is too complex to be trained by Logistic Regression.
   - (A2)I aimed to apply this model because logistic regression works well for data of simplicity. Complex models can have high variance when the data is simple enough to be modeled by simpler model such as Logistic Regression.
2. SVM with rbf kernel
   - (A1)SVM is mainly used for classification problems and it also works for regression problems. It has a powerful trick called kernel. By using different types of kernel functions, we can model data of different shapes. On the contrary, SVM has a weakness that it is too slow to train underlying data. Also, SVM has low explanatory power than simple algorithms such as Logistic Regression. That is, it is hard to understand what is going on under SVM.
   - (A2)I chose this model to train complex data by using kernel tricks. RBF kernel helps to understand nonlinear relationships and it would show better performances than simpler models (e.g., Logistic Regression) for complex data.
3. Adaboost
   - (A1)Adaboost is one sort of ensemble method that combines multiple weak classifiers. It is utilized for classification problems. It shows great performances for real world problems. By combining multiple weak classifiers (i.e., Decision Tree), this model shows good performances with preventing for overfitting. On the other hand, Adaboost is sensitive to outliers because it uses weak classifier of high biases. Nevertheless, because Adaboost combines multiple weak classifiers, the combined model tends to be a strong classifier of low bias. Another weakness is the computing time. Because this model combines multiple classifiers, it generally takes longer time than classifiers using a single model.
   - (A2)I chose Adaboost because this model is known to show the best performances among out-of-the-box classifiers.

Note: You need to produce 3 such tables - one for each model.

In [9]:
```
# Train a model
import time

def train_classifier(clf, X_train, y_train):
    print "Training {}...".format(clf.__class__.__name__)
    start = time.time()
    clf.fit(X_train, y_train)
    end = time.time()
    print "Done!\nTraining time (secs): {:.3f}".format(end - start)

# Choose a model, import it and instantiate an object
from sklearn import linear_model
clf = linear_model.LogisticRegression()

# Fit model to training data
train_classifier(clf, X_train, y_train)  # note: using entire training set here
print clf  # you can inspect the learned model by printing it
```

```
Training LogisticRegression...
Done!
Training time (secs): 0.006
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          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 [10]:
```
# Predict on training set and compute F1 score
from sklearn.metrics import f1_score

def predict_labels(clf, features, target):
    print "Predicting labels using {}...".format(clf.__class__.__name__)
    start = time.time()
    y_pred = clf.predict(features)
    end = time.time()
    print "Done!\nPrediction time (secs): {:.3f}".format(end - start)
    return f1_score(target.values, y_pred, pos_label='yes')

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

```
Predicting labels using LogisticRegression...
Done!
Prediction time (secs): 0.002
F1 score for training set: 0.820512820513
```

In [11]:
```
# Predict on test data
print "F1 score for test set: {}".format(predict_labels(clf, X_test, y_test))
```

```
Predicting labels using LogisticRegression...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.810810810811
```

In [12]:
```
# Train and predict using different training set sizes
def train_predict(clf, X_train, y_train, X_test, y_test):
```

```
            print "------------------------------------------"
            print "Training set size: {}".format(len(X_train))
            train_classifier(clf, X_train, y_train)
            print "F1 score for training set: {}".format(predict_labels(clf, X_train, y_train))
            print "F1 score for test set: {}".format(predict_labels(clf, X_test, y_test))
            print "------------------------------------------"

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

```
------------------------------------------
Training set size: 300
Training LogisticRegression...
Done!
Training time (secs): 0.003
Predicting labels using LogisticRegression...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.820512820513
Predicting labels using LogisticRegression...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.810810810811
------------------------------------------
```

```
In [19]: # Train and predict using two other models
         from sklearn import svm
         print 'SVC using rbf kernel'
         clf_svm_rbf = svm.SVC(kernel='rbf')
         train_predict(clf_svm_rbf, X_train, y_train, X_test, y_test)

         from sklearn import ensemble
         clf_adaboost = ensemble.AdaBoostClassifier()
         print 'AdaBoost'
         train_predict(clf_adaboost, X_train, y_train, X_test, y_test)
```

```
SVC using rbf kernel
------------------------------------------
Training set size: 300
Training SVC...
Done!
Training time (secs): 0.007
Predicting labels using SVC...
Done!
Prediction time (secs): 0.004
F1 score for training set: 0.866666666667
Predicting labels using SVC...
Done!
Prediction time (secs): 0.002
F1 score for test set: 0.825806451613
------------------------------------------
AdaBoost
------------------------------------------
Training set size: 300
```

```
Training AdaBoostClassifier...
Done!
Training time (secs): 0.083
Predicting labels using AdaBoostClassifier...
Done!
Prediction time (secs): 0.007
F1 score for training set: 0.863309352518
Predicting labels using AdaBoostClassifier...
Done!
Prediction time (secs): 0.005
F1 score for test set: 0.755244755245
----------------------------------------
```

1.5 5. Choosing the Best Model

- Based on the experiments you performed earlier, in 1-2 paragraphs explain to the board of supervisors what single model you chose as the best model. Which model is generally the most appropriate based on the available data, limited resources, cost, and performance?
  - I would choose SVM with the rbf kernel by considering the available data, limited resources, cost, and performance. Among three candidates that I chose (Logistic Regression, SVM, AdaBoost), Logistic Regression is another strong candidate. In terms of computation time and cost, this simple model takes shorter time with a smaller amount of computing cost. It performed quite good with the moderate level of scores (0.8205 for the training set and 0.8108 for the test set) to identify students who needs interventions. It is even better than AdaBoost, which is known to be very effective in practical problems. Despite all of this, I chose SVM because the most important criteria to decide a model is performance. In other words, SVM is the best algorithm to identify students to be intervened among those candidates.
- In 1-2 paragraphs explain to the board of supervisors in layman's terms how the final model chosen is supposed to work (for example if you chose a Decision Tree or Support Vector Machine, how does it make a prediction).
  - Let us assume that our data points are distributed on the two dimensional space. We can think of each point represents each student. Each student has a label on whether each student succeeds or not. SVM tries to find a line to separate two groups of points. When a new point are given to be classified, we can predict the class of a point by using the line. If the point falls into the area of success, the student is predicted to succeed. However, in real-world settings, it becomes more complex. We usually doesn't have linearly separable lines. To solve the problem, we project points into higher dimension and find hyperplanes to classify those points. This trick called the kernel trick and it is how SVM works.
- Fine-tune the model. Use Gridsearch with at least one important parameter tuned and with at least 3 settings. Use the entire training set for this.

By conducting GridSearchCV for SVM, the best model has a F1 score of 0.8481.

In [18]:
```
# Fine-tune your model and report the best F1 score
from sklearn import grid_search
from sklearn.metrics import f1_score
from sklearn.metrics import make_scorer


C_range = np.logspace(-2, 10, 13)
gamma_range = np.logspace(-9, 3, 13)
```

```
        parameters = dict(C=C_range, gamma=gamma_range)
        f1_scorer = make_scorer(f1_score, pos_label="yes")

        grid = grid_search.GridSearchCV(svm.SVC(kernel='rbf'), param_grid=parameters, cv=5, scoring=f1_
        grid.fit(X_train, y_train)

        print("The best parameters are %s with a score of %0.2f"
              % (grid.best_params_, grid.best_score_))

        svm_best = svm.SVC(kernel='rbf', C=grid.best_params_['C'],
                                         gamma=grid.best_params_['gamma'])
        train_predict(svm_best, X_train, y_train, X_test, y_test)
```

```
The best parameters are {'C': 100.0, 'gamma': 0.0001} with a score of 0.81
----------------------------------------
Training set size: 300
Training SVC...
Done!
Training time (secs): 0.006
Predicting labels using SVC...
Done!
Prediction time (secs): 0.004
F1 score for training set: 0.814655172414
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.848101265823
----------------------------------------
```

```
In [ ]:
```