student_intervention

June 9, 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?

This is a Classification problem because whether or not a student needs intervention is a binary yes or no answer and is not a continous value.

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
```

```
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]: # TODO: Compute desired values - replace each '?' with an appropriate expre
        n_students = student_data.shape[0]
        n_features = student_data.shape[1] - 1
        n_passed = student_data.query("passed == 'yes'").shape[0]
        n_failed = student_data.query("passed == 'no'").shape[0]
        grad_rate = 100*(float(n_passed)/float(n_students))
```

```
        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: 30
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
        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', '
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 [5]: # 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 wit
                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. 'schoo

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

            return outX

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

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

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 [6]: # 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
        # TODO: Then, select features (X) and corresponding labels (y) for the trai
        # Note: Shuffle the data or randomly select samples to avoid any bias due t
        #permutation = np.random.permutation(num_all)
        #student_data.reindex(permutation)
        #y_all.reindex(permutation)

        from sklearn.cross_validation import train_test_split
        X_train_svm, X_test, y_train_svm, y_test = train_test_split(
             X_all, y_all, train_size=300, test_size=95, random_state=42)

        #X_train = X_all.ix[0:num_train-1]
        #y_train = y_all.ix[0:num_train-1]
        #X_test  = X_all.ix[num_train:num_all]
        #y_test  = y_all.ix[num_train:num_all]
        print "Training set: {} samples".format(X_train_svm.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 is the theoretical O(n) time & space complexity in terms of input size?

  Decision Tree: nsamples nfeatures log nsamples (time) and nfeatures (space)

  SVM: nfeatures x nasamples (a is 2 or 3 depending on cache) (time) and nfeatures (space)

  kNN: The time complexity depends on the choice of algorithm: D*N (Brute Force), D*log(N) (Ball Tree), and D*N (worst case)(K-D Tree). The space complexity depends on the intrinsic dimensionality of the original data.

- What are the general applications of this model? What are its strengths and weaknesses?

  These models are generally used for classification problems. Decision Trees can make strong high-level splits in the data. They can extract the features that provide the highest level information gain and split labels based on those. SVM models lack this notion of information gain and instead rely on the location of the feature vector in a higher dimensional space. But if a wide enough margin can be found to exist, this lack of information gain is irrelevant. Furthermore, SVM models can capture non-linear relationships, whereas Decision Trees don't. kNN models rely on the assumed proximity of data points that reside in the same class. It can be very successful in cases in which an irregular decision boundary must be defined. The primary weakness of this model is that in the event of ties (a given data point having an equally high number of neighbors in each set), the outcome is not well defined. This means that the outcome can depend on the ordering of your datasets rather than any legitimate metric.

- Given what you know about the data so far, why did you choose this model to apply?

  The choice to apply a Decision Tree model was fairly straightforward. If you were someone in education you could probably deduce from a serious of simple questions (while also referencing your experience) and make an educated guess yourself. This is of course what Decision Tree's do. But they can do it on a much more fine-grain scale.

  For the SVM that I applied I wanted to try an alternative to a decision tree that would perhaps capture in finer detail the non-linear relationships in the data (if any exists).

  Finally, for kNN I thought that if the data samples are distinct enough (i.e. the students being modeled fit well into distinct social groups based on outcome), then a kNN approach would probably be able to derive this seperation.

- 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.

  The tables below are for Decision Tree, SVM, and kNN, respectively.

Training Set Size
300
200
100

Training Set Size
300
200
100

Training Set Size
300
200
100

In [7]:
```
# 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)

# TODO: Choose a model, import it and instantiate an object
from sklearn import svm
clf = svm.SVC()

# Fit model to training data
train_classifier(clf, X_train_svm, y_train_svm)  # note: using entire trai
print clf  # you can inspect the learned model by printing it
```

```
Training SVC...
Done!
Training time (secs): 0.013
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [8]:
```
# 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_svm, y_train_svm)
print "F1 score for training set: {}".format(train_f1_score)
```

```
Predicting labels using SVC...
Done!
```

```
Prediction time (secs): 0.008
F1 score for training set: 0.876068376068
```

```
In [9]: # Predict on test data
        print "F1 score for test set: {}".format(predict_labels(clf, X_test, y_test
```

```
Predicting labels using SVC...
Done!
Prediction time (secs): 0.004
F1 score for test set: 0.783783783784
```

```
In [10]: # 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_tra
             print "F1 score for test set: {}".format(predict_labels(clf, X_test, y

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

         X_train_svm, _, y_train_svm, _ = train_test_split(
              X_all, y_all, train_size=200, test_size=95, random_state=42)
         train_predict(clf, X_train_svm, y_train_svm, X_test, y_test)

         X_train_svm, _, y_train_svm, _ = train_test_split(
              X_all, y_all, train_size=100, test_size=95, random_state=42)
         train_predict(clf, X_train_svm, y_train_svm, X_test, y_test)
```

```
------------------------------------------
Training set size: 300
Training SVC...
Done!
Training time (secs): 0.011
Predicting labels using SVC...
Done!
Prediction time (secs): 0.007
F1 score for training set: 0.876068376068
Predicting labels using SVC...
Done!
Prediction time (secs): 0.003
F1 score for test set: 0.783783783784
------------------------------------------
Training set size: 200
Training SVC...
```

```
Done!
Training time (secs): 0.006
Predicting labels using SVC...
Done!
Prediction time (secs): 0.004
F1 score for training set: 0.867924528302
Predicting labels using SVC...
Done!
Prediction time (secs): 0.002
F1 score for test set: 0.781456953642
------------------------------------------
Training set size: 100
Training SVC...
Done!
Training time (secs): 0.002
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.877697841727
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.774647887324
```

In [11]:
```
# TODO: Train and predict using two other models
from sklearn import tree
clf = tree.DecisionTreeClassifier(max_depth=5)

X_train_dt, _, y_train_dt, _ = train_test_split(
     X_all, y_all, train_size=300, test_size=95, random_state=42)
train_predict(clf, X_train_dt, y_train_dt, X_test, y_test)

X_train_dt, _, y_train_dt, _ = train_test_split(
     X_all, y_all, train_size=200, test_size=95, random_state=42)
train_predict(clf, X_train_dt, y_train_dt, X_test, y_test)

X_train_dt, _, y_train_dt, _ = train_test_split(
     X_all, y_all, train_size=100, test_size=95, random_state=42)
train_predict(clf, X_train_dt, y_train_dt, X_test, y_test)


from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)

X_train_knn, _, y_train_knn, _ = train_test_split(
     X_all, y_all, train_size=300, test_size=95, random_state=42)
train_predict(clf, X_train_knn, y_train_knn, X_test, y_test)
```

```
        X_train_knn, _, y_train_knn, _ = train_test_split(
             X_all, y_all, train_size=200, test_size=95, random_state=42)
        train_predict(clf, X_train_knn, y_train_knn, X_test, y_test)

        X_train_knn, _, y_train_knn, _ = train_test_split(
             X_all, y_all, train_size=100, test_size=95, random_state=42)
        train_predict(clf, X_train_knn, y_train_knn, X_test, y_test)
```

```
------------------------------------------
Training set size: 300
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.004
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.871194379391
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.763358778626
------------------------------------------
Training set size: 200
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.001
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.896321070234
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.744525547445
------------------------------------------
Training set size: 100
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.001
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.887218045113
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.755244755245
```

```
------------------------------------------
Training set size: 300
Training KNeighborsClassifier...
Done!
Training time (secs): 0.001
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.007
F1 score for training set: 0.880733944954
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.003
F1 score for test set: 0.731343283582
------------------------------------------
Training set size: 200
Training KNeighborsClassifier...
Done!
Training time (secs): 0.001
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.004
F1 score for training set: 0.896551724138
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.002
F1 score for test set: 0.741258741259
------------------------------------------
Training set size: 100
Training KNeighborsClassifier...
Done!
Training time (secs): 0.001
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.832116788321
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.748201438849
```

2 5. Choosing the Best Model

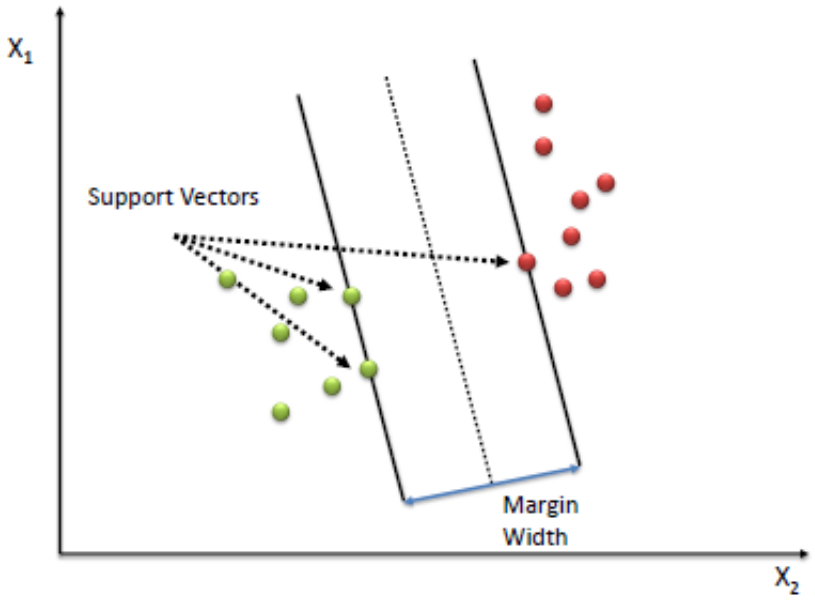
- Based on the experiments you performed earlier, in 1-2 paragraphs explain to the board of supervisors what single model you chose as the best model. Which model is generally the most appropriate based on the available data, limited resources, cost, and performance?

  The best model for predicting whether a student needs intervention on behalf of the

district was found to be a SVM based classifier. Based on some preliminary experimentation this simple classifier produced the highest F1 score on the available test set. The best F1 score achieved using this simple classifier was roughly 0.78 (on the test set with 300 training examples). The use of an SVM model (as opposed to a decision tree based model) seems to be the most useful approach in this context because it can capture some of the more non-linear aspects of the data. The Decision Tree tries to make "simple" linear splits in the data which nearly works but not quite as well (as can be seen by the lower F1 scores on the test set).

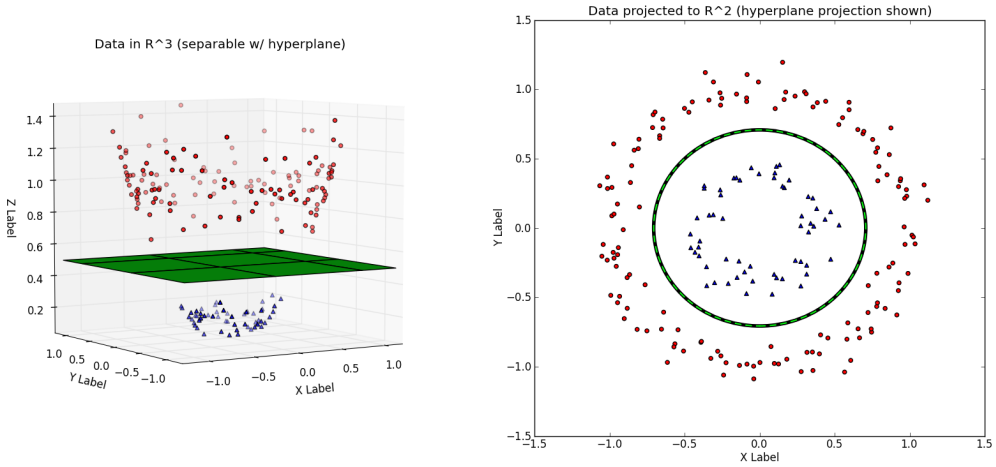
- In 1-2 paragraphs explain to the board of supervisors in layman's terms how the final model chosen is supposed to work (for example if you chose a Decision Tree or Support Vector Machine, how does it make a prediction).

  The Support Vector Machine makes it's decision by taking all of the features and putting them into a higher dimensional space. It looks at all the features and labels and tries to seperate the positive examples from the negative examples by putting a hyperplane between them. In the 2D case this would simply be a line seperating the data as shown in the following figure. In this case the data is said to be linearly seperable (in 2 dimensions):



title

In some cases you must go to a higher dimension to successfully seperate the data (if you have more than two features, for example). In the case shown below, with the addition of a 3rd feature you can go into a higher dimensional space and seperate the data successfully (this is a case in which the points are said to be not linearly seperable in 2 dimensions):



title

A hyperplane is a generalization of such a plane to more than 3 dimensions and is not shown here.

It's reliability is further enhanced by the fact that the algorithm attempts to create a plane with as much seperation as possible between itself and the positive and negative examples (as illustrated in the image for the 2D case above. These types of models can be made even more accurate as more data samples are gathered. Only 300 training examples were used in this model. These models are particularly suseptible to the curse of dimensionality which means that as the number of features grows you need exponentially more samples so that the algorithm can see enough examples of the data to create as accurate a model as possible. But given the amount of data we had available and the fact that we achieved an approximaetly 80% F1 score, i'd say we have a 'sufficient' amount of sample points to build an acceptable model. But there might be room for further improvement.

- 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.

  Shown below…

- What is the model's final F1 score?

  79.5

```
In [13]: # TODO: Fine-tune your model and report the best F1 score
         from sklearn import svm, grid_search
         from sklearn.metrics import f1_score
         from sklearn.metrics import make_scorer

         f1_scorer = make_scorer(f1_score, pos_label="yes")

         tuned_parameters = [{'kernel': ['rbf','poly','sigmoid'],
                              'gamma': ['auto', 1e-3, 1e-4, 1e-5],
```

```
                                'C': [1, 10, 100, 1000],
                                'shrinking':[True, False]}]

        clf = svm.SVC()
        clf = grid_search.GridSearchCV(clf, tuned_parameters, scoring=f1_scorer)
        X_train_svm, X_test, y_train_svm, y_test = train_test_split(
             X_all, y_all, train_size=300, test_size=95, random_state=42)
        train_predict(clf, X_train_svm, y_train_svm, X_test, y_test)
```

```
------------------------------------------
Training set size: 300
Training GridSearchCV...
Done!
Training time (secs): 3.531
Predicting labels using GridSearchCV...
Done!
Prediction time (secs): 0.006
F1 score for training set: 0.832298136646
Predicting labels using GridSearchCV...
Done!
Prediction time (secs): 0.002
F1 score for test set: 0.794520547945
```

```
In [ ]:
```