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:
The Question belongs to the classification problem because the target of the prediction is the label、type of the new observation except of the continuous values of the explanation_values. 因为预测目标并非针对 解释变量的连续数值上的预测，而是对新观测值的类型，标签进行预测。

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 [58]:
```python
# 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!
```

In [59]:
```python
student_data.head()
```

Out[59]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	internet	romantic	famrel	freetime	goout	Dalc	Walc	health	absences	passed
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	no	no	4	3	4	1	1	3	6	no
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	yes	no	5	3	3	1	1	3	4	no
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	yes	no	4	3	2	2	3	3	10	yes
3	GP	F	15	U	GT3	T	4	2	health	services	...	yes	yes	3	2	2	1	1	5	2	yes
4	GP	F	16	U	GT3	T	3	3	other	other	...	no	no	4	3	2	1	2	5	4	yes

5 rows × 31 columns

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 [60]:
```
n_passed = student_data[student_data['passed'] == 'yes']
```

In [61]:
```
len(n_passed)
```

Out[61]: 265

In [62]:
```
# TODO: Calculate number of students
n_students = len(student_data.index.values)

# TODO: Calculate number of features
n_features = len(student_data.columns.values) - 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 [63]:
```
# 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, respe
ctively)
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', 'absences'
]

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()` function to perform this transformation. Run the code cell below to perform the preprocessing routine discussed in this section.

In [64]:
```
def preprocess_features(X):
    ''' Preprocesses the student data and converts non-numeric binary variabl
es into
        binary (0/1) variables. Converts categorical variables into dummy var
iables. '''

    # 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.c
olumns), 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', 'Mjob
_at_home', 'Mjob_health', 'Mjob_other', 'Mjob_services', 'Mjob_teacher', 'Fjo
b_at_home', 'Fjob_health', 'Fjob_other', 'Fjob_services', 'Fjob_teacher', 're
ason_course', 'reason_home', 'reason_other', 'reason_reputation', 'guardian_f
ather', 'guardian_mother', 'guardian_other', 'traveltime', 'studytime', 'fail
ures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'int
ernet', '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 [65]:
```
X_all[:10]
```

Out[65]:

	school_GP	school_MS	sex_F	sex_M	age	address_R	address_U	famsize_GT3	famsize_LE3	Pstatus_A	...	higher	internet	romantic	famrel	freetime	goout	Dalc	Walc	health	absences
0	1.0	0.0	1.0	0.0	18	0.0	1.0	1.0	0.0	1.0	...	1	0	0	4	3	4	1	1	3	6
1	1.0	0.0	1.0	0.0	17	0.0	1.0	1.0	0.0	0.0	...	1	1	0	5	3	3	1	1	3	4
2	1.0	0.0	1.0	0.0	15	0.0	1.0	0.0	1.0	0.0	...	1	1	0	4	3	2	2	3	3	10
3	1.0	0.0	1.0	0.0	15	0.0	1.0	1.0	0.0	0.0	...	1	1	1	3	2	2	1	1	5	2
4	1.0	0.0	1.0	0.0	16	0.0	1.0	1.0	0.0	0.0	...	1	0	0	4	3	2	1	2	5	4
5	1.0	0.0	0.0	1.0	16	0.0	1.0	0.0	1.0	0.0	...	1	1	0	5	4	2	1	2	5	10
6	1.0	0.0	0.0	1.0	16	0.0	1.0	0.0	1.0	0.0	...	1	1	0	4	4	4	1	1	3	0
7	1.0	0.0	1.0	0.0	17	0.0	1.0	1.0	0.0	1.0	...	1	0	0	4	1	4	1	1	1	6
8	1.0	0.0	0.0	1.0	15	0.0	1.0	0.0	1.0	1.0	...	1	1	0	4	2	2	1	1	1	0
9	1.0	0.0	0.0	1.0	15	0.0	1.0	1.0	0.0	0.0	...	1	1	0	5	5	1	1	1	5	0

10 rows × 48 columns

In [66]:

```
from sklearn.cross_validation import train_test_split,ShuffleSplit
cv_sets = ShuffleSplit(X_all.shape[0], n_iter = 10, test_size = 0.20, random_
state = 0)
```

In [67]:

```
# TODO: Import any additional functionality you may need here
import numpy as np
import pandas as pd
from sklearn.cross_validation import train_test_split,StratifiedShuffleSplit


#make a function to reuse to split the train and test data.
def shuffle_data(X, y, n_iters, num_train, num_random):
    cv_sets = ShuffleSplit(X.shape[0], n_iter = n_iters, train_size = num_tra
in,random_state = num_random)

    for train_index,test_index in cv_sets:
        X_train,X_test = X.ix[train_index], X.ix[test_index]
        y_train,y_test = y.ix[train_index], y.ix[test_index]

    return X_train,X_test,y_train,y_test


# def stratified_shuffle_data(X, y, n_iters, num_train, num_random):
#  stratifiedshufflesplit is not better than shufflesplit in the case compari
ng the f1_score.

#     cv_sets = StratifiedShuffleSplit(y, n_iter = n_iters, train_size = num_
train,random_state = num_random)

#     for train_index,test_index in cv_sets:
#         X_train,X_test = X.ix[train_index], X.ix[test_index]
#         y_train,y_test = y.ix[train_index], y.ix[test_index]

#     return X_train,X_test,y_train,y_test


X_train,X_test,y_train,y_test = shuffle_data(X_all,y_all,10,300,5)

# 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 40 samples.
```

Training and Evaluating Models

In this section, you will choose 3 supervised learning models that are appropriate for this problem and available in `scikit-learn`. You will first discuss the reasoning behind choosing these three models by considering what you know about the data and each model's strengths and weaknesses. You will then fit the model to varying sizes of training data (100 data points, 200 data points, and 300 data points) and measure the F_1 score. You will need to produce three tables (one for each model) that shows the training set size, training time, prediction time, F_1 score on the training set, and F_1 score on the testing set.

Question 2 - Model Application

List three supervised learning models that are appropriate for this problem. What are the general applications of each model? What are their strengths and weaknesses? Given what you know about the data, why did you choose these models to be applied?

Answer:

1. LogisticRegression( )

Strengths: Generalized linear regression The predicted results are bounded between 0 and 1 probability Due to this case including only two results in the label,LR would be certain compatible as a solution.

Weakness: Logistic Regression is not very well compatible with complex relationships. LR is more sensitive to the multiple co-linearity of the dependent variables in the model .

2. RandomForestClassifier(criterion='entropy')

Strengths:

Using the maximun information gain algorithm and ensemble random dicision tress ,RF has little noise from train set to advoid the overfitting problem. Meanwile, in the sample collection,RF apply a balance of performance-independency via 'Sample Jitter' & ‘Attribution Jitter’.

Weakness: Its run_time performance is not good enough as a descriptive tool.Comparing to a single decision tree, the results of learning are incomprehensive.And also,they are hard to make incremental comparing with the naive bayes algorithm.In addtion,there is a certain degree of strong dependence between learners in this case.RF has a less applicability than AdaBoost in the case.

3. AdaBoostClassifier( ) :

However, Its hard to advoid the independency between the samples. According to the attributions in this case , income ,careers,location and other attributions share a certain positive correlation somehow.

Strengths: The learners are training from these samples ,so that there is some degree of strong dependence between individual learners in this case.So AdaBoostClassifier can be more applicability to predict the result than RF.

Weakness: On above,AdaBoostClassifier is less compability with the independency learners in some case,comparing to RandomForest Classifier.

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 [68]:
```
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.__c
lass__.__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_t
rain, 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 [69]:
```
# TODO: Import the three supervised learning models from sklearn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.cross_validation import train_test_split,ShuffleSplit,cross_val_
score

from sklearn.linear_model.logistic import LogisticRegression # A
from sklearn.ensemble import RandomForestClassifier # B
from sklearn.ensemble import AdaBoostClassifier  # C



# TODO: Initialize the three models

clf_A = LogisticRegression(random_state = 0)
clf_B = RandomForestClassifier(max_depth = 3,random_state = 0)
clf_C = AdaBoostClassifier(n_estimators= 5,learning_rate = 0.5,random_state =
0)

# TODO: Set up the training set sizes
# def shuffle_data(X, y, n_iters, num_train, num_random):

X_train_100,X_test_100,y_train_100,y_test_100 = shuffle_data(X_all,y_all,10,1
00,5)

X_train_200,X_test_200,y_train_200,y_test_200 = shuffle_data(X_all,y_all,10,2
00,5)

X_train_300,X_test_300,y_train_300,y_test_300 = shuffle_data(X_all,y_all,10,3
00,5)


# TODO: Execute the 'train_predict' function for each classifier and each tra
ining set size
# train_predict(clf_A, X_train, y_train, X_test, y_test)
# train_predict(clf_B, X_train, y_train, X_test, y_test)
# train_predict(clf_C, X_train, y_train, X_test, y_test)


#clf_A:
print('###1.  LogisticRegression: Train_size = 100 \n')
train_predict(clf_A, X_train_100, y_train_100, X_test_100, y_test_100)
print(' \n')
print('###1.  LogisticRegression: Train_size = 200 \n')
train_predict(clf_A, X_train_200, y_train_200, X_test_200, y_test_200)
print(' \n')
print('###1.  LogisticRegression: Train_size = 300 \n')
train_predict(clf_A, X_train_300, y_train_300, X_test_300, y_test_300)
print(' \n')
print(' \n')


#clf_B:
print('###2.  RandomForestClassifier: Train_size = 100 \n')
train_predict(clf_B, X_train_100, y_train_100, X_test_100, y_test_100)
print(' \n')
print('###2.  RandomForestClassifier: Train_size = 200 \n')
train_predict(clf_B, X_train_200, y_train_200, X_test_200, y_test_200)
print(' \n')
print('###2.  RandomForestClassifier: Train_size = 300 \n')
train_predict(clf_B, X_train_300, y_train_300, X_test_300, y_test_300)
print(' \n')
print(' \n')


#clf_C:
print('###3.  AdaBoostClassifier: Train_size = 100 \n')
train_predict(clf_C, X_train_100, y_train_100, X_test_100, y_test_100)
print(' \n')
print('###3.  AdaBoostClassifier: Train_size = 200 \n')
train_predict(clf_C, X_train_200, y_train_200, X_test_200, y_test_200)
print(' \n')
print('###3.  AdaBoostClassifier: Train_size = 300 \n')
train_predict(clf_C, X_train_300, y_train_300, X_test_300, y_test_300)
```

```
print(' \n')
print(' \n')




# plt.title('RandomForestClassifier:')
# plt.xlabel('train_size')
# plt.ylabel('F1 Score for test size')
# f1_score_100 = predict_labels(clf_C, X_test_100, y_test_100)
# f1_score_200 = predict_labels(clf_C, X_test_200, y_test_200)
# f1_score_300 = predict_labels(clf_C, X_test_300, y_test_300)
# f1_score =[f1_score_100,f1_score_200,f1_score_300]
# train_set = [100,200,300]
# plt.plot(f1_score, train_set,'k-')
```

```
###1.  LogisticRegression: Train_size = 100

Training a LogisticRegression using a training set size of 100. . .
Trained model in 0.0029 seconds
Made predictions in 0.0003 seconds.
F1 score for training set: 0.8784.
Made predictions in 0.0003 seconds.
F1 score for test set: 0.8065.


###1.  LogisticRegression: Train_size = 200

Training a LogisticRegression using a training set size of 200. . .
Trained model in 0.0023 seconds
Made predictions in 0.0003 seconds.
F1 score for training set: 0.8750.
Made predictions in 0.0002 seconds.
F1 score for test set: 0.7667.


###1.  LogisticRegression: Train_size = 300

Training a LogisticRegression using a training set size of 300. . .
Trained model in 0.0028 seconds
Made predictions in 0.0002 seconds.
F1 score for training set: 0.8249.
Made predictions in 0.0002 seconds.
F1 score for test set: 0.8615.




###2.  RandomForestClassifier: Train_size = 100

Training a RandomForestClassifier using a training set size of 100. . .
Trained model in 0.0303 seconds
Made predictions in 0.0016 seconds.
F1 score for training set: 0.8519.
Made predictions in 0.0014 seconds.
F1 score for test set: 0.8571.


###2.  RandomForestClassifier: Train_size = 200

Training a RandomForestClassifier using a training set size of 200. . .
Trained model in 0.0341 seconds
Made predictions in 0.0017 seconds.
F1 score for training set: 0.8282.
Made predictions in 0.0011 seconds.
F1 score for test set: 0.8406.


###2.  RandomForestClassifier: Train_size = 300

Training a RandomForestClassifier using a training set size of 300. . .
Trained model in 0.0301 seconds
Made predictions in 0.0013 seconds.
F1 score for training set: 0.8225.
Made predictions in 0.0011 seconds.
F1 score for test set: 0.8571.




###3.  AdaBoostClassifier: Train_size = 100

Training a AdaBoostClassifier using a training set size of 100. . .
Trained model in 0.0132 seconds
Made predictions in 0.0008 seconds.
F1 score for training set: 0.8667.
Made predictions in 0.0008 seconds.
F1 score for test set: 0.8657.


###3.  AdaBoostClassifier: Train_size = 200
```

```
Training a AdaBoostClassifier using a training set size of 200. . .
Trained model in 0.0135 seconds
Made predictions in 0.0009 seconds.
F1 score for training set: 0.8452.
Made predictions in 0.0008 seconds.
F1 score for test set: 0.8696.


###3.  AdaBoostClassifier: Train_size = 300

Training a AdaBoostClassifier using a training set size of 300. . .
Trained model in 0.0142 seconds
Made predictions in 0.0010 seconds.
F1 score for training set: 0.8286.
Made predictions in 0.0008 seconds.
F1 score for test set: 0.8824.
```

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.

Random_state = 0:

Classifer 1 - LogisticRegression

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0013	0.0002	0.8980	0.5957
200	0.0018	0.0002	0.8443	0.7308
300	0.0028	0.0002	0.8456	0.7778

Classifer 2 - RandomForestClassifier

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0254	0.0011	0.8625	0.7813
200	0.0268	0.0011	0.8232	0.7619
300	0.0330	0.0011	0.8340	0.7813

Classifer 3 - AdaBoostClassifier

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0177	0.0014	0.8608	0.8136
200	0.0215	0.0013	0.8452	0.8000
300	0.0215	0.0013	0.8408	0.8136

Random_state = 5:

Classifer 1 - LogisticRegression

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0016	0.0002	0.8784	0.8065
200	0.0017	0.0002	0.8750	0.7667

300	0.0025	0.0002	0.8249	0.8615

Classifer 2 - RandomForestClassifier

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0315	0.0015	0.8519	0.8571
200	0.0314	0.0011	0.8282	0.8406
300	0.0477	0.0016	0.8225	0.8571

Classifer 3 - AdaBoostClassifier

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0177	0.0008	0.8667	0.8657
200	0.0176	0.0013	0.8452	0.8696
300	0.0145	0.0014	0.8286	0.8824

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:

Training Time:

可以看到 逻辑回归的训练时间和预测耗费时间即耗费计算资源都是最低的，比后两者要第一个数量级，因为逻辑回归是单一学习器 逻辑回归的复杂度也是最低的。

AdaBoost分类器的训练时间 介于中间，RF训练时间最长。预测时间方面，AdaBoost分类器预测时间要长于RF。 原因在于： AdaBoost 在样本训练过程中 要根据基学习器表现情况 对样本分布进行调整，并且在学习器结合策略方面：属于加权结合。计算复杂度应该是三者中最高的。RF因为采样时候采取的是随机选取子集样本后的选择最优属性，复杂度要比AdaBoost 低。

F1_score ：

如上面的三个列表可见，表现***The best performance of f1_score is Adaboost Model***。

对比（逻辑回归） ，Adaboost由多个弱分类器组合，但是每个的权重alpha都是不一样的，会根据错误率来调整（低错误率则增加权重，反之则减少）最终 通过'投票' 线性加权组合 产生最终结果。会得到一个精度较高的结果并且不会产生过拟合现象。

对比 随机森林分类器，因为AdaBoost 更适用于 各个基学习器具有一定依赖性的环境 使用，而随机森林的 更适合在 各个基学习器相互独立的情况，相信是因为这一点，在这个各个参数有一定依赖性的案例下： F1_score上RF 得分低于 AdaBoost的原因。

Conclusion:

Adaboost classifier is the best model in this case . Because it make the best f1_score in the three. And also the training time it made between Logistic Regression and RandomForest which is acceptable.

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:

AdaBoostClassifier:

```
Based on Boosting algorithm,AdaBoostClassifier made the first base le
arner from the initial training set.
And then the training set distribution will be adjusted according to
the performance of the base learner.
The wrong prediction will accept more attention.
After the train set adjusted appropriately,AdaBoost will train the ne
xt base learner.
This process will repeate several times until the target numbers of t
he base learner have made.


The individual learners can be weak, but as long as the performance o
f each one is slightly better than random guessing (e.g., their error
rate is smaller than 0.5 for binary classification), the final model
can be proven to converge to a strong learner.(a democratic vote will
hold in theses base learners via a linear combination.)
```

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 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 [70]:

```
range(2,20,1)
```

Out[70]:

```
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

In [71]:

```
np.arange(0.1,0.9,0.1)
```

Out[71]:

```
array([ 0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8])
```

In [72]:

```
X_all = student_data.drop('passed',axis=1)
```

In [73]:

```
# TODO: Import 'GridSearchCV' and 'make_scorer'
import numpy as np
import pandas as pd

from sklearn.cross_validation import ShuffleSplit
from sklearn.grid_search import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
from sklearn.metrics import make_scorer

# Clean X_train,y_train
X_all = student_data.drop('passed',axis=1)
X_all = preprocess_features(X_all)
y_all= student_data['passed']

#Set the cv in a ShuffleSplit way
cv_sets = ShuffleSplit(X_all.shape[0], n_iter = 10, train_size = 300, random_state = 5)


#Initialize RF example
# clf = RandomForestClassifier( random_state = 0)
# parameters ={"n_estimators": range(2,10,1),'max_depth': np.arange(1,9,1)}


# TODO: Create the parameters list you wish to tune
parameters ={"n_estimators": range(2,20,1),'learning_rate': np.arange(0.1,0.9,0.1)}

# TODO: Initialize the classifier
clf = AdaBoostClassifier( random_state = 0)


# 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 = GridSearchCV(clf, parameters,cv = cv_sets, n_jobs = -1, verbose = 1, scoring =
f1_scorer )


# TODO: Fit the grid search object to the training data and find the optimal parameters
grid_obj = grid_obj.fit(X_all,y_all)

# Get the estimator
clf = grid_obj.best_estimator_

#print the best parameters
best_parameters = clf.get_params()
for param_name in sorted(parameters.keys()):
    print('{} : {}'.format(param_name, best_parameters[param_name]))

# Report the final F1 score for training and testing after parameter tuning
print "Tuned model has a training F1 score of {:.4f}.".format(predict_labels(clf, X_train,
y_train))
print "Tuned model has a testing F1 score of {:.4f}.".format(predict_labels(clf, X_test, y
_test))
```

```
Fitting 10 folds for each of 144 candidates, totalling 1440 fits
```

```
[Parallel(n_jobs=-1)]: Done 160 tasks      | elapsed:    1.8s
[Parallel(n_jobs=-1)]: Done 1060 tasks      | elapsed:   10.2s
```

```
learning_rate : 0.5
n_estimators : 3
Made predictions in 0.0009 seconds.
Tuned model has a training F1 score of 0.8212.
Made predictions in 0.0005 seconds.
Tuned model has a testing F1 score of 0.8657.
```

```
[Parallel(n_jobs=-1)]: Done 1440 out of 1440 | elapsed:   13.6s finished
```

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?

AdaBoostClassifier

manual params :

AdaBoostClassifier(n_estimators= 5,learning_rate = 0.5,random_state = 0)

ShuffleSplit(random_state = 0)

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0177	0.0014	0.8608	0.8136
200	0.0215	0.0013	0.8452	0.8000
300	0.0215	0.0013	0.8408	0.8136

AdaBoostClassifier

manual params :

AdaBoostClassifier(n_estimators= 5,learning_rate = 0.5,random_state = 0)

ShuffleSplit(random_state = 5)

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0177	0.0008	0.8667	0.8657
200	0.0176	0.0013	0.8452	0.8696
300	0.0145	0.0014	0.8286	0.8824

The final model's F1 score for training and testing:

AdaBoostClassifier

ShuffleSplit(random_state = 5)

GridSearchCV
parameters ={"n_estimators": range(2,20,1),'learning_rate': np.arange(0.1,0.9,0.1)}

Best_estimator.get_params( ):

```
learning_rate : 0.5
n_estimators : 3
```

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
300	0.0012	0.0005	0.8212	0.8657

Answer:

In Conclusion:

Due to the imbalance of the data set, we should adjust the random_state in ShuffleSplit to balance the imbalance.

As seen in the results of these graphs above,
random_state = 5 make the performance of f1_socre in either RandomForest Classifier nor AdaBoostClassifier get a better score than in the random_state = 0 .

finally,Adaboost is better than RandomForest in f1_score performance in this case.

> **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.