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:

I would say that this is a typical classification problem because labels(outcomes) are descrete. There is only two possible outcomes. Whether student needs intervention or not.

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 [9]:

```
# Import libraries
import pandas as pd
import numpy as np

# Read student data
student_data = pd.read_csv("student-data.csv")
print("Student data read successfully!")
```

```
Student data read successfully!
```

Implementation: Data Exploration

Let's begin by investigating the dataset to determine how many students we have information on, and learn about the graduation rate among these students. In the code cell below, you will need to compute the following:

- The total number of students, `n_students`.
- The total number of features for each student, `n_features`.
- The number of those students who passed, `n_passed`.
- The number of those students who failed, `n_failed`.
- The graduation rate of the class, `grad_rate`, in percent (%).

In [10]:

```
# 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 = len(student_data[student_data['passed'] == 'yes'].index)

# TODO: Calculate failing students
n_failed = len(student_data[student_data['passed'] == 'no'].index)

# TODO: Calculate graduation rate
grad_rate = n_passed/(n_passed + n_failed)

# 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: 0.67%
```

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 [11]:

```
# Extract feature columns
feature_cols = list(student_data.columns[:-1])

# Extract target column 'passed'
target_col = student_data.columns[-1]

# Show the list of columns
print("Feature columns:\n{}".format(feature_cols))
print("\nTarget column: {}".format(target_col))

# Separate the data into feature data and target data (X_all and y_all, respec
tively)
X_all = student_data[feature_cols]
y_all = student_data[target_col]

# Show the feature information by printing the first five rows
print("\nFeature values:")
print(X_all.head())
```

```
Feature columns:
['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu',
'Fedu', 'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'study
time', 'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'n
ursery', 'higher', 'internet', 'romantic', '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   t
eacher
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  se
rvices
4     GP   F   16       U     GT3       T     3     3    other
other

    ...    higher internet  romantic  famrel  freetime goout Dalc
Walc health  \
0   ...       yes       no        no       4         3     4    1
1      3
1   ...       yes      yes        no       5         3     3    1
1      3
2   ...       yes      yes         no       4         3     2    2
3      3
3   ...       yes      yes       yes       3         2     2    1
1      5
4   ...       yes       no        no       4         3     2    1
2      5

  absences
0        6
1        4
2       10
3        2
4        4

[5 rows x 30 columns]
```

Preprocess Feature Columns

As you can see, there are several non-numeric columns that need to be converted! Many of them are simply `yes`/`no`, e.g. `internet`. These can be reasonably converted into `1`/`0` (binary) values.

Other columns, like `Mjob` and `Fjob`, have more than two values, and are known as *categorical variables*. The recommended way to handle such a column is to create as many columns as possible values (e.g. `Fjob_teacher`, `Fjob_other`, `Fjob_services`, etc.), and assign a `1` to one of them and `0` to all others.

These generated columns are sometimes called *dummy variables*, and we will use the `pandas.get_dummies()` (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html?highlight=get_dummies#pandas.get_dummies) function to perform this transformation. Run the code cell below to perform the preprocessing routine discussed in this section.

In [12]:

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

    # Initialize new output DataFrame
    output = pd.DataFrame(index = X.index)

    # Investigate each feature column for the data
    for col, col_data in X.iteritems():
        # If data type is non-numeric, replace all yes/no values with 1/0
        if col_data.dtype == object:
            col_data = col_data.replace(['yes', 'no'], [1, 0])

        # If data type is categorical, convert to dummy variables
        if col_data.dtype == object:
            # Example: 'school' => 'school_GP' and 'school_MS'
            col_data = pd.get_dummies(col_data, prefix = col)

        # Collect the revised columns
        output = output.join(col_data)

    return output

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

```
Processed feature columns (48 total features):
['school_GP', 'school_MS', 'sex_F', 'sex_M', 'age', 'address_R', '
address_U', 'famsize_GT3', 'famsize_LE3', 'Pstatus_A', 'Pstatus_T'
, 'Medu', 'Fedu', 'Mjob_at_home', 'Mjob_health', 'Mjob_other', 'Mj
ob_services', 'Mjob_teacher', 'Fjob_at_home', 'Fjob_health', 'Fjob
_other', 'Fjob_services', 'Fjob_teacher', 'reason_course', 'reason
_home', 'reason_other', 'reason_reputation', 'guardian_father', 'g
uardian_mother', 'guardian_other', 'traveltime', 'studytime', 'fai
lures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'h
igher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Da
lc', '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 [13]:

```
# TODO: Import any additional functionality you may need here
from sklearn import cross_validation
# 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 = cross_validation.train_test_split(X_all, y_
all,
                                                                     train_siz
e=num_train,
                                                                     test_size
=num_test,
                                                                     random_st
ate=True)

# 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]))

#Also, let's check which features are important
# from sklearn.ensemble import ExtraTreesClassifier

# forest = ExtraTreesClassifier(n_estimators=250, random_state=0)
# forest.fit(X_train, y_train)

# importances = forest.feature_importances_
# indices = np.argsort(importances)[::-1]

# print("Feature Importances: ")
# for i in indices:
#     print("%s : %f" % (X_all.columns[i], importances[i]))
```

```
Training set has 300 samples.
Testing set has 95 samples.
```

Training and Evaluating Models

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

Question 2 - Model Application

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

Answer:

In this project I decided to cover those prediction models which were most detailed covered in supporting materials. From my point of view those are Decision Tree, Support Vector Machines and Naive Bayes classifiers. I have chodsen them because all of them provide completely different prediction approaches:

1. Decision Tree - http://scikit-learn.org/stable/modules/tree.html (http://scikit-learn.org/stable/modules/tree.html) :

Decision Tree Classifier builds a model which makes a prediction based on decision rules inferred from the data features.

```
Advantages:

    * Very simple for understanding and visualising model.
    * Very fast in prediction(Logarighmic cost in the number of data po
ints used to train model).
    * Easly handles numerical and categorical inputs. Moreover, model c
an process categorical data without creating a dummy features. Although
we created dummy-features to process our data, decision tree classifier
can handle even original features.

Disadvantages:

    * As will be shown in default DecisionTreeClassifier below, model m
ay become overfitted which doesn't generalize data well. But that's whe
re ensemble methods like random forests (or boosted trees) come in.
    * Decision tree learners create biased trees if some classes domina
te. It is therefore recommended to balance the dataset prior to fitting
with the decision tree.
    * Decision trees can be unstable because small variations in the da
ta might result in a completely different tree being generated. This pr
oblem is mitigated by using decision trees within an ensemble.
```

1. Support Vector Machines - http://scikit-learn.org/stable/modules/svm.html (http://scikit-learn.org/stable/modules/svm.html) :

SVM classifier builds a linear decision boundary in high dimensional space. If data cannot be splitted linearly classifier creates additional dimension using provided kernel method where data could be splitted linearly.

```
Advantages:

    * Effective on data with big amount of features.
    * Effective if amount of features is higher than amount of samples.
    * Flexible. Various kernel methods. Possible to define custom kerne
l functions.

Disadvantages:

    * Poor performance if amount of features is much higher than sample
size.
    * Model doesn't directly provide probability estimates.
```

1. Naive Bayes http://scikit-learn.org/stable/modules/naive_bayes.html (http://scikit-learn.org/stable/modules/naive_bayes.html) :

Naive Bayes is a kind of probablistic classifier which calculates a probability that given input corresponds to some label. Based on Bayesian theorem. Assumes that all features between input are independent(for instance, it expects that there is no correlation between you gender and age while models calculate whether student needs intervention or not).

```
Advantages:
    * Known as a good classifier in real-world situations.
    * Requires a small amount of training data to estimate the necessar
y parameters well.

Disadvantages:
    * Although naive Bayes is known as a good classifier, it is known t
o be a bad estimator.
    * Although it requires a small amount of data for training, data sh
ould be representative. Because, if a given class and feature value nev
er occur together in the training data, then the frequency-based probab
ility estimate will be zero. As far as I know, to prevent it we could a
dd some fake data to the dataset. I learned this technique on one of Se
bastian's Intro to Statistic courses.
```

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 [14]:

```
from time import time
from sklearn.metrics import f1_score

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, y_pred, pos_label='yes')


def train_predict(clf, X_train, y_train, X_test, y_test):
    ''' Train and predict using a classifer based on F1 score. '''

    # Indicate the classifier and the training set size
    print("Training a {} using a training set size of {}. . .".format(clf.__cl
ass__.__name__, len(X_train)))

    # Train the classifier
    train_classifier(clf, X_train, y_train)

    # Print the results of prediction for both training and testing
    print("F1 score for training set: {:.4f}.".format(predict_labels(clf, X_tr
ain, y_train)))
    print("F1 score for test set: {:.4f}.".format(predict_labels(clf, X_test,
y_test)))
```

Implementation: Model Performance Metrics

With the predefined functions above, you will now import the three supervised learning models of your choice and run the `train_predict` function for each one. Remember that you will need to train and predict on each classifier for three different training set sizes: 100, 200, and 300. Hence, you should expect to have 9 different outputs below — 3 for each model using the varying training set sizes. In the following code cell, you will need to implement the following:

- Import the three supervised learning models you've discussed in the previous section.
- Initialize the three models and store them in `clf_A`, `clf_B`, and `clf_C`.
  - Use a `random_state` for each model you use, if provided.
- 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 [15]:

```
# TODO: Import the three supervised learning models from sklearn
from sklearn import tree
from sklearn import svm
from sklearn import naive_bayes

# TODO: Initialize the three models
clf_A = tree.DecisionTreeClassifier()
clf_B = svm.SVC()
clf_C = naive_bayes.MultinomialNB()

# TODO: Set up the training set sizes
X_train_100 = X_train[:100]
y_train_100 = y_train[:100]

X_train_200 = X_train[:200]
y_train_200 = y_train[:200]

X_train_300 = X_train
y_train_300 = y_train

# TODO: Execute the 'train_predict' function for each classifier and each trai
ning set size
print("##### DECISION TREE CLASSIFIER #####")
train_predict(clf_A, X_train_100, y_train_100, X_test, y_test)
print()
train_predict(clf_A, X_train_200, y_train_200, X_test, y_test)
print()
train_predict(clf_A, X_train_300, y_train_300, X_test, y_test)
print("\n\n\n\n")
print("##### SUPPORT VECTOR MACHINES CLASSIFIER #####")
train_predict(clf_B, X_train_100, y_train_100, X_test, y_test)
print()
train_predict(clf_B, X_train_200, y_train_200, X_test, y_test)
print()
train_predict(clf_B, X_train_300, y_train_300, X_test, y_test)
print("\n\n\n\n")
print("##### NAIVE BAYES CLASSIFIER #####")
train_predict(clf_C, X_train_100, y_train_100, X_test, y_test)
print()
train_predict(clf_C, X_train_200, y_train_200, X_test, y_test)
print()
train_predict(clf_C, X_train_300, y_train_300, X_test, y_test)
```

```
##### DECISION TREE CLASSIFIER #####
Training a DecisionTreeClassifier using a training set size of 100
. . .
Trained model in 0.0009 seconds
Made predictions in 0.0005 seconds.
F1 score for training set: 1.0000.
Made predictions in 0.0001 seconds.
F1 score for test set: 0.6612.

Training a DecisionTreeClassifier using a training set size of 200
. . .
Trained model in 0.0015 seconds
```

```
Made predictions in 0.0001 seconds.
F1 score for training set: 1.0000.
Made predictions in 0.0001 seconds.
F1 score for test set: 0.7328.

Training a DecisionTreeClassifier using a training set size of 300
. . .
Trained model in 0.0025 seconds
Made predictions in 0.0002 seconds.
F1 score for training set: 1.0000.
Made predictions in 0.0001 seconds.
F1 score for test set: 0.6880.




##### SUPPORT VECTOR MACHINES CLASSIFIER #####
Training a SVC using a training set size of 100. . .
Trained model in 0.0020 seconds
Made predictions in 0.0008 seconds.
F1 score for training set: 0.8591.
Made predictions in 0.0007 seconds.
F1 score for test set: 0.8333.

Training a SVC using a training set size of 200. . .
Trained model in 0.0036 seconds
Made predictions in 0.0023 seconds.
F1 score for training set: 0.8581.
Made predictions in 0.0011 seconds.
F1 score for test set: 0.8408.

Training a SVC using a training set size of 300. . .
Trained model in 0.0070 seconds
Made predictions in 0.0047 seconds.
F1 score for training set: 0.8584.
Made predictions in 0.0015 seconds.
F1 score for test set: 0.8462.




##### NAIVE BAYES CLASSIFIER #####
Training a MultinomialNB using a training set size of 100. . .
Trained model in 0.0081 seconds
Made predictions in 0.0015 seconds.
F1 score for training set: 0.8209.
Made predictions in 0.0001 seconds.
F1 score for test set: 0.7647.

Training a MultinomialNB using a training set size of 200. . .
Trained model in 0.0013 seconds
Made predictions in 0.0001 seconds.
F1 score for training set: 0.8099.
Made predictions in 0.0001 seconds.
F1 score for test set: 0.8333.
```

```
Training a MultinomialNB using a training set size of 300. . .
Trained model in 0.0012 seconds
Made predictions in 0.0001 seconds.
F1 score for training set: 0.8019.
Made predictions in 0.0001 seconds.
F1 score for test set: 0.8148.
```

Tabular Results

Edit the cell below to see how a table can be designed in Markdown (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#tables). You can record your results from above in the tables provided.

Classifer 1 - Decision Tree

Training Set Size	Prediction Time (train)	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0002 seconds	0.0005 seconds	1.0000	0.6500
200	0.0004 seconds	0.0001 seconds	1.0000	0.7231
300	0.0058 seconds	0.0001 seconds	1.0000	0.7031

Classifer 2 - Support Vector Machines

Training Set Size	Prediction Time (train)	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0006 seconds	0.0006 seconds	0.8591	0.8333
200	0.0032 seconds	0.0016 seconds	0.8581	0.8408
300	0.0058 seconds	0.0020 seconds	0.8584	0.8462

Classifer 3 - Naive Bayes Classifier

Training Set Size	Prediction Time (train)	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0001 seconds	0.0001 seconds	0.8209	0.7647
200	0.0001 seconds	0.0001 seconds	0.8099	0.8333
300	0.0001 seconds	0.0001 seconds	0.8019	0.8148

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:

From my point of view, in this case Naive Bayes Classifier shows the best result. It built upon assumption that all features are independent: P(X_i|y, X_1,...,X_i-1,X_i+1,...X_n) = P(X_i|y). The model trains extremely fast and provides good prediction accuracy. It has a bit less accuracy comparing to SVM, but the learning speed is dramatically faster as amount of data growth. So from my point of view this minor difference in accuracy and major difference in time performance could be a reasanable point to choose Naive Bayes classifier. The only disadvantage of NBC is that if

Speaking about Decision Tree Classifier it showed the worst result. The time complexity also growth quite fast and it provides the lowest accuracy rate on test data. Moreover the accuracy on training set is maximum(f1 score equails 1), which means that this model is overfitted.

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:

Naive Bayes Classifier is an example of probabilistic classification, which means that classifier provides probability that input corresponds to some class rather than just putting some most likely class on input. Naive Bayes Classifier (NBC) is based on simple Bayesian Theorem, which is following *Posterior = (Likelihood * Prior) / Evidence*.

Let's build an example. Since NBC assumes that there is no relation between features in a class we provide example with just one feature and outcome:

Age	Passed
15	no
17	yes
17	yes
17	no
16	yes
16	no
15	no
15	no

16	yes
15	yes

Frequemcy Table

Age	yes	no
15	1	3
16	2	1
17	2	1

Likelihood table

Age	yes	no		
15	1	3	4/10	0.4
16	2	1	3/10	0.3
17	2	1	3/10	0.3
ALL	5	5		
	5/10	5/10		
	0.5	0.5		

All these steps above happen for each feature during training phase. Now, suppose we want to know does 15th years old student needs intervention or not. To do that we have to compute probability whether student needs intervention or not, being 15 y. o:

P(yes|15) = P(15|yes) *P(yes)* / *P(15) = (1/5)* (0.5) / (4/10) = 0.25

P(no|15) = P(15|no) *P(no)/P(15) = (3/5)* (0.5) / (4/10) = 0.75

Based on the calculations above 15 y. o. student needs an intervention with probability 75%. These calculations happen during the test phase.

Implementation: Model Tuning

Fine tune the chosen model. Use grid search (`GridSearchCV`) with at least one important parameter tuned with at least 3 different values. You will need to use the entire training set for this. In the code cell below, you will need to implement the following:

- Import `sklearn.grid_search.gridSearchCV` (http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html) and `sklearn.metrics.make_scorer` (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.make_scorer.html).
- Create a dictionary of parameters you wish to tune for the chosen model.
  - Example: `parameters = {'parameter' : [list of values]}`.
- Initialize the classifier you've chosen and store it in `clf`.
- Create the F_1 scoring function using `make_scorer` and store it in `f1_scorer`.
  - Set the `pos_label` parameter to the correct value!
- Perform grid search on the classifier `clf` using `f1_scorer` as the scoring method, and store it in `grid_obj`.
- Fit the grid search object to the training data (`X_train`, `y_train`), and store it in `grid_obj`.

In [17]:

```
# TODO: Import 'gridSearchCV' and 'make_scorer'
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import make_scorer, f1_score

# TODO: Create the parameters list you wish to tune
parameters = {'alpha':[0, 0.3, 0.6, 1.0], 'fit_prior':[True, False]}

# TODO: Initialize the classifier
clf = naive_bayes.MultinomialNB()

# 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 scori
ng method
grid_obj = GridSearchCV(clf, parameters, f1_scorer)

# TODO: Fit the grid search object to the training data and find the optimal p
arameters
grid_obj = grid_obj.fit(X_train, y_train)

# Get the estimator
clf = grid_obj.best_estimator_

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

```
Made predictions in 0.0001 seconds.
Tuned model has a training F1 score of 0.8019.
Made predictions in 0.0001 seconds.
Tuned model has a testing F1 score of 0.8148.
```

Question 5 - Final F_1 Score

What is the final model's F_1 score for training and testing? How does that score compare to the untuned model?

Answer:

The best estimator provided by GridSearchCV gives same F1 score as model without tuning, which means that MultinomialNB configuration is the best one.

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.