student_intervention

2016 年 7 月 14 日

1 Machine Learning Engineer Nanodegree

1.1 Supervised Learning

1.2 Project 2: Building a Student Intervention System

Welcome to the second project of the Machine Learning Engineer Nanodegree! In this notebook, some template code has already been provided for you, and it will be your job to implement the additional functionality necessary to successfully complete this project. Sections that begin with **'Implementation'** in the header indicate that the following block of code will require additional functionality which you must provide. Instructions will be provided for each section and the specifics of the implementation are marked in the code block with a `'TODO'` statement. Please be sure to read the instructions carefully!

In addition to implementing code, there will be questions that you must answer which relate to the project and your implementation. Each section where you will answer a question is preceded by a **'Question X'** header. Carefully read each question and provide thorough answers in the following text boxes that begin with **'Answer:'**. Your project submission will be evaluated based on your answers to each of the questions and the implementation you provide.

> **Note:** Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. In addition, Markdown cells can be edited by typically double-clicking the cell to enter edit mode.

1.2.1 Question 1 - Classification vs. Regression

Your goal for this project is to identify students who might need early intervention before they fail to graduate. Which type of supervised learning problem is this, classification or regression? Why?

Answer: 我认为该问题为分类问题而非回归问题。可以将输出定义为是否需要教学干预，即一个二分类问题『是』或者『不是』。

1.3 Exploring the Data

Run the code cell below to load necessary Python libraries and load the student data. Note that the last column from this dataset, `'passed'`, will be our target label (whether the student graduated or didn't graduate). All other columns are features about each student.

In [1]:
```
# Import libraries
import numpy as np
import pandas as pd
from time import time
from sklearn.metrics import f1_score

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

```
Student data read successfully!
```

1.3.1 Implementation: Data Exploration

Let's begin by investigating the dataset to determine how many students we have information on, and learn about the graduation rate among these students. In the code cell below, you will need to compute the following: - The total number of students, `n_students`. - The total number of features for each student, `n_features`. - The number of those students who passed, `n_passed`. - The number of those students who failed, `n_failed`. - The graduation rate of the class, `grad_rate`, in percent (%).

In [2]:
```
# TODO: Calculate number of students
n_students = len(student_data)

# 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'])

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

# TODO: Calculate graduation rate
grad_rate = (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%
```

1.4 Preparing the Data

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

1.4.1 Identify feature and target columns

It is often the case that the data you obtain contains non-numeric features. This can be a problem, as most machine learning algorithms expect numeric data to perform computations with.

Run the code cell below to separate the student data into feature and target columns to see if any features are non-numeric.

```
In [3]: # Extract feature columns
        feature_cols = list(student_data.columns[:-1])

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

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

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

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

```
Feature columns:
['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu', 'Mjob', 'Fjob', 'reason'

Target column: passed

Feature values:
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  \
0     GP   F   18       U     GT3       A     4     4  at_home   teacher
1     GP   F   17       U     GT3       T     1     1  at_home     other
2     GP   F   15       U     LE3       T     1     1  at_home     other
3     GP   F   15       U     GT3       T     4     2   health  services
4     GP   F   16       U     GT3       T     3     3    other     other

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

  absences
0        6
1        4
2       10
3        2
4        4

[5 rows x 30 columns]
```

1.4.2 Preprocess Feature Columns

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

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

These generated columns are sometimes called dummy variables, and we will use the `pandas.get_dummies()` function to perform this transformation. Run the code cell below to

perform the preprocessing routine discussed in this section.

```
In [4]: def preprocess_features(X):
            ''' Preprocesses the student data and converts non-numeric binary variables into
                binary (0/1) variables. Converts categorical variables into dummy variables. '''

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

            # Investigate each feature column for the data
            for col, col_data in X.iteritems():

                # If data type is non-numeric, replace all yes/no values with 1/0
                if col_data.dtype == object:
                    col_data = col_data.replace(['yes', 'no'], [1, 0])

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

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

            return output

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

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

1.4.3 Implementation: Training and Testing Data Split

So far, we have converted all categorical features into numeric values. For the next step, we split the data (both features and corresponding labels) into training and test sets. In the following code cell below, you will need to implement the following: - Randomly shuffle and split the data (`X_all`, `y_all`) into training and testing subsets. - Use 300 training points (approximately 75%) and 95 testing points (approximately 25%). - Set a `random_state` for the function(s) you use, if provided. - Store the results in `X_train`, `X_test`, `y_train`, and `y_test`.

```
In [5]: # TODO: Import any additional functionality you may need here
        from sklearn.cross_validation import train_test_split

        # TODO: Set the number of training points
        num_train = 300

        # Set the number of testing points
        num_test = X_all.shape[0] - num_train

        random_state = 10
        # TODO: Shuffle and split the dataset into the number of training and testing points above
        X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_size = num_test, ra
        # Show the results of the split
        print ("Training set has {} samples.".format(X_train.shape[0]))
        print ("Testing set has {} samples.".format(X_test.shape[0]))
```

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

1.5 Training and Evaluating Models

In this section, you will choose 3 supervised learning models that are appropriate for this problem and available in `scikit-learn`. You will first discuss the reasoning behind choosing these three models by considering what you know about the data and each model' s strengths and weaknesses. You will then fit the model to varying sizes of training data (100 data points, 200 data points, and 300 data points) and measure the F1 score. You will need to produce three tables (one for each model) that shows the training set size, training time, prediction time, F1 score on the training set, and F1 score on the testing set.

1.5.1 Question 2 - Model Application

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

Answer: 1. 决策树。该算法多用于企业管理实践，企业投资决策。优点是易于理解，容易提取出规则。缺点是容易过拟合，并且如果数据缺失会出现问题。由于题目中提供了多重 feature，且大部分为 true/false（0/1）形式，我认为采用决策树提问题的形式可以得出有效的结论。2.SVM。SVM 在图像识别文本分类有较好的表现，在解决小样本、非线性及高维模式识别问题中表现出许多特有的优势。但是核函数比较抽象，并且对数据缺失敏感性。本项目中问题正适合将其映射到高维度进行分类解决。3.KNN 算法。KNN 是基于数据的算法，在数据到来之前不需要进行分析。在分类领域有很多的实践，当样本容量较大时耗费时间会很久，但是我认为本项目中数据量不是很大，因此可以采用 KNN 算法。

1.5.2 Setup

Run the code cell below to initialize three helper functions which you can use for training and testing the three supervised learning models you' ve chosen above. The functions are as follows: - `train_classifier` - takes as input a classifier and training data and fits the classifier to the data. - `predict_labels` - takes as input a fit classifier, features, and a target labeling and makes predictions using the F1 score. - `train_predict` - takes as input a classifier, and the training and testing data, and performs `train_clasifier` and `predict_labels`. - This function will report the F1 score for both the training and testing data separately.

In [6]:
```
def train_classifier(clf, X_train, y_train):
    ''' Fits a classifier to the training data. '''

    # Start the clock, train the classifier, then stop the clock
    start = time()
    clf.fit(X_train, y_train)
    end = time()

    # Print the results
    print ("Trained model in {:.4f} seconds".format(end - start))


def predict_labels(clf, features, target):
    ''' Makes predictions using a fit classifier based on F1 score. '''

    # Start the clock, make predictions, then stop the clock
    start = time()
    y_pred = clf.predict(features)
    end = time()

    # Print and return results
    print( "Made predictions in {:.4f} seconds.".format(end - start))
    return f1_score(target.values, y_pred, pos_label='yes')


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

    # Indicate the classifier and the training set size
    print ("Training a {} using a training set size of {}. . .".format(clf.__class__.__nam
```

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

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

1.5.3 Implementation: Model Performance Metrics

With the predefined functions above, you will now import the three supervised learning models of your choice and run the `train_predict` function for each one. Remember that you will need to train and predict on each classifier for three different training set sizes: 100, 200, and 300. Hence, you should expect to have 9 different outputs below — 3 for each model using the varying training set sizes. In the following code cell, you will need to implement the following: - Import the three supervised learning models you've discussed in the previous section. - Initialize the three models and store them in `clf_A`, `clf_B`, and `clf_C`. - Use a `random_state` for each model you use, if provided. - **Note:** Use the default settings for each model — you will tune one specific model in a later section. - Create the different training set sizes to be used to train each model. - Do not reshuffle and resplit the data! The new training points should be drawn from *X_train* and *y_train*. - Fit each model with each training set size and make predictions on the test set (9 in total).
Note: Three tables are provided after the following code cell which can be used to store your results.

In [9]:
```
# TODO: Import the three supervised learning models from sklearn
# from sklearn import model_A
# from sklearn import model_B
# from skearln import model_C
from sklearn import tree
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

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

# TODO: Set up the training set sizes
X_train_100 = X_train[0:100]
y_train_100 = y_train[0:100]
```

```
        X_train_200 = X_train[0:200]
        y_train_200 = y_train[0:200]

        X_train_300 = X_train[0:300]
        y_train_300 = y_train[0:300]
        models = [clf_A, clf_B, clf_C]
        # TODO: Execute the 'train_predict' function for each classifier and each training set siz
        # train_predict(clf, X_train, y_train, X_test, y_test)
        for clf in models:
            print ("\n{}: \n".format(clf.__class__.__name__))
            for n in [100, 200, 300]:
                train_predict(clf, X_train[:n], y_train[:n], X_test, y_test)
```

```
DecisionTreeClassifier:

Training a DecisionTreeClassifier using a training set size of 100. . .
Trained model in 0.0029 seconds
Made predictions in 0.0011 seconds.
F1 score for training set: 1.0000.
Made predictions in 0.0004 seconds.
F1 score for test set: 0.6870.
Training a DecisionTreeClassifier using a training set size of 200. . .
Trained model in 0.0019 seconds
Made predictions in 0.0003 seconds.
F1 score for training set: 1.0000.
Made predictions in 0.0003 seconds.
F1 score for test set: 0.7059.
Training a DecisionTreeClassifier using a training set size of 300. . .
Trained model in 0.0093 seconds
Made predictions in 0.0006 seconds.
F1 score for training set: 1.0000.
Made predictions in 0.0003 seconds.
F1 score for test set: 0.6720.

SVC:

Training a SVC using a training set size of 100. . .
Trained model in 0.0056 seconds
Made predictions in 0.0014 seconds.
F1 score for training set: 0.8366.
```

```
Made predictions in 0.0014 seconds.
F1 score for test set: 0.8228.
Training a SVC using a training set size of 200. . .
Trained model in 0.0064 seconds
Made predictions in 0.0056 seconds.
F1 score for training set: 0.8552.
Made predictions in 0.0021 seconds.
F1 score for test set: 0.7947.
Training a SVC using a training set size of 300. . .
Trained model in 0.0156 seconds
Made predictions in 0.0085 seconds.
F1 score for training set: 0.8615.
Made predictions in 0.0026 seconds.
F1 score for test set: 0.8079.

KNeighborsClassifier:

Training a KNeighborsClassifier using a training set size of 100. . .
Trained model in 0.0017 seconds
Made predictions in 0.0036 seconds.
F1 score for training set: 0.8108.
Made predictions in 0.0025 seconds.
F1 score for test set: 0.7763.
Training a KNeighborsClassifier using a training set size of 200. . .
Trained model in 0.0011 seconds
Made predictions in 0.0042 seconds.
F1 score for training set: 0.8333.
Made predictions in 0.0063 seconds.
F1 score for test set: 0.7794.
Training a KNeighborsClassifier using a training set size of 300. . .
Trained model in 0.0012 seconds
Made predictions in 0.0061 seconds.
F1 score for training set: 0.8421.
Made predictions in 0.0069 seconds.
F1 score for test set: 0.7714.
```

1.5.4 Tabular Results

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

** Classifer 1 - DecisionTree**

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0029	0.0004	1	0.6870
200	0.0019	0.0003	1	0.7059
300	0.0093	0.0003	1	0.6720

** Classifer 2 -SVM**

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0056	0.0014	0.8366	0.8228
200	0.0064	0.0021	0.8552	0.7947
300	0.0156	0.0026	0.8615	0.8079

** Classifer 3 - KNN**

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0017	0.0019	0.8108	0.7763
200	0.0011	0.0016	0.8333	0.7794
300	0.0012	0.0041	0.8421	0.7714

1.6 Choosing the Best Model

In this final section, you will choose from the three supervised learning models the best model to use on the student data. You will then perform a grid search optimization for the model over the entire training set (`X_train` and `y_train`) by tuning at least one parameter to improve upon the untuned model’ s F1 score.

1.6.1 Question 3 - Chosing the Best Model

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

Answer: 经上图比较我认为 SVM 是相比较最优的。1. 从准确率来说由图可知 SVM 毫无疑问是最高的。随着 train points 增加，决策树对于测试集的预测呈现出越来越不准确的趋势，其预测上限是 100 个数据时的 0.7068. 而 KNN 我当前设定其 K 为 3，随着数据集增加预测率呈现上升趋势，但是最

高也没有高过 SVM。SVM 的预测准确率一直能维持在 0.8 附近，因此从正确率来说我认为各个模型之间差距还是比较大的，因此选取 SVM。2. 由于没有控件开销，从时间开销进行分析。首先，该项目是为了预测学生是否需要教育帮助，因此真正实践的样本应该很久才回更新一次。也就是说模型经过一次训练后进行多次预测。相比于模型训练时间，我们优先选择预测时间小的模型。从表中可以看出，KNN 算法每次预测时都需要耗费最多的时间，因此先将其排除。比较决策树与 SVM，在此决策树胜出，SVM 处于三个算法的中间。综上所述，我认为模型首要的是保证预测的准确率，在保证准确率的情况下尽量选择时间开销少的模型。因此最终选择 SVM 进行模型预测。

1.6.2 Question 4 - Model in Layman's Terms

In one to two paragraphs, explain to the board of directors in layman's terms how the final model chosen is supposed to work. For example if you've chosen to use a decision tree or a support vector machine, how does the model go about making a prediction?

Answer:

最终选择的 SVM 即支持向量机。SVM 在本项目中用于将学生分类，通过读入学生的特征预测一个结果来显示该学生是否需要教育干预。通过前期的数据训练，最终能达到中等资源耗费情况下的高准确率。其原理简单阐述如下。参考图 SVM_2.png，红色和绿色代表二维平面的两个集群（如同『需要教育干预』和『无须教育干预』的两类学生）。我们要做的是找一条线将这两个种类分开。Margin width 是这两个种类在该平面下的距离。直观上看，我们想要的直线处于该区域的正中间时为最好情况，此时它距离两个分类的 “距离” 都为最大。SVM 就是找到这样一条直线的算法。找到这条直线后我们就可以将数据分类了。但是由于我们的 feature 过多，在一个平面内可能无法生成这样一条直线。为了解决这个问题，我们需要借助于一个叫做核函数的工具，对当前特征进行一系列内积运算，然后产生一个映射，在这个空间里我们的数据是线性可分的。于是通过核函数我们找到了这样的一个空间满足数据线性可分。进行分类后我们自然能够预测一个学生需不需要进行教育干预了。

1.6.3 Implementation: Model Tuning

Fine tune the chosen model. Use grid search (`GridSearchCV`) with at least one important parameter tuned with at least 3 different values. You will need to use the entire training set for this. In the code cell below, you will need to implement the following: - Import `sklearn.grid_search.gridSearchCV` and `sklearn.metrics.make_scorer`. - Create a dictionary of parameters you wish to tune for the chosen model. - Example: `parameters = {'parameter' : [list of values]}`. - Initialize the classifier you've chosen and store it in `clf`. - Create the F1 scoring function using `make_scorer` and store it in `f1_scorer`. - Set the `pos_label` parameter to the correct value! - Perform grid search on the classifier `clf` using `f1_scorer` as the scoring method, and store it in `grid_obj`. - Fit the grid search object to the training data (`X_train`, `y_train`), and store it in `grid_obj`.

In [68]:
```
# TODO: Import 'GridSearchCV' and 'make_scorer'
from sklearn.metrics import make_scorer
from  sklearn.grid_search import GridSearchCV
```

```
from sklearn.metrics import f1_score
# TODO: Create the parameters list you wish to tune
parameters = {'kernel':['linear', 'rbf','sigmoid']}

# TODO: Initialize the classifier
clf = SVC()

# TODO: Make an f1 scoring function using 'make_scorer'
f1_scorer = make_scorer(f1_score,pos_label = 'yes')

# TODO: Perform grid search on the classifier using the f1_scorer as the scoring method
grid_obj = GridSearchCV(clf, parameters, f1_scorer)
# TODO: Fit the grid search object to the training data and find the optimal parameters
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(clf, X_trai
print ("Tuned model has a testing F1 score of {:.4f}.".format(predict_labels(clf, X_test,
```

```
Made predictions in 0.0052 seconds.
Tuned model has a training F1 score of 0.8615.
Made predictions in 0.0019 seconds.
Tuned model has a testing F1 score of 0.8079.
```

1.6.4 Question 5 - Final F1 Score

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

Answer: 训练结果显示，经过 gridsearch 产生的 svm 的最优 kernel 依然是采用默认的 rbf. 得到的训练集 F1 为 0.8615，测试集的 f1 为 0.8079. 与之前一样

> **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.