Module 16 - Evaluating Data Models

Author: Jessica Cervi

Reviewer: Jessica Cervi

Expected time = 2.5 hours

Total points = 100 points

Assignment Overview

This assignment will test your ability to implement an OLS (ordinary least squares) regression in Python. We'll briefly review some of the lecture content, followed by an overarching research question that will be guiding this assignment. Throughout the assignment, you will be asked to use the popular `scikit-learn` libraries to implement your LS regression. You will also create several functions throughout the assignment in order to resolve problems and roadblocks to your analysis.

This assignment is designed to build your familiarity and comfort coding in Python while also helping you review key topics from each module. As you progress through the assignment, answers will get increasingly complex. It is important that you adopt a data scientist's mindset when completing this assignment. **Remember to run your code from each cell before submitting your assignment.** Running your code beforehand will notify you of errors and give you a chance to fix your errors before submitting. You should view your Vocareum submission as if you are delivering a final project to your manager or client.

Vocareum Tips

- Do not add arguments or options to functions unless you are specifically asked to. This will cause an error in Vocareum.
- Do not use a library unless you are expicitly asked to in the question.
- You can download the Grading Report after submitting the assignment. This will include feedback and hints on incorrect questions.

Learning Objectives

- Apply linear regression to a dataset in python
- Determine the input variables for a linear regression model
- Analyze the output of linear regression to form conclusions
- Work with classification and optimization models in python

Index:

Module 16 - Evaluating Data Models

Module 16 - Evaluating Data Models

Linear regression

Linear regression is widely used across industries including healthcare, economics, and social sciences. You can expect to frequently encounter data science questions in your career for which a linear regression is the best possible solution.

Regression Equation

A regression analysis yields an equation similar to the slope of a line, which is a mathematical representation of the shape of how your input variables predict your output variables. It's often presented as follows:

$$Y = \alpha + \beta_1 X_1 + \beta_2 X_2 + e$$

Y, representing a value for your outcome variable, is predicted by the slope of the line α (alpha) plus a coefficient, β (beta), multiplied by each X value.

This is similar to $Y = mx + b$, the equation for the slope of a line that you probably learned in grade school math.

Statistical Assumptions

There are several key steps to take in order to produce accurate regression analysis results. Namely, your data needs to meet key *statistical assumptions*:

- **Linearity:** your data is linearly related, or can be transformed to create a linear relationship (i.e., take the square root of a predictor).
- **Multivariate normality:** the residuals produced by your output are normally distributed.
- **Little or no multicollinearity:** your predictors are independent and not highly correlated with each other.
- **Homoscedasticity:** equal variance of errors.

We will cover all of these in detail as we analyze the data set.

The dataset

For this assignment, we will use the `bank_marketing.csv` dataset. Because this dataset has many attributes, we list them below with a short description for your convenience.

Input variables:

Bank client data:

- age (numeric)
- job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')
- marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)
- education (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')
- default: has credit in default? (categorical: 'no','yes','unknown')
- housing: has housing loan? (categorical: 'no','yes','unknown')
- loan: has personal loan? (categorical: 'no','yes','unknown')

Related with the last contact of the current campaign:

- contact: contact communication type (categorical: 'cellular','telephone')
- month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
- day_of_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')
- duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

Other attributes:

- campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
- pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
- previous: number of contacts performed before this campaign and for this client (numeric)
- poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')

social and economic context attributes

- emp.var.rate: employment variation rate - quarterly indicator (numeric)
- cons.price.idx: consumer price index - monthly indicator (numeric)
- cons.conf.idx: consumer confidence index - monthly indicator (numeric)
- euribor3m: euribor 3 month rate - daily indicator (numeric)
- nr.employed: number of employees - quarterly indicator (numeric)

Desired target: Output variable

- y - has the client subscribed a term deposit? (binary: 'yes','no') ```

Our goal will be to use scikitlearn to implement and refine a `LogisticRegression` model to predict the target feature -- `y`.

In [ ]:

```
#importing the necessary libraries
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
```

In [ ]:

```
#Read the dataset
df = pd.read_csv('./data/bank_marketing.csv', index_col=0)
df.head()
```

Back to top

Question 1:

5 points

To begin, we will read in the data and get a high level overview of the data. One thing we are looking for is outliers in the numerical features. We use the `.describe()` method to examine the descriptive statistics of the column `age`. Using a rule of thumb of outliers being defined as:

$$\text{mean} \pm 1.5 \times \text{standard deviation}$$

Your goal is to examine the `age` column for outliers.

Define a function, `outlier_counter`. Your function should take as input a column name, as a string, and should return the count of outliers for that column.

Note that for your convenience, we have included some comments to guide through the steps when creating the function.

In [ ]:

```
df.age.describe()
```

In [ ]:

```
### GRADED

### YOUR SOLUTION HERE
def outlier_counter(col):
    mean = df[col].mean()
    std = df[col].std()
    return df[col][ (df[col] > (mean + (1.5*std))) |\
            (df[col] < (mean - (1.5*std))) ].count()
### END SOLUTION
```

Back to top

Question 2:

5 points

Now, we want to examine the percentage of values in each of the target classes. We can select the column `y` and use the `.value_counts()` method to get started. Next, we need to divide by the length of the data to get a percentage.

Store your answer as a dictionary named `ans2` with the key equal to the class label, and the corresponding value equal to the percentage of that class as a float value. For example:

```
ans3 = {
    0: 0.7,
    1: 0.3
}
```

The above would be comprised 70% with 0 labels and 30% with 1 label.

In [ ]:

```
### GRADED

### YOUR SOLUTION HERE
ans2 = { x[0]: x[1] for x in map(lambda x: (x, \
    df['y'].value_counts(normalize=True)[x]), \
    range(df['y'].nunique()))}
### END SOLUTION
```

Back to top

Question 3:

5 points

In classification problems, we want to be aware of the proportion of data in each class. If our data is **balanced**, we have equal amounts of each class, e.g. 50% class 0 and 50% class 1.

Evaluate the truth of the following statement: "Our dataset has balanced classes." Assign a boolean of True or False to `ans3`.

In [ ]:

```
### GRADED

### YOUR SOLUTION HERE
ans3 = False
### END SOLUTION
```

In [ ]:

```
### BEGIN HIDDEN TESTS
ans3_ = False
#
#
assert ans3_ == ans3, 'Use the same strategy as Question 3 to re-evaluate your answer.'
print("Great job!")
### END HIDDEN TESTS
```

Back to top

Question 4:

5 points

Now, we will prepare our data for a model.

We realize that there are outliers in the data and that we have imbalanced classes, but let us proceed and see how a somewhat troubled Logistic Regression model can do. First, we will subset `df` to only numeric columns and separate the input from output features.

Save the numeric input features as a dataframe to the variable `X` and the target feature as a series to the variable `y` below.

In [ ]:

```
### GRADED

### YOUR SOLUTION HERE
X = df.select_dtypes(['float', 'int']).drop('y', axis = 1)
y = df.y
### END SOLUTION
```

Back to top

Question 5:

5 points

We want to evaluate our model on data that it has not seen before, so we will create a test/train split using the sklearn `train_test_split` method. To ensure our results are the same, we will fix the `random_state` to 24. Use your `X` and `y` variables from above to create your new train and test sets and assign the partitions to `X_train`, `X_test`, `y_train`, and `y_test` below.

In [ ]:

```
from sklearn.model_selection import train_test_split
```

In [ ]:

```
### GRADED

### YOUR SOLUTION HERE
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 24)
### END SOLUTION
```

Back to top

Question 6:

5 points

With our data in hand, we create and fit the model. Using the `LogisticRegression` class to instantiate the variable `lgr`.

Next, fit a model with your *training data* from above using the `.fit()` method. When fitting the model, use the `lbfgs` solver. You should also make sure that your fitting converges (you may have to modify `max_iter` to do this).

For reproducibility, set `random_state =24`.

In [ ]:

```
from sklearn.linear_model import LogisticRegression
```

In [ ]:

```
### GRADED

### YOUR SOLUTION HERE
lgr = LogisticRegression(random_state = 24, solver='lbfgs', max_iter=1000)
lgr.fit(X_train, y_train)
### END SOLUTION
```

Back to top

Question 7:

5 points

After fitting the classifer, we can examine its performance using built in `.score()` method. This is the percentage of predictions that were correct. Evaluate your classifier using the scorer and determine if we have done better than just guessing the majority class.

Evaluate your model on the test set and compare your answer to that of the baseline majority class percentage. Did your classifier perform better than simply guessing 0 every time? Assign a boolean to `ans7` below, with True being a higher accuracy than guessing 0 for every class.

In [ ]:

```
### GRADED

### YOUR SOLUTION HERE
ans7= True
### END SOLUTION
```

Back to top

Question 8:

5 points

Using the `.predict` method of the `LogisticRegression` class, *after fitting your model against your X train set* , generate an array of predicted values against your `X_test` set, saving the result as a `numpy.ndarray` object into the `ans8` variable.

In [ ]:

```
### GRADED

### YOUR SOLUTION HERE
ans8 = lgr.predict(X_train)
### END SOLUTION
```

Back to top

Question 9:

5 points

By default, the sklearn `confusion_matrix` takes in an array of "true" y values, and an array of predictions from a model. It returns a confusion matrix with the true 0 class represented in the first row, and true 1 class by the second. The first column represents points labeled as 0 by the model, and the second column those that were labeled 1. The resulting answer should be a dataframe that looks similar to this:

.	0	1
0	123	456
1	789	876

Use your true and predicted (computed in Question 8) values for `y` to make a confusion matrix with the `confusion_matrix` method. Save your confusion matrix to `ans9`.

In [ ]:

```
from sklearn.metrics import confusion_matrix
```

In [ ]:

```
### GRADED

### YOUR SOLUTION HERE
pred = lgr.predict(X_train)
ans9 = confusion_matrix(y_train, pred)
### END SOLUTION
```

Evaluating results

Consider the following fable:

> *A shepherd boy gets bored tending to the town's flock. To have some fun, he cries out, "Wolf!" even though no wolf is in sight. The villagers run to protect the flock, but then get really mad when they realize the boy was playing a joke on them.*

Let's make the following definitions:

- "Wolf" is a positive class.
- "No wolf" is a negative class.

We can summarize our "wolf-prediction" model using a 2x2 confusion matrix that depicts all four possible outcomes:

A **true positive** is an outcome where the model correctly predicts the positive class. Similarly, a true negative is an outcome where the model correctly predicts the negative class.

A **false positive** is an outcome where the model incorrectly predicts the positive class. And a false negative is an outcome where the model incorrectly predicts the negative class.

As mentioned earlier, the `.score()` method uses **accuracy** to score the classifier. This represents the percentage correct predictions:

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$

For a deeper discussion on evaluation metrics please see the lectures this week and this article.

Back to top

Question 10:

5 points

In the language of the confusion matrix, our example translates to the following four outcomes.

- True Negatives: Classified as 0 and really 0
- False Positives: Classified as 1 and really 0
- False Negatives: Classified as 0 and really 1
- True Positives: Classified as 1 and really 1

We can save these values to compute additional evaluation metrics for our classifier.

RECALL that by default, a `confusion_matrix` uses the labels that correspond with indicies of the matrix, i.e. row 0 column 0 is an object that was truly 0 and labeled as 0.

Use this example to help you use the `confusion_matrix` function to save your classifiers performance on the test set in terms of:

- true negatives : tn
- false positives: fp
- false negatives: fn
- true positives: tp

In [ ]:

```
### GRADED

### YOUR SOLUTION HERE
pred = lgr.predict(X_train)
tn, fp, fn, tp = confusion_matrix(y_train, pred).ravel()
### END SOLUTION
```

Back to top

Question 11:

5 points

The notion of precision can also be described as the *positive predictive value* of a classifier. We compute it using the true positives and false positives of the classifier.

$$PPV = \frac{TP}{TP+FN}$$

Use your values to compute the precision of your classifier. Save your answer to `ans11` below.

In [ ]:

```
### GRADED

### YOUR SOLUTION HERE
ans11 = tp/(tp + fp)
### END SOLUTION
```

Back to top

Question 12:

5 points

Recall can be considered the *true positive rate* and is computed as follows:

$$TPR = \frac{TP}{TP+FN}$$

Using your values from the confusion matrix, calculate the recall score on your test data. Save your solution to `ans12` below.

In [ ]:

```
### GRADED

### YOUR SOLUTION HERE
ans12 = tp/(tp + fn)
### END SOLUTION
```

Back to top

Question 13:

5 points

To make a plot of the the `ROC` curve, we will need to save the true positive rate and false positive rates along with the different values for thresholds that yield these. In sklearn, we use the `roc_curve` method. This takes in an array of true `y` values and a list from the `.decision_function` method of a fit `LogisticRegression` classifier.

Use your fit `LogisticRegression` classifier to get the threshold values from the `decision_function()` method on your X test data. Save the results to `ans13` below of type numpy.ndarray of shape (10297,).

In [ ]:

```
from sklearn.metrics import roc_curve
```

In [ ]:

```
### GRADED

### YOUR SOLUTION HERE
ans13 =  lgr_ans.decision_function(X_train)
### END SOLUTION
```

Back to top

Question 14:

5 points

Now, we save the false positive rates and true positive rates together with their accompanying threshold values using the `roc_curve` method.

Save the false positive rate, the true positive rate, and threshold values below using the roc curve method and your `decison_funtion` from above. Name your variables `fpr`, `tpr` and `thresholds`.

HINT: These variables are NumPy arrays.

In [ ]:

```
from sklearn.metrics import roc_curve
```

In [ ]:

```
### GRADED

### YOUR SOLUTION HERE
thresholds = lgr_ans.decision_function(X_train)
fpr, tpr, thresholds = roc_curve(y_train, thresholds)
### END SOLUTION
```

Back to top

Question 15:

5 points

Using these values, we can plot the ROC curve. We want our false positive rates across the x-axis and true positive rates as the y-axis. Your plot should resemble the one shown below.

Make a *unique* plot displaying `fpr` vs `tpr` and `tpr` vs `fpr` computed above. Save your plot as `plot15.png` in the `results` folder. Do not specify any other option when plotting.

In [ ]:

```
### GRADED

### YOUR SOLUTION HERE
fig = plt.figure()
plt.plot(afpr, atpr)
plt.plot(afpr, afpr)
plt.savefig("results/plot15.png")
### END SOLUTION
```

Back to top

Question 16:

5 points

To see how our labels are applied in making predictions, we can use the `.predict_proba()` method of our `LogisticRegression` classifier. We are interested in the probabilities of predicting membership in class 1 which is the second column of the data.

Save the predicted probabilities of predicting class 1 to `ans16` below. Your answer should be of object type numpy.ndarray and shape (10297,)

In [ ]:

```
### GRADED

### YOUR SOLUTION HERE
ans16 = lgr_ans.predict_proba(X_train)[:,1]
### END SOLUTION
```

Back to top

Question 17:

5 points

Now that we have our thresholds, we can adjust our predictions based on a different threshold than 0.5 -- the default value. To do so, we will create a new array of predictions where we label anything greater than a 0.4 probability as a 1, and anything else as a 0.

Use `np.where` together with your solution to problem 16 to make predictions based on 0.4 threshold. Save your results to `ans17` below.

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
ans17 = np.where(lgr_ans.predict_proba(X_train)[:,1] < 0.4, 0, 1)
### END SOLUTION
```

Back to top

Question 18:

5 points

Now, we can compare our new predictions precision and recall scores. We will use the built-in sklearn scorers from the `metrics` module to do so.

Use the predictions from question 17 to evaluate the performance of your new classifier on the test data. Save the precision score to `ans18` below.

In [ ]:
```
from sklearn.metrics import precision_score, recall_score
```

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
ans18  = precision_score(y_train, ans17)
### END SOLUTION
```

Back to top

Question 19:

5 points

Similarly, we can use the `recall_score` to examine the recall performance of our new classifier.

Evaluate the recall of your classifier from problem 17 on the test set. Save your solution to `ans19` below.

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
ans19  = recall_score(y_train, ans17)
### END SOLUTION
```

Back to top

Question 20:

Now that we adjusted the threshold, we have improved one of our metrics and seen decline in the other. Perhaps this is desirable, or maybe we should move the threshold the other way depending on our metrics and business setting.

What metric improved, `precision` or `recall`? Save your answer as a string to `ans20` below

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
ans20  = "precision"
### END SOLUTION
```