Putting It All Together

June 6, 2021

Putting It All Together As you might have guessed from the last notebook, using all of the variables was allowing you to drastically overfit the training data. This was great for looking good in terms of your Rsquared on these points. However, this was not great in terms of how well you were able to predict on the test data.

We will start where we left off in the last notebook. First read in the dataset.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
import AllTogether as t
import seaborn as sns
%matplotlib inline

df = pd.read_csv('./survey_results_public.csv')
df.head()
```

Out[1]:

	Respondent	Professional	ProgramHobby	Country	University	EmploymentStatus
0	1	Student	Yes, both	United States	No	Not employed, and not looking for work
1	2	Student	Yes, both	United Kingdom	Yes, full-time	Employed part-time
2	3	Professional developer	Yes, both	United Kingdom	No	Employed full-time
3	4	Professional non-developer who sometimes write...	Yes, both	United States	No	
4	5	Professional developer	Yes, I program as a hobby	Switzerland	No	

```
3                     Employed full-time
4                     Employed full-time

                                  FormalEducation  \
0                                Secondary school
1  Some college/university study without earning ...
2                               Bachelor's degree
3                                 Doctoral degree
4                                 Master's degree

                               MajorUndergrad  \
0                                         NaN
1       Computer science or software engineering
2       Computer science or software engineering
3  A non-computer-focused engineering discipline
4       Computer science or software engineering

                                      HomeRemote  \
0                                            NaN
1            More than half, but not all, the time
2  Less than half the time, but at least one day ...
3  Less than half the time, but at least one day ...
4                                          Never

              CompanySize       ...       StackOverflowMakeMoney Gender  \
0                     NaN       ...            Strongly disagree   Male
1       20 to 99 employees       ...            Strongly disagree   Male
2  10,000 or more employees       ...                     Disagree   Male
3  10,000 or more employees       ...                     Disagree   Male
4       10 to 19 employees       ...                          NaN    NaN

  HighestEducationParents                          Race         SurveyLong  \
0             High school  White or of European descent  Strongly disagree
1       A master's degree  White or of European descent     Somewhat agree
2   A professional degree  White or of European descent     Somewhat agree
3       A doctoral degree  White or of European descent              Agree
4                     NaN                           NaN               NaN

  QuestionsInteresting QuestionsConfusing InterestedAnswers    Salary  \
0       Strongly agree           Disagree    Strongly agree       NaN
1       Somewhat agree           Disagree    Strongly agree       NaN
2                Agree           Disagree             Agree  113750.0
3                Agree     Somewhat agree    Strongly agree       NaN
4                  NaN               NaN               NaN       NaN

   ExpectedSalary
0             NaN
1         37500.0
```

```
2             NaN
3             NaN
4             NaN

[5 rows x 154 columns]
```

Question 1 **1.** To begin fill in the format function below with the correct variable. Notice each { } holds a space where one of your variables will be added to the string. This will give you something to do while the the function does all the steps you did throughout this lesson.

```
In [2]: a = 'test_score'
        b = 'train_score'
        c = 'linear model (lm_model)'
        d = 'X_train and y_train'
        e = 'X_test'
        f = 'y_test'
        g = 'train and test data sets'
        h = 'overfitting'

        q1_piat = '''In order to understand how well our {} fit the dataset,
                    we first needed to split our data into {}.
                    Then we were able to fit our {} on the {}.
                    We could then predict using our {}  by providing
                    the linear model the {} for it to make predictions.
                    These predictions were for {}.

                    By looking at the {}, it looked like we were doing awesome because
                    it was 1!  However, looking at the {} suggested our model was not
                    extending well.  The purpose of this notebook will be to see how
                    well we can get our model to extend to new data.

                    This problem where our data fits the training data well, but does
                    not perform well on test data is commonly known as
                    {}.'''.format(c, g, c, d, c, e, f, b, a, h) #replace a with the correct vari

        print(q1_piat)
```

```
In order to understand how well our linear model (lm_model) fit the dataset,
            we first needed to split our data into train and test data sets.
            Then we were able to fit our linear model (lm_model) on the X_train and y_train.
            We could then predict using our linear model (lm_model)  by providing
            the linear model the X_test for it to make predictions.
            These predictions were for y_test.

            By looking at the train_score, it looked like we were doing awesome because
            it was 1!  However, looking at the test_score suggested our model was not
            extending well.  The purpose of this notebook will be to see how
            well we can get our model to extend to new data.
```

```
This problem where our data fits the training data well, but does
not perform well on test data is commonly known as
overfitting.
```

```
In [3]: # Print the solution order of the letters in the format
        t.q1_piat_answer()
```

```
This one is tricky - here is the order of the letters for the solution we had in mind:
 c, g, c, d, c, e, f, b, a, h
```

Question 2 **2.** Now, we need to improve the model . Use the dictionary below to provide the true statements about improving **this model**. **Also consider each statement as a stand alone**. Though, it might be a good idea after other steps, which would you consider a useful **next step**?

```
In [4]: a = 'yes'
        b = 'no'

        q2_piat = {'add interactions, quadratics, cubics, and other higher order terms': b,
                   'fit the model many times with different rows, then average the responses': a
                   'subset the features used for fitting the model each time': a,
                   'this model is hopeless, we should start over': b}
```

```
In [5]: #Check your solution
        t.q2_piat_check(q2_piat)
```

```
Nice job! That looks right!  These two techniques are really common in Machine Learning algorith
```

Question 3 **3.** Before we get too far along, follow the steps in the function below to create the X (explanatory matrix) and y (response vector) to be used in the model. If your solution is correct, you should see a plot similar to the one shown in the Screencast.

```
In [28]: def clean_data(df):
             '''
             INPUT
             df - pandas dataframe

             OUTPUT
             X - A matrix holding all of the variables you want to consider when predicting the
             y - the corresponding response vector

             Perform to obtain the correct X and y objects
             This function cleans df using the following steps to produce X and y:
             1. Drop all the rows with no salaries
             2. Create X as all the columns that are not the Salary column
```

```
        3. Create y as the Salary column
        4. Drop the Salary, Respondent, and the ExpectedSalary columns from X
        5. For each numeric variable in X, fill the column with the mean value of the colum
        6. Create dummy columns for all the categorical variables in X, drop the original c
        '''
    #drop all rows with no salaries
        X = df.dropna(axis=0, subset=['Salary'])

    #create y as Salary column
        y = X['Salary']

    #drop Salary column from X
        X = X.drop(['Salary'], axis=1)

    #drop Salary, Respondent & ExpectedSalary from X
        X = X.drop(['Respondent', 'ExpectedSalary'], axis=1)

    #split X into numerical and categorical variables
        X_num = X.select_dtypes(exclude = 'object').columns
        X_cat = X.select_dtypes('object').columns

    #for numeric variable, fill column with mean value
        for col in X_num:
            X[col].fillna(value = X[col].mean(), inplace=True)

    #for categorical variables, create dummy columns & drop original column
        for col in X_cat:
            X_cat = pd.concat([X_cat.drop(col, axis=1),
                               pd.get_dummies(X_cat[col], prefix=col, prefix_sep='_', drop_fir
                                 axis=1)

        return X, y

    #Use the function to create X and y
    X, y = clean_data(df)


---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-28-f29a4e670801> in <module>()
 47
 48 #Use the function to create X and y
---> 49 X, y = clean_data(df)


<ipython-input-28-f29a4e670801> in clean_data(df)
```

```
     39 #for categorical variables, create dummy columns & drop original column
     40     for col in X_cat:
---> 41         X_cat = pd.concat([X_cat.drop(col, axis='columns'),
     42                            pd.get_dummies(X_cat[col], prefix=col, prefix_sep='_', drop_
     43                              axis=1)


TypeError: drop() got an unexpected keyword argument 'axis'
```

```
In [8]: X_num = X.select_dtypes(exclude = 'object').columns
        X_num

Out[8]: Index(['CareerSatisfaction', 'JobSatisfaction', 'HoursPerWeek',
               'StackOverflowSatisfaction'],
              dtype='object')
```

0.0.1 Run the Cell Below to Acheive the Results Needed for Question 4

```
In [29]: #cutoffs here pertains to the number of missing values allowed in the used columns.
         #Therefore, lower values for the cutoff provides more predictors in the model.
         cutoffs = [5000, 3500, 2500, 1000, 100, 50, 30, 25]

         #Run this cell to pass your X and y to the model for testing
         r2_scores_test, r2_scores_train, lm_model, X_train, X_test, y_train, y_test = t.find_op
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-29-594899327099> in <module>()
  4
  5 #Run this cell to pass your X and y to the model for testing
----> 6 r2_scores_test, r2_scores_train, lm_model, X_train, X_test, y_train, y_test = t.find


NameError: name 'X' is not defined
```

Question 4 **4.** Use the output and above plot to correctly fill in the keys of the **q4_piat** dictionary with the correct variable. Notice that only the optimal model results are given back in the above - they are stored in **lm_model**, **X_train**, **X_test**, **y_train**, and **y_test**. If more than one answer holds, provide a tuple holding all the correct variables in the order of first variable alphabetically to last variable alphabetically.

```
In [30]: print(X_train.shape[1]) #Number of columns
         print(r2_scores_test[np.argmax(r2_scores_test)]) # The model we should implement test_r
         print(r2_scores_train[np.argmax(r2_scores_test)]) # The model we should implement train
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-30-be97c388b35b> in <module>()
----> 1 print(X_train.shape[1]) #Number of columns
      2 print(r2_scores_test[np.argmax(r2_scores_test)]) # The model we should implement tes
      3 print(r2_scores_train[np.argmax(r2_scores_test)]) # The model we should implement tr


NameError: name 'X_train' is not defined
```

```
In [31]: a = 'we would likely have a better rsquared for the test data.'
         b = 1000
         c = 872
         d = 0.69
         e = 0.82
         f = 0.88
         g = 0.72
         h = 'we would likely have a better rsquared for the training data.'

         q4_piat = {'The optimal number of features based on the results is': #letter here,
                        'The model we should implement in practice has a train rsquared of': #le
                        'The model we should implement in practice has a test rsquared of': #let
                        'If we were to allow the number of features to continue to increase': #l
         }


          File "<ipython-input-31-1ebccb9533f1>", line 11
        'The model we should implement in practice has a train rsquared of': #letter here,
                                                                            ^
    SyntaxError: invalid syntax
```

```
In [ ]: #Check against your solution
        t.q4_piat_check(q4_piat)
```

Question 5 **5.** The default penalty on coefficients using linear regression in sklearn is a ridge (also known as an L2) penalty. Because of this penalty, and that all the variables were normalized, we can look at the size of the coefficients in the model as an indication of the impact of each variable on the salary. The larger the coefficient, the larger the expected impact on salary.

Use the space below to take a look at the coefficients. Then use the results to provide the **True** or **False** statements based on the data.

Run the below to complete the following dictionary

```
In [ ]: def coef_weights(coefficients, X_train):
            '''
            INPUT:
            coefficients - the coefficients of the linear model
            X_train - the training data, so the column names can be used
            OUTPUT:
            coefs_df - a dataframe holding the coefficient, estimate, and abs(estimate)

            Provides a dataframe that can be used to understand the most influential coefficient
            in a linear model by providing the coefficient estimates along with the name of the
            variable attached to the coefficient.
            '''
            coefs_df = pd.DataFrame()
            coefs_df['est_int'] = X_train.columns
            coefs_df['coefs'] = lm_model.coef_
            coefs_df['abs_coefs'] = np.abs(lm_model.coef_)
            coefs_df = coefs_df.sort_values('abs_coefs', ascending=False)
            return coefs_df

        #Use the function
        coef_df = coef_weights(lm_model.coef_, X_train)

        #A quick look at the top results
        coef_df.head(20)

In [ ]: a = True
        b = False

        #According to the data...
        q5_piat = {'Country appears to be one of the top indicators for salary': #letter here,
                       'Gender appears to be one of the indicators for salary': #letter here,
                       'How long an individual has been programming appears to be one of the top
                       'The longer an individual has been programming the more they are likely t

In [ ]: t.q5_piat_check(q5_piat)
```

Congrats of some kind Congrats! Hopefully this was a great review, or an eye opening experience about how to put the steps together for an analysis. List the steps. In the next lesson, you will look at how take this and show it off to others so they can act on it.

```
In [ ]:
```