Lab 4 – Regression (95)

Doing a regression

If we look at the visualizations from Lab 2, we see that height and weight are related. The data is more tightly grouped for the lower height/weight pairs and becomes noisier as they increase. We can use either height or weight as the target, but for this lab we will use *weight* as the target. We could restrict ourselves to either children or adults, but we will use all the data instances.

Linear Regression has been around for a long time. If we pick a good error function like ordinary least squares to minimize we can solve for the "best" regression directly. If not, we will need to do a search (gradient descent) over the internal parameters. A search is not guaranteed to converge and introduces other complications. As the number of features increases, we may experience overfitting (model matches each training point closely, but changes rapidly a short distance away.) Depending on the amount of noise in the data a little bit of "regularization" can help. The intercept term is also called the bias. In a simple x predicts y regression, it is the value of the prediction when x=0. Unless you have a reason to believe that the bias is zero, you should allow the model to learn it. This may result in predictions close to zero that are meaningless.

Analysis prelims

We are going to make a copy of our data exploration notebook (lab2.ipynb) and name it lab4_starter.ipynb. We will trim that notebook down and do our training and analysis there.

We need to change the intro cell. Double click on it and change it to:

```
# Lab 4 - Doing regression.
Here is what we will do:
1. Prepare the data
2. Train and analyze a linear regression
     - single input feature
     - multiple input features
3. Train and analyze a polynomial regression
4. Train and analyze a regularized model
```

Click on play while the cell is selected and it will re-render it.

Change the next cell to the following and then re-render it

```
### Prepare the data
```

Removing exploration code.

There is a lot of code that we don't need when creating our models, so let's get rid of it.

In the third cell, get rid of the line howell_full.info().

Select and cut the two cells for **Quick Look at Distributions:**

Select and cut the two cells for **Quick Visualization:**

In the code for **A Better Plot** comment out the first and third plot.

Select and cut the two cells for **Handling Missing Data**: (Normally we would keep any cleaning needed for the data, but the Howell dataset was fine as is.)

While we will not use BMI for our models, it does not hurt to include it. Similarly for the BMI class.

Remove the two print statements from **Adding a new Feature** and paste what remains at the end of the #third cell. When you run the third cell, the head should have the bmi feature.
You may now cut the two cells for **Adding a new Feature.**

Remove the print statement from the first code cell for **Creating a Categorical Feature** and paste what remains at the end of the #third cell. When you run the #third cell, category counts should match from before.
You may now cut the three cells for **Creating a Categorical Feature.**

Remove the three cells from **Splitting the Data by Age**.
.
Remove the cells from **Plot with masking**

Keep the two **Train/Test Data Split** cells, but replace **howell_adults** by **howell_full.** Remove the last three print statements.

Remove the cells from **Stratified Train/Test Data Split** cells.

When you run the cell, you should see train/test counts of

```
Train size:  435 Test size:  109
```

Submission 1 of 12: Progress mark (10)
Screen shot preparation code.

Train and evaluate the model.

Create a new text cell after **Train/Test Data Split** cell and add in the following

```
### Train and evaluate a Linear Regression Model
```

Create a new code cell and add in the following:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

1. These imports give us our model and the metric functions.

Add the following lines to the code cell:

```
X = train_set[['height']]
y = train_set['weight']

X_test = test_set[['height']]
y_test = test_set['weight']
```

1. Predict on the feature height.
2. Use weight as the target feature
3. Test the trained model height
4. Use weight as the target feature

Add the following lines to the code cell:

```
lr_model = LinearRegression()
lr_model.fit(X,y)
```

1. Create an instance of a decision tree model.
2. Train the model on the training data

Add the following lines to the code cell:

```
y_pred = lr_model.predict(X)
print('Results for linear regression on training data')
print('  Default settings')
print('Internal parameters:')
print('   Bias is ', lr_model.intercept_)
print('   Coefficients', lr_model.coef_)
print('   Score', lr_model.score(X,y))

print('MAE is  ', mean_absolute_error(y, y_pred))
print('RMSE is ', np.sqrt(mean_squared_error(y, y_pred)))
print('MSE is ', mean_squared_error(y, y_pred))
print('R^2    ', r2_score(y,y_pred))
```

1. Get the predictions of the model on the training data
2. Print the internal parameters for the regression (Bias plus a coefficient for each of the input parameters. Notice the training _.
3. Use the model to compute a score on the training data. This should be the same as r2.
4. Compute the standard metrics. RMSE gives a good estimate of the spread of the predictions .

Run the code cell and you should see something like

```
Results for linear regression on training data
  Default settings
Internal parameters:
   Bias is  -34.33484184267892
   Coefficients [0.50625269]
   Score 0.8892970528522413
MAE is   3.9868013743337323
RMSE is  4.901419255121661
MSE is  24.023910714477378
R^2     0.8892970528522413
```

Go to the end of the note book and add in a markdown cell. This is where we are going to store our results.

```
# Results
Basic results for our regression models to predict weight on
the Howell data.

| Model | Training Features | Set | RMSE | R2 |
|:---|:---|:---|:---|:---|
|Linear Regression|Height|Training|4.90|88.93|
```

1. We are going to record the RMSE and r2 scores as a baseline. Depending on the goals of your analysis, you may want to record other values as well.
2. I am going to record RMSE values to two decimal places. R2 scores will be converted t percentages and rounded to two places.

We now want to get the performance of the model on the test data. Add the following lines to the code cell:

```
y_test_pred = lr_model.predict(X_test)
print()
print('Results for linear regression on test data')

print('MAE is  ', mean_absolute_error(y_test, y_test_pred))
print('RMSE is ', np.sqrt(mean_squared_error(y_test,
y_test_pred)))
print('MSE is ', mean_squared_error(y_test, y_test_pred))
print('R^2    ', r2_score(y_test,y_test_pred))
```

1. This is a modified copy of the performance reporting code for the training set. Feel free to copy and modify or retype. If you miss a renaming, you will most likely get an error telling you that you have a size mismatch.

Run the code cell. In addition to the previous performance report on the training data, you should get something like:

```
Results for linear regression on test data
MAE is  4.268058841968444
RMSE is 5.308233145876227
MSE is  28.177339130979025
R^2    0.8677460846087206
```

Submission 2 of 12: Progress mark (10)
Screen shot of train/test performance.

Submission 3 of 12: Analysis (5)
Looking at the performance of the model on the training and test sets and make an argument on whether the model is underfitting the training set.

Record the results in the final cell by adding another row in the table

```
|Linear Regression|Height|Test|5.31|86.77|
```

Graph the Model.

Create a new text cell after the **Train and Evaluate Linear Regression Model** cell and add in the following

```
### Plot linear regression model
 Using height to predict weight
```

Create a new code cell and copy the code from the **A Better Plot** cell. Get rid of the extra plots that are commented out. Run the cell to make sure it still works.

We are now going to generate a list of heights and feed those values into the model to get corresponding weight predictions. Add the following code before the plot.

```
min_height = height.min()
max_height = height.max()
points = 200
step_by = (max_height - min_height)/(points-1)
```

1. Find the minimum value in the training set heights.
2. Find the maximum value in the training set height.
3. Set the number of points we want in the line plot. More will look smoother. For this linear regression model, two points would work.
4. Determine the spacing so it is even.

We are ready to use the model to compute the points. Add the following code before the plot

```
x_values = [min_height + i*step_by for i in range(0, points)]
inputs = [[x] for x in x_values]
y_values = lr_model.predict(inputs)
```

1. This is a list comprehension. We use I values from the range starting at 0 and ending at points-1. The x values are computed from i.
2. This is also a list comprehension. We use x values from the list we just created. The model is expecting an array of inputs, each of which is an array. We convert the x value into and array holding it.
3. The model uses each input to compute the predicted value which comes back as an array.

Time to create the line plot. Add the following code just before the show.

```
plt.plot(x_values, y_values, c='red')
```

1. Do a line plot. It will overlay on the points. Make the color of the line red for better visibility.

Submission 4 of 12: Progress mark (5)
Screen shot of linear fit

Submission 5 of 12: Analysis (5)
Looking at the plot, can we improve the fit of the model significantly by adding more training instances? Does this indicate under or over fitting by the model?

Add a feature to the model.

Copy the markdown cell **Train and Evaluate a Linear Regression Model** and the corresponding code cell before the **Results** cell.

Change the markdown to the following

```
### Add a Feature to the Model
```

In the code cell, we want to change the model so that it uses both height and age in the prediction. There are only a few lines we need to change

```
X = train_set[['height', 'age']]
y = train_set['weight']

X_test = test_set[['height', 'age']]
y_test = test_set['weight']
```

1. We add a new feature to pull out of the dataframe on both training and test.

Add the following two new lines to the code cell. This won't affect the results, but it is helpful to mark the changes for this run.

```
print('Results for linear regression on training data')
print('Input: Height, age')
```

```
print('Results for linear regression on test data')
print('Input: Height, age')
```

We have a decision to make with respect to the new model. Do we want to reuse the model name (which is lr_model) or do we want to give this model a new name. In this case, I am going to give it a new name. The main reason for doing so, is that the graph code assumes that the model it is getting is just using a single feature as input. If we reuse the name and then try to run the graphing code it will break. Everywhere in the code cell for the new model, change lr_model to lrTwoInput_model.

Run the code cell. You should get something like:

Results for linear regression on test data
Results for linear regression on training data
Input: Height, age
Default settings
Internal parameters:
Bias is -32.981786351678274
Coefficients [0.48959129 0.03198353]
Score 0.8903963296817052
MAE is 3.9844579179380495
RMSE is 4.877023070139709
MSE is 23.78535402667495
R^2 0.8903963296817052

Results for linear regression on test data
Input: Height, age
MAE is 4.196716851946104
RMSE is 5.183466125624855
MSE is 26.868321075500344
R^2 0.8738901268956872

Record the results in the final cell by adding two rows in the table

```
|Linear Regression|Height,Age|Training|4.88|89.04|
|Linear Regression|Height,Age|Test|5.18|87.39|
```

Submission 6 of 12: Explore (10)
Explain: Did adding the feature improve the performance of the model? Propose an explanation for the results.

Non-Linear fitting.

There is a really common trick you can play that will allow you to fit your data with something different to a line. We will add in extra features. For example, if our data is x,y and you want to fit a cubic, you compute square(x) and cube(x) as new features. The function to fit is

$$y = \omega_0 + \omega_1 \times x + \omega_2 \times square(x) + \omega_3 \times \text{cube}(x)$$

which is now linear in the three features. These extra computed terms can be a problem as they may grow quite large compared with the original terms. While we can add other kinds of terms, we will demonstrate the use with polynomials.

Add Polynomial Features to the Model.

Copy the markdown cell **Train and Evaluate a Linear Regression Model** and the corresponding code cell before the **Results** cell.

Change the markdown to the following

```
### Use Polynomial Regression
```

In the code cell, we are going to change the data set by adding in extra features. In particular, we are going to add in terms like height squared, height cubed, etc…
While we could do this manually, it is convenient to use the built in capability that SciKit provides.

At the front of the code cell add the following lines

```
from sklearn.preprocessing import PolynomialFeatures
power = 3
poly_process = PolynomialFeatures(degree=power, include_bias=False)
```

1. Get the object that will manipulate the data.
2. Set the power of the highest added feature. Using 3 now, but can change this later. Our model already has a bias term, so we don't need to add one to the transformed data.
3. This is the object that will process the data

Add the following lines so that we create the transformed data

```
X = train_set[['height']]
y = train_set['weight']
X_poly = poly_process.fit_transform(X)

X_test = test_set[['height']]
y_test = test_set['weight']
X_poly_test = poly_process.fit_transform(X_test)
```

1. X_poly is the transformed data set for training.
2. X_poly_test is the transformed data set for testing.
3. We don't need to do anything to the target.

Change the rest of the code cell to use X_poly and X_poly_test instead of X and X_test.

```
lr_model = LinearRegression()
lr_model.fit(X_poly,y)

y_pred = lr_model.predict(X_poly)
print('Results for linear regression on training data')
print('Polynomial regression with degree ', power)
print('  Default settings')
print('Internal parameters:')
print('   Bias is ', lr_model.intercept_)
print('   Coefficients', lr_model.coef_)
print('   Score', lr_model.score(X_poly,y))

print('MAE is  ', mean_absolute_error(y, y_pred))
print('RMSE is ', np.sqrt(mean_squared_error(y, y_pred)))
print('MSE is ', mean_squared_error(y, y_pred))
print('R^2    ', r2_score(y,y_pred))

y_test_pred = lr_model.predict(X_poly_test)
```

We are going to reuse the model name this time, but we still need to specialize the graphing code to use the polynomial features preprocessing.

Run the code cell. You should get something like: (Notice that we get 3 coefficients now and that the higher degree terms have smaller coefficients.)

```
Results for linear regression on training data
Polynomial regression with degree  3
  Default settings
Internal parameters:
   Bias is  38.755672491337386
   Coefficients [-9.68874202e-01  8.36011255e-03 -1.20345929e-05]
   Score 0.9327619061923518
MAE is   2.9208443646670466
RMSE is  3.819882089468078
MSE is  14.59149917743901
R^2    0.9327619061923518

Results for linear regression on test data
MAE is   3.2929418948562046
RMSE is  4.305579019074042
MSE is  18.538010689490587
R^2    0.9129894953581681
```

Record the results in the final cell by adding two rows in the table

```
|Polynomial Regression degree 3|Height|Training|3.82|83.28|
|Polynomial Regression degree 3|Height|Test|4.31|91.30|
```

Graph the Polynomial Curve.

Create a new text cell after the **Use Polynomial Regression** cell and add in the following

```
### Polynomial Regress Graph
```

Create a new code cell and copy the code from the code cell after **Plot linear regression model**. We just need to make a couple small changes to work with the model.

The polynomial model is expecting more features, so we need to replace the line

```
y_values = lr_model.predict(inputs)
```

With a couple lines .

```
inputs_poly = poly_process.fit_transform(inputs)
y_values = lr_model.predict(inputs_poly)
```

1. Use the transform from the previous cell to add features into inputs.
2. Use the transformed input to make the predictions.

Submission 7 of 12: Progress mark (10)
Screen shot of scatter graph and cubic fit.

Submission 8 of 12: Explore (10)
Explain: Did the polynomial fit do better? Where does it fit the best?
.

Try a higher degree.

Change the power to be 8.

Run the code cell. You should get something like:

Results for linear regression on training data
Polynomial regression with degree 8

```
 Default settings
Internal parameters:
  Bias is  30.63239107840527
  Coefficients [-1.05821512e-06 -5.84608126e-05 -1.70146004e-03  6.23656892e-05
 -9.32819301e-07  7.00941575e-09 -2.62048570e-11  3.88233282e-14]
  Score 0.9379323643080313
MAE is  2.692252534691684
RMSE is  3.670074348601383
MSE is  13.469445724261865
R^2    0.9379323643080313

Results for linear regression on test data
MAE is  3.1025511400431527
RMSE is  4.25758864994199
MSE is  18.127061112114855
R^2    0.9149183393268514
```

Record the results in the final cell by adding two rows in the table

```
|Polynomial Regression degree 8|Height|Training|3.67|93.79|
|Polynomial Regression degree 8|Height|Test|4.26|91.50|
```

Submission 9 of 12: Progress mark (10)
Screen shot of scatter graph and degree 8 polynomial fit.

Submission 10 of 12: Explore (5)
Compare the results with power=3 and power=8. Argue on whether the increase in degree is warranted by improved performance of the model.
.

Use a Regularized Model .

In this section, we are going to use a model that is less susceptible to overfitting. The elastic net is a combination of Ridge and Lasso. One of the differences for this model is that it can not solve directly while training, but must perform a search.

Copy the markdown cell **Use Polynomial Regression** and the corresponding code cell before the **Results** cell.

Change the markdown to the following

```
### Elastic Net with Poly Features
```

Since we don't have that many input features for the regular regression, we will use the polynomial features code that we just developed.

At the front of the code cell add the following line

```
from sklearn.linear_model import ElasticNet
```

1. Get the model.

Change the following line

```
lr_model = LinearRegression()
```

to

```
reg_lr_model = ElasticNet(alpha=0.3, l1_ratio=0.5)
```

1. Get the model. We are going to use a different name for our model. It will have a moderate amount regularization controlled by alpha. The mix of Ridge to Lasso is 50%.

Change the rest of the code cell to use reg_lr_model instead of lr_model.

Change the print statements to

```
print('Results for elastic net on training data')
```

```
print('Results for elastic net on test data')
```

Run the code cell. You should get something like the following. Because this involved a randomized search, you should not expect the values to be exactly the same.

```
Results for elastic net on training data
Polynomial regression with degree  8
 Default settings
Internal parameters:
  Bias is  18.84611340392237
  Coefficients [-2.55756771e-01  6.87204298e-04  8.52528900e-06  3.67563933e-08
 1.17760939e-10  1.01431642e-13 -2.55533600e-15 -3.04757085e-17]
  Score 0.9349092303329695
MAE is  2.8731176934054683
RMSE is  3.758391044295953
MSE is  14.125503241844026
R^2    0.9349092303329695
```

```
Results for elastic net on test data
MAE is  3.2252187133698116
RMSE is  4.2163950622270505
MSE is  17.777987320772656
R^2  0.9165567614450963
/Users/ios/ds-venv/lib/python3.9/site-packages/sklearn/linear_model/_coordinate_descent.py:529: Conver
genceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap:
3091.1645496107085, tolerance: 9.440038797566226
  model = cd_fast.enet_coordinate_descent(
```

Record the results in the final cell by adding two rows in the table

```
|Elastic Net degree 8|Height|Training|3.76|93.49|
|Elastic Net degree 8|Height|Test|4.22|91.66|
```

Graph the Elastic Net Curve.

Create a new text cell after the **Use Polynomial Regression** cell and add in the following

```
### Elastic Net Graph
```

Copy the code cell after **Polynomial Regress graph** and place it just after the cell you created.

This graph code is already set up to use polynomial features. We just need to make sure that it is using the correct model.

Change the line

```
y_values = lr_model.predict(inputs_poly)
```

With

```
y_values = reg_lr_model.predict(inputs_poly)
```

Submission 11 of 12: Progress mark (10)
Screen shot of scatter graph and degree 8 polynomial fit with Elastic net.

Submission 12 of 12: Explore (5)

Compare the coefficients for the regular fit with power=8 to the Elastic net. Which coefficients were reduced (closer to zero). What does this tell us about our data?

(internal parameters for the model are bias, Linear coefficient, Quadratic coefficient, Cubic coefficient, Quartic coefficient, degree 5 coefficient, degree 6 coefficient, degree 7 coefficient, degree 8 coefficient)

.