Regression with Linear Algebra - Lab

Introduction

In this lab, we shall apply regression analysis using simple matrix manipulations to fit a model to given data, and then predict new values for previously unseen data. We shall follow the approach highlighted in previous lesson where we used numpy to build the appropriate matrices and vectors and solve for the β (unknown variables) vector. The beta vector will be used with test data to make new predictions. We shall also evaluate how good our model fit was.

In order to make this experiment interesting. We shall use NumPy at every single stage of this experiment i.e. loading data, creating matrices, performing test train split, model fitting and evaluations.

Objectives

You will be able to:

- Use linear algebra to apply simple regression modeling in Python and NumPy only
- Apply train/test split using permutations in NumPy
- Use matrix algebra with inverses and dot products to calculate the beta
- Make predictions from the fitted model using previously unseen input features
- Evaluate the fitted model by calculating the error between real and predicted values

First let's import necessary libraries

In [ ]:
```
import csv # for reading csv file
import numpy as np
```

Dataset

The dataset we will use for this experiment is "**Sales Prices in the City of Windsor, Canada**", something very similar to the Boston Housing dataset. This dataset contains a number of input (independent) variables, including area, number of bedrooms/bathrooms, facilities(AC/garage) etc. and an output (dependent) variable, **price**. We shall formulate a linear algebra problem to find linear mappings from input to out features using the equation provided in the previous lesson.

This will allow us to find a relationship between house features and house price for the given data, allowing us to find unknown prices for houses, given the input features.

A description of dataset and included features is available at THIS LINK

In your repo, the dataset is available as `windsor_housing.csv` containing following variables:

there are 11 input features (first 11 columns):

```
lotsize    bedrooms    bathrms    stories    driveway    recroom    fullbase    gashw    airco    garagepl    prefarea
```

and 1 output feature i.e. **price** (12th column).

The focus of this lab is not really answering a preset analytical question, but to learn how we can perform a regression experiment, similar to one we performed in statsmodels, using mathematical manipulations. So we we wont be using any Pandas or statsmodels goodness here. The key objectives here are to a) understand regression with matrix algebra, and b) Mastery in NumPy scientific computation.

Stage 1: Prepare Data for Modeling

Let's give you a head start by importing the dataset. We shall perform following steps to get the data ready for analysis:

- Initialize an empty list `data` for loading data
- Read the csv file containing complete (raw) `windsor_housing.csv` . Use `csv.reader()` for loading data.. Store this in `data` one row at a time.
- Drop the first row of csv file as it contains the names of variables (header) which won't be used during analysis (keeping this will cause errors as it contains text values).
- Append a column of all 1s to the data (bias) as the first column
- Convert `data` to a numpy array and inspect first few rows

NOTE: `read.csv()` would read the csv as a text file, so we must convert the contents to float at some stage.

In [ ]:
```
# Create Empty lists for storing X and y values.
data = []

#Read the data from the csv file
with open('windsor_housing.csv') as f:
    raw = csv.reader(f)
    # Drop the very first line as it contains names for columns - not actual data.
    next(raw)
    # Read one row at a time. Append one to each row
    for row in raw:
        ones = [1.0]
        for r in row:
            ones.append(float(r))
        # Append the row to data
        data.append(ones)
```

```
data = np.array(data)
data[:5,:]

# array([[1.00e+00, 5.85e+03, 3.00e+00, 1.00e+00, 2.00e+00, 1.00e+00,
#         0.00e+00, 1.00e+00, 0.00e+00, 0.00e+00, 1.00e+00, 0.00e+00,
#         4.20e+04],
#        [1.00e+00, 4.00e+03, 2.00e+00, 1.00e+00, 1.00e+00, 1.00e+00,
#         0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00,
#         3.85e+04],
#        [1.00e+00, 3.06e+03, 3.00e+00, 1.00e+00, 1.00e+00, 1.00e+00,
#         0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00,
#         4.95e+04],
#        [1.00e+00, 6.65e+03, 3.00e+00, 1.00e+00, 2.00e+00, 1.00e+00,
#         1.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00,
#         6.05e+04],
#        [1.00e+00, 6.36e+03, 2.00e+00, 1.00e+00, 1.00e+00, 1.00e+00,
#         0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00,
#         6.10e+04]])
```

Step 2: Perform a 80/20 test train Split

Explore NumPy's official documentation to manually split a dataset using `numpy.random.shuffle()` , `numpy.random.permutations()` or using simple resampling method.

- Perform a **RANDOM** 80/20 split on data using a method of your choice , in NumPy using one of the methods above.
- Create x_test, y_test, x_train and y_train arrays from the split data.
- Inspect the contents to see if the split performed as expected.

In [ ]:

```
# PErform an 80/20 split
training_idx = np.random.randint(data.shape[0], size=round(546*.8))
test_idx = np.random.randint(data.shape[0], size=round(546*.2))
training, test = data[training_idx,:], data[test_idx,:]

# Check the shape of datasets
print ('Raw data Shape: ', data.shape)
print ('Train/Test Split:', training.shape, test.shape)

# Create x and y for test and training sets
x_train = training[:,:-1]
y_train = training [:,-1]

x_test = test[:,:-1]
y_test = test[:,-1]

# Check the shape of datasets
print ('x_train, y_train, x_test, y_test:', x_train.shape, y_train.shape, x_test.shape, y_test.shape)


# Raw data Shape:  (546, 13)
# Train/Test Split: (437, 13) (109, 13)
# x_train, y_train, x_test, y_test: (437, 12) (437,) (109, 12) (109,)
```

Step 3: Calculate the `beta`

With our X and y in place, We can now compute our beta values with x_train and y_train as:

β = **(x_train^T . x_train)$^{-1}$. x_train^T . y_train**

- Using numpy operations (transpose, inverse) that we saw earlier, compute the above equation in steps.
- Print your beta values

In [ ]:

```
# Calculate Xt.X and Xt.y for beta = (XT . X)-1 . XT . y - as seen in previous lessons
Xt = np.transpose(x_train)
XtX = np.dot(Xt,x_train)
Xty = np.dot(Xt,y_train)

# Calculate inverse of Xt.X
XtX_inv = np.linalg.inv(XtX)

# Take the dot product of XtX_inv with Xty to compute beta
beta = XtX_inv.dot(Xty)

# Print the values of computed beta
print(beta)

# [-3.07118956e+03  2.13543921e+00  4.04283395e+03  1.33559881e+04
#   5.75279185e+03  7.82810082e+03  3.73584043e+03  6.51098935e+03
#   1.28802060e+04  1.09853850e+04  6.14947126e+03  1.05813305e+04]
```

Step 4: Make Predictions

Great , we now have a set of coefficients that describe the linear mappings between X and y. We can now use the calculated beta values with the test datasets that we left out to calculate y predictions. For this we need to perform the following tasks:

Now we shall all features in each row in turn and multiply it with the beta computed above. The result will give a prediction for each row which we can append to a new array of predictions.

$\hat{y}$ = **x.**β = β_0 + β_1 . x_1 + β_2 . x_2 + .. + β_m . x_m

- Create new empty list (y_pred) for saving predictions.
- For each row of x_test, take the dot product of the row with beta to calculate the prediction for that row.
- Append the predictions to y_pred.
- Print the new set of predictions.

In [ ]:
```
# Calculate and print predictions for each row of X_test
y_pred = []
for row in x_test:
    pred = row.dot(beta)
    y_pred.append(pred)
```

Step 5: Evaluate Model

Visualize Actual vs. Predicted

This is exciting, so now our model can use the beta value to predict the price of houses given the input features. Let's plot these predictions against the actual values in y_test to see how much our model deviates.

In [ ]:
```
# Plot predicted and actual values as line plots
import matplotlib.pyplot as plt
from pylab import rcParams
rcParams['figure.figsize'] = 15, 10
plt.style.use('ggplot')

plt.plot(y_pred, linestyle='-', marker='o', label='predictions')
plt.plot(y_test, linestyle='-', marker='o', label='actual values')
plt.title('Actual vs. predicted values')
plt.legend()
plt.show()
```

This doesn't look so bad, does it ? Our model, although isn't perfect at this stage, is making a good attempt to predict house prices although a few prediction seem a bit out. There could a number of reasons for this. Let's try to dig a bit deeper to check model's predictive abilities by comparing these prediction with actual values of y_test individually. That will help us calculate the RMSE value (Root Mean Squared Error) for our model.

Root Mean Squared Error

Here is the formula for this again.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

- Initialize an empty array `err`.
- for each row in y_test and y_pred, take the squared difference and append error for each row in err array.
- Calculate RMSE from `err` using the formula shown above.

In [ ]:
```
# Calculate RMSE
err = []
for pred,actual in zip(y_pred,y_test):
    sq_err = (pred - actual) ** 2
    err.append(sq_err)
mean_sq_err = np.array(err).mean()
root_mean_sq_err = np.sqrt(mean_sq_err)
root_mean_sq_err
```

Normalized Root Mean Squared Error

The above error is clearly in terms of the dependant variable i.e. the final house price. We can also use a normlized mean squared error in case of multiple regression which can be calculated from RMSE using following formula:

- Calculate normalized Root Mean Squared Error

$$NRMSE = \frac{RMSE}{\max_i y_i - \min_i y_i}$$

In [ ]:
```
root_mean_sq_err/(y_train.max() - y_train.min())

# 0.08495931785402822
```

SO there it is. A complete multiple regression analysis using nothing but numpy. Having good programming skills in numpy would allow to dig deeper into analytical algorithms in machine learning and deep learning. Using matrix multiplication techniques we saw here, we can easily build a whole neural network from scratch.

Level up - Optional

- Calculated the R_squared and adjusted R_squared for above experiment.

- Plot the residuals (similar to statsmodels) and comment on the variance and heteroscedasticity.
- Run the experiment in statsmodels and compare the performance of both approaches in terms of computational cost.

Summary

So there we have it. A predictive model for predicting house prices in a given dataset. Remember this is a very naive implementation of regression modeling. The purpose here was to get an introduction to the applications of linear algebra into machine learning and predictive analysis. We still have a number of shortcomings in our modeling approach and we can further apply a number of data modeling techniques to improve this model.