Course: CS550 - Machine Learning and Business Intelligence
Faculty: Prof. Henry Chang

Machine Learning: Linear Regression using the Normal Equation

Name: Haripriya A
ID: 19579

NORTHWESTERN POLYTECHNIC
UNIVERSITY

Table of Contents

Introduction

- Machine learning is a method of data analysis investigation that automates logical model structure. Machine learning algorithms are largely used to predict, classify, or cluster.
- Machine Learning
    * Supervised Learning
        - Regression : Linear Regression Algorithm
        - Classification : KNN Algorithm
    * Unsupervised Learning
        - Clustering : K-Mean Algorithm

About Linear Regression- Normal Equation

- Linear Regression Model : A regression is a statistical analysis assessing the association between two variables. It is used to find the relationship between two variables.
- To train a Linear Regression model, we need to find the value of **θ.** To find the value of **θ** that minimizes the cost function, there is a closed-form solution—in other words, a mathematical equation that gives the result directly. This is called the Normal Equation
- We modify the code in Linear regression using the Normal Equation.

Data from CSV

- Using google colab import files, we import the data from csv file

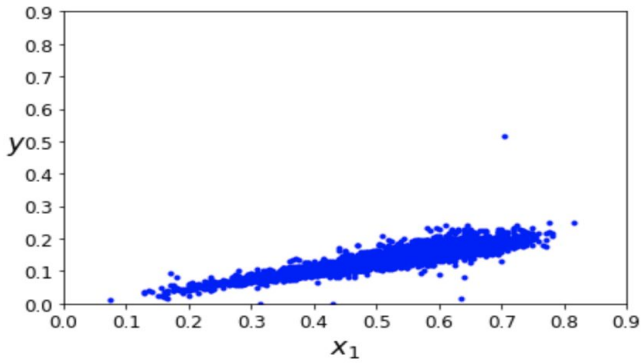
```
from google.colab import files
uploaded  = files.upload()

import io
abalone = pd.read_csv(
    io.BytesIO(uploaded['abalone_train.csv']),
    names=["Length", "Diameter", "Height", "Whole weight", "Shucked weight",
           "Viscera weight", "Shell weight", "Age"])
```

```
Choose Files abalone_train.csv
• abalone_train.csv(text/csv) - 145915 bytes, last modified: 5/30/2021 - 100% done
Saving abalone_train.csv to abalone_train (2).csv
```

Data Plot

```
plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.axis([0, 0.9, 0, 0.9])
save_fig("generated_data_plot")
plt.show()
print(len(X))
```

```
Saving figure generated_data_plot
```



- we use the inv() function from NumPy's linear algebra module (np.linalg) to compute the inverse of a matrix, and the dot() method for matrix multiplication.
- The function that we used to generate the data is $y = 4 + 3x_1 +$ Gaussian noise.
- We got for $\theta_0 = 0.01$ and $\theta_1 = 0.28$

```
[ ]  X_b = np.c_[np.ones((3320, 1)), X]  # add x0 = 1 to each instance
     theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
```

```
[ ]  theta_best
```

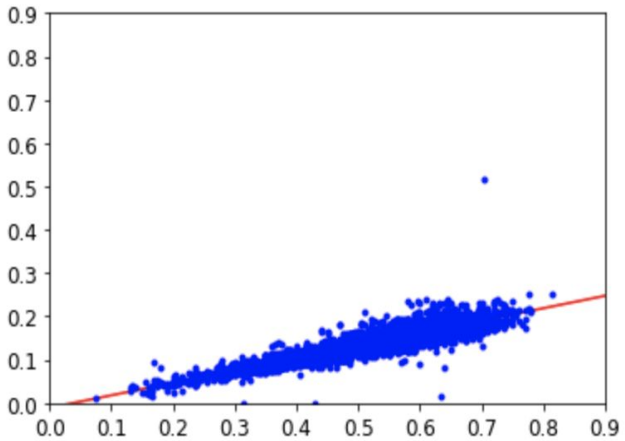
```
array([[-0.0108267 ],
       [ 0.28716253]])
```

```
     X_new = np.array([[0], [2]])
     X_new_b = np.c_[np.ones((2, 1)), X_new]  # add x0 = 1 to each instance
     print(X_new_b)
     y_predict = X_new_b.dot(theta_best)
     y_predict
```

```
[[1. 0.]
 [1. 2.]]
array([[-0.0108267 ],
       [ 0.56349837]])
```

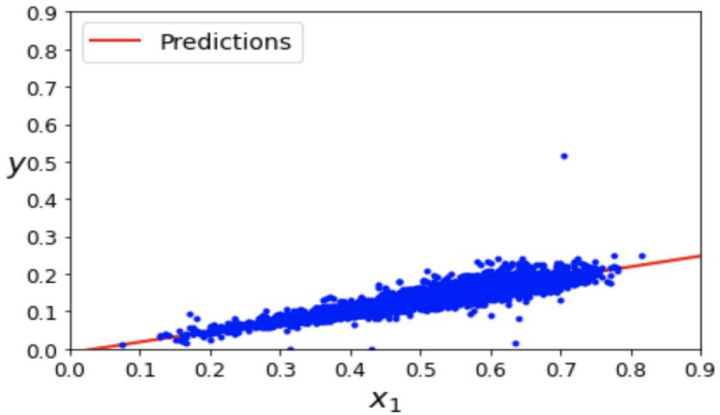
Plot for Model's Predictions

```
[ ] plt.plot(X_new, y_predict, "r-")
    plt.plot(X, y, "b.")
    plt.axis([0, 0.9, 0, 0.9])
    plt.show()
```



```
plt.plot(X_new, y_predict, "r-", linewidth=2, label="Predictions")
plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14)
plt.axis([0, 0.9, 0, 0.9])
save_fig("linear_model_predictions_plot")
plt.show()
```

```
Saving figure linear_model_predictions_plot
```



- Performing Linear Regression using Scikit-Learn is simple:2
- The LinearRegression class is based on the scipy.linalg.lstsq() function (the name stands for “least squares”), which you could call directly.
- This function computes $\hat{\theta}=X^{+}y$, where X^{+} is the *pseudoinverse* of **X** (specifically, the Moore-Penrose inverse). We can use np.linalg.pinv() to compute the pseudoinverse directly.

```
[ ]  from sklearn.linear_model import LinearRegression

     lin_reg = LinearRegression()
     lin_reg.fit(X, y)
     lin_reg.intercept_, lin_reg.coef_
```

```
(array([-0.0108267]), array([[0.28716253]]))
```

```
lin_reg.predict(X_new)
```

```
array([[-0.0108267 ],
       [ 0.56349837]])
```

```
[ ]  theta_best_svd, residuals, rank, s = np.linalg.lstsq(X_b, y, rcond=1e-6)
     theta_best_svd
```

```
array([[-0.0108267 ],
       [ 0.28716253]])
```

```
[ ]  np.linalg.pinv(X_b).dot(y)
```

```
array([[-0.0108267 ],
       [ 0.28716253]])
```

Conclusion

- Once we trained Linear Regression model using the Normal Equation or any other algorithm, predictions are very fast: the computational complexity is linear with regard to both the number of instances we want to make predictions on and the number of features.

References

- https://colab.research.google.com/github/ageron/handson-ml2/blob/master/04_training_linear_models.ipynb#scrollTo=5FlyaQhuuxPi
- https://npu85.npu.edu/~henry/npu/classes/hands_on_ml_with_schikit_2nd/train_model/slide/The_Normal_Equation.html
- https://npu85.npu.edu/~henry/npu/classes/machine_learning/colab/slide/get_start.html#3%20Ways%20to%20Load%20CSV%20files%20into%20Colab

Google slides URL :

https://docs.google.com/presentation/d/1YKZSR2Nm5-7cuMZI-xCppoGgUmam3gdqGdWRm9nwTBg/edit?usp=sharing