CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

# Machine Learning

## Machine Learning

Lecture: Linear and Multivariate Regression

Ted Scully

# Contents

# Multiple (Multi-variate) Linear Regression

- In simple linear regression we look at the relationship between two variables (a single feature and a target variable) such as the relationship between office floor space and weekly rental values or a patients weight and their blood pressure.

- Now let's make the more realistic assumption that we have other features that will impact the value of the target.

- For example let's assume we build a model for predicting rental prices. We might incorporate features such as the **energy rating** of the building, the **broadband speed** and the **floor number** of the office.

# Multiple (Multi-variate) Linear Regression

- Let's assume we have multiple features $f_1$, $f_2$, $f_3$, … $f_n$.

| $f_1$ | $f_2$ | $f_3$ | $f_4$ | …. | $f_n$ | Y |
|---|---|---|---|---|---|---|
| .. | .. | .. | .. | .. | .. | $y^1$ |
| .. | .. | .. | .. | .. | .. | $y^2$ |
| .. | .. | .. | .. | .. | .. | $y^3$ |
| .. | .. | .. | .. | .. | .. | .. |

- We can build a multi-variate linear regression model using a similar approach to the one we used in linear regression.
- Our MLR model can take in a vector of n feature values <$x_1$, $x_2$, $x_3$, $x_4$, .. $x_n$>

$$h(x) = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3 + ..... + \lambda_n x_n + b$$

- How many tuneable parameters are there in our MLR model above?

# Multiple (Multi-variate) Linear Regression

- We can express the square error cost function for multi-variate linear regression as shown below
  - Rather than listing each $\lambda$ value we represent them as W (weights).

$$C(W, b) = \frac{1}{m+2} \sum_{i=1}^{m} ((h(\boldsymbol{x^i}) - y^i))^2$$

- Where $x^i$ is a **vector** of input features for the i$^{th}$ training example. (Remember previously in linear regression $x^i$ was just a single value for the single input feature).

- We will introduce one additional notational convention, $\boldsymbol{x_j^i}$ , which is the the i$^{th}$ data element for feature j. If this is confusing think of it in terms of your dataset. Then $\boldsymbol{x_j^i}$ would refer to the data item in the **i$^{th}$ row** and the **j$^{th}$ column**.

# Multiple (Multi-variate) Linear Regression

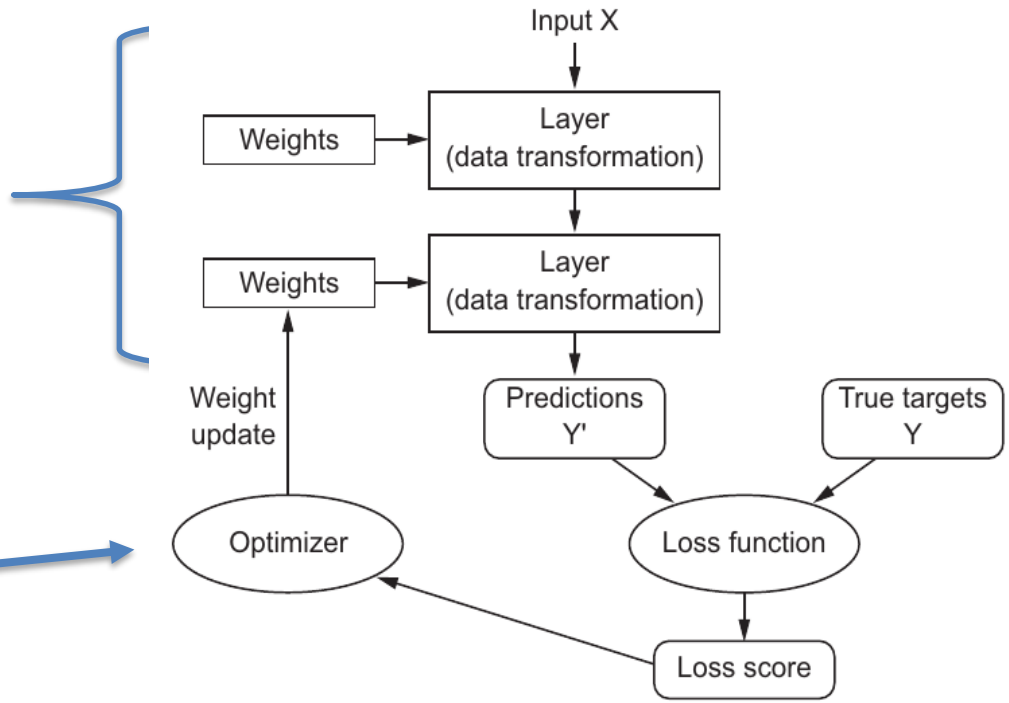
| | $f_1$ | $f_2$ | $f_3$ | $f_4$ | .... | $f_n$ | Y |
|---|---|---|---|---|---|---|---|
| $x^1$ → | $x_1^1$ | $x_2^1$ | $x_3^1$ | $x_4^1$ | .. | $x_n^1$ | $y^1$ |
| $x^2$ → | $x_1^2$ | $x_2^2$ | $x_3^2$ | $x_4^2$ | .. | $x_n^2$ | $y^2$ |
| $x^3$ → | $x_1^3$ | $x_2^3$ | $x_3^3$ | $x_4^3$ | .. | $x_n^3$ | $y^3$ |
| | .. | .. | .. | .. | .. | .. | .. |

- Where $x^i$ is a **vector** of input features for the i[th] training example. (Remember previously in linear regression $x^i$ was just a single value for the single input feature).

- We will introduce one additional notational convention, $\boldsymbol{x_j^i}$ , which is the the i[th] data element for feature j. If this is confusing think of it in terms of your dataset. Then $\boldsymbol{x_j^i}$ would refer to the data item in the **i[th] row** and the **j[th] column**.

| $f_1$ | $f_2$ | $f_3$ | $f_4$ | .... | $f_n$ | Y |
|---|---|---|---|---|---|---|
| $x_1^1$ | $x_2^1$ | $x_3^1$ | $x_4^1$ | .. | $x_n^1$ | $y^1$ |
| $x_1^2$ | $x_2^2$ | $x_3^2$ | $x_4^2$ | .. | $x_n^2$ | $y^2$ |
| $x_1^3$ | $x_2^3$ | $x_3^3$ | $x_4^3$ | .. | $x_n^3$ | $y^3$ |
| .. | .. | .. | .. | .. | .. | .. |

$$h(x) = \lambda_1 x_1 + \ldots\ldots + \lambda_n x_n + b$$




***Gradient Descent Optimizer***

$$\frac{1}{m+2}\sum_{i=0}^{m}((h(\boldsymbol{x^i}) - y^i))^2$$

# Multiple (Multi-variate) Linear Regression

- As we did with linear regression we can use gradient descent to find the optimal values for each $\lambda_i$ and b

- **Repeat {**

$$\lambda_j = \lambda_j - \alpha \frac{\partial}{\partial \lambda_j} C(W, b)$$

$$b = b - \alpha \frac{\partial}{\partial b} C(W, b)$$

**}**

Where:

$$\frac{\partial}{\partial \lambda_j} C(W, b) = \frac{1}{2m} \sum_{i=0}^{m} ((h(x^i) - y^i))(x_j^i)$$

$$\frac{\partial}{\partial b} C(W, b) = \frac{1}{2m} \sum_{i=0}^{m} ((h(x^i) - y^i))$$

Remember in **linear regression** we updated $\lambda_1$ as:

$$\frac{\partial}{\partial \lambda_1} C(\lambda_1, b) = \frac{1}{2m} \sum_{i=0}^{m} ((h(x^i) - y^i))(x^i)$$

We could roughly describe this as multiplying the prediction error <u>for each training</u> example (blue) by the feature value for that training example (red). As this is linear regression there is only one feature.

In multiple linear regression we have **multiple features** and consequently **multiple $\lambda$ values** (one for each feature). The derivative with respect to any $\lambda$ for an MLR can be defined as follows (notice it is the same as above except that j indicates the specific feature we are focused on (remember $\lambda_j$ is associated with *feature j* ).

Therefore, if we are updating $\lambda_1$ then we multiply the prediction error for each example by the feature 1 value for that example. ):

$$\frac{\partial}{\partial \lambda_1} C(\lambda_1) = \frac{1}{2m} \sum_{i=0}^{m} ((h(x^i) - y^i))(x_1^i)$$

We can express this more generally for any $\lambda_i$ as:

$$\frac{\partial}{\partial \lambda_j} C(\lambda_j) = \frac{1}{2m} \sum_{i=0}^{m} ((h(x^i) - y^i))(x_j^i)$$

# Multiple (Multi-variate) Linear Regression

**Where** $\frac{\partial}{\partial \lambda_j} C(W, b) = \frac{1}{2m} \sum_{i=0}^{m} ((h(x^i) - y^i))(x_j^i)$

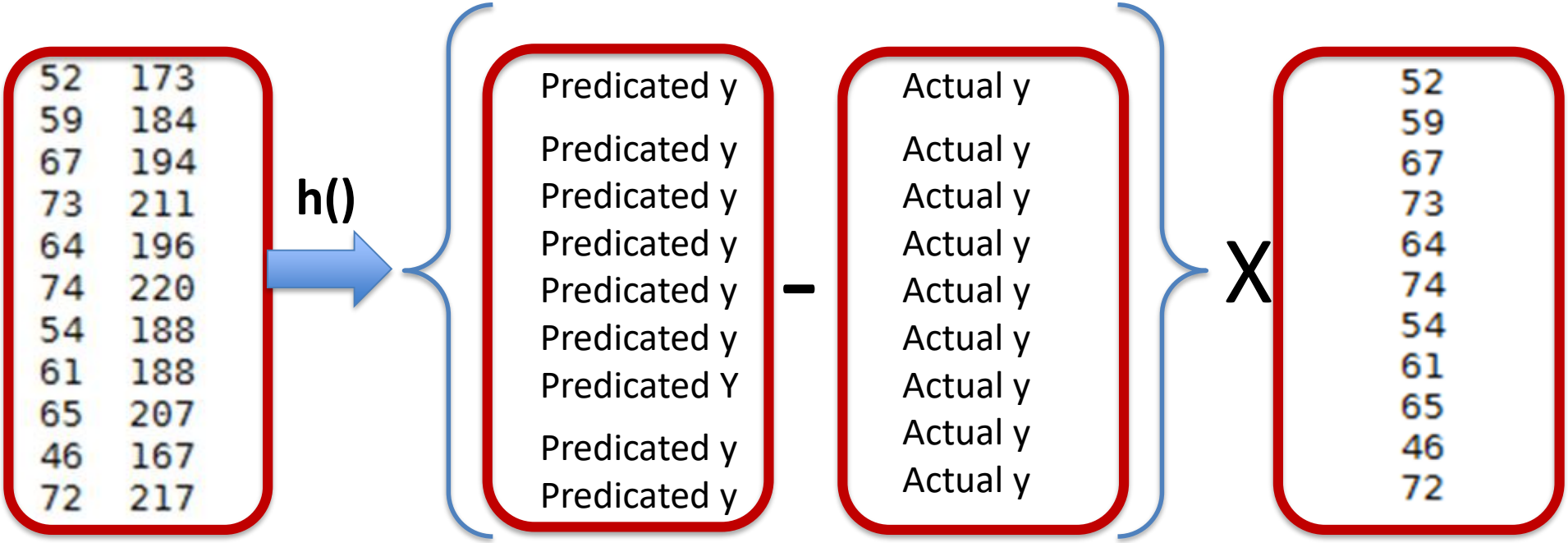
**Where $x_j^i$ is the value of feature j for the i$^{th}$ training example**

The above might look complex but as you have seen it is very similar to what we have already done with basic linear regression.

- When getting the derivate for $\lambda_j$ we input the training data into the our hypothesis function h.
- We then get the difference between the values returned and the actual target values y.
- We then multiply each of these differences by the values of the feature j (Remember $\lambda_j$ is the coefficient for feature j).

| | |
|---|---|
| 52 | 173 |
| 59 | 184 |
| 67 | 194 |
| 73 | 211 |
| 64 | 196 |
| 74 | 220 |
| 54 | 188 |
| 61 | 188 |
| 65 | 207 |
| 46 | 167 |
| 72 | 217 |

# Multiple (Multi-variate) Linear Regression




So if I want to get the derivative for $\lambda_1$ then I would pass all the training data X into the hypothesis function. Get the difference between the predicated values of the y and the actual values of y and multiply the result by the values of feature 1 of X (the training data). I will then add up all the resulting values and average them.

# Multiple (Multi-variate) Linear Regression

- So we can see that multiple linear regression is a relatively straight forward advancement of linear regression. We can now add as many features as we wish.

- Two issues we have not yet mentioned is the issue of selecting values for the **learning rate** and how to determine **convergence**.

- Unfortunately there is no proven method of selecting the best value for a learning rate. Typically you try a range of values and observe the error as gradient descent iterates.

- For example, you might start with [10, 5, 1, 0.5, 0.1, 0.05, 0.01, ….]

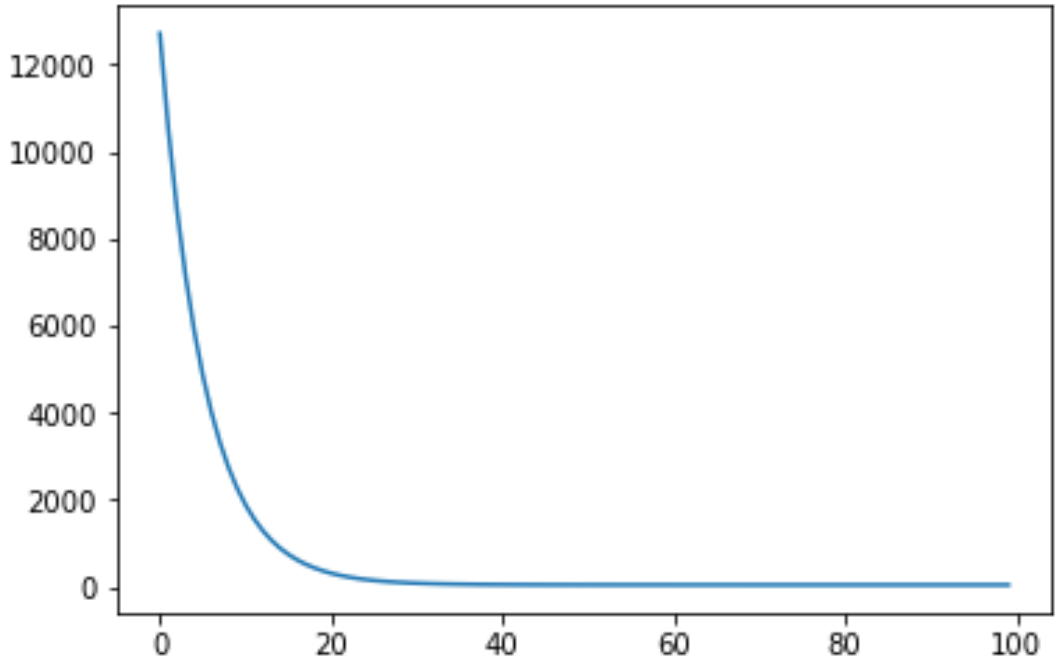
# Monitoring Convergence

It is useful to monitor the convergence of the gradient descent algorithm to ensure it is behaving as expected.

$$\frac{1}{m+2}\sum_{i=1}^{m}((h(\boldsymbol{x^i}) - y^i))^2$$

The plot on the right shows the squared error cost function (see eq on right) over 100 iterations of gradient descent.

As we will see later on it is important to not just monitor the loss for the training data but also very important to monitor the loss for unseen validation data as well.



# Standardizing Data

- Standardization facilitates the transformation of features to **a standard Gaussian(normal) distribution** with a **mean of 0** and a and a unit-variance.
- The scaling happens independently on each individual feature by computing the relevant statistics on the samples in the training set.
- Standardization of a dataset is a common requirement for dealing with dataset features with different ranges but also for many machine learning estimators such as clustering techniques , linear regression and logistic regression, neural networks, SVMs.

$$z = \frac{x - \mu}{\sigma}$$

```
print (X)

X = (X - np.mean(X))/np.std (X))
print (X)
```

| Input | Standardized |
|---|---|
| 0.0 | -1.33 |
| 1.0 | -0.80 |
| 2.0 | -0.26 |
| 3.0 | 0.26 |
| 4.0 | 0.80 |
| 5.0 | 1.33 |