CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

scikit learn

Machine Learning

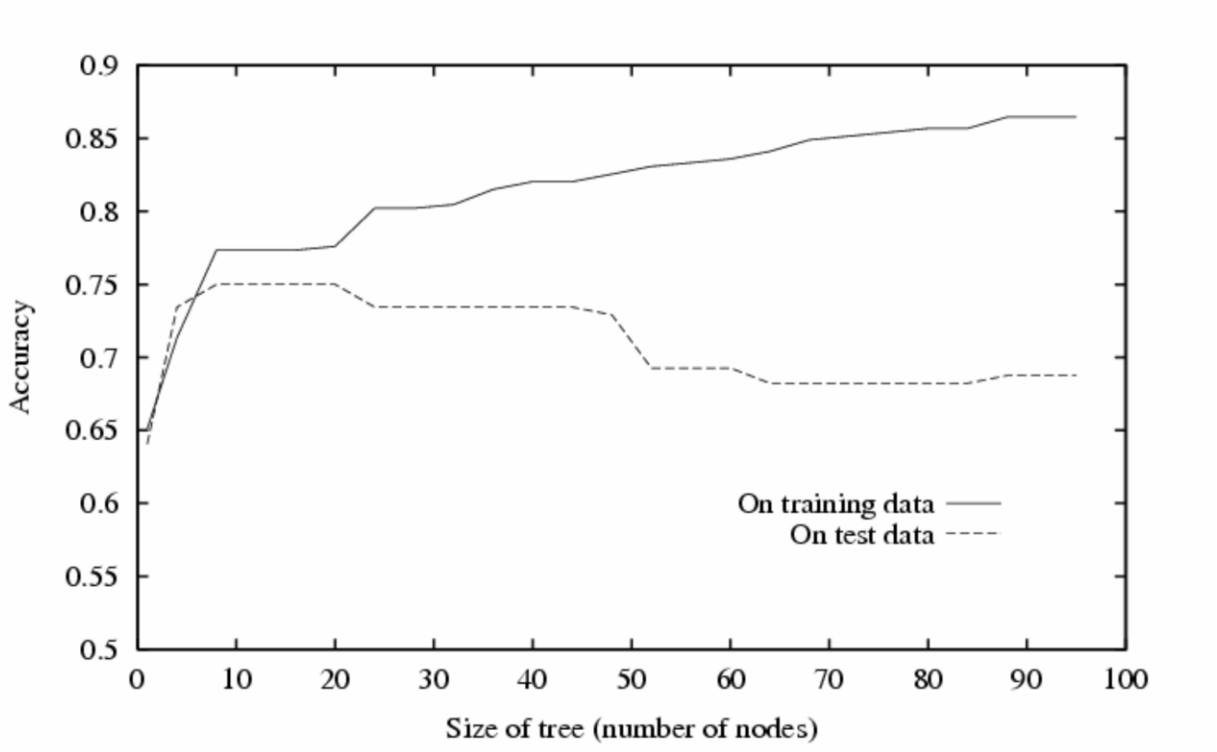
Machine Learning

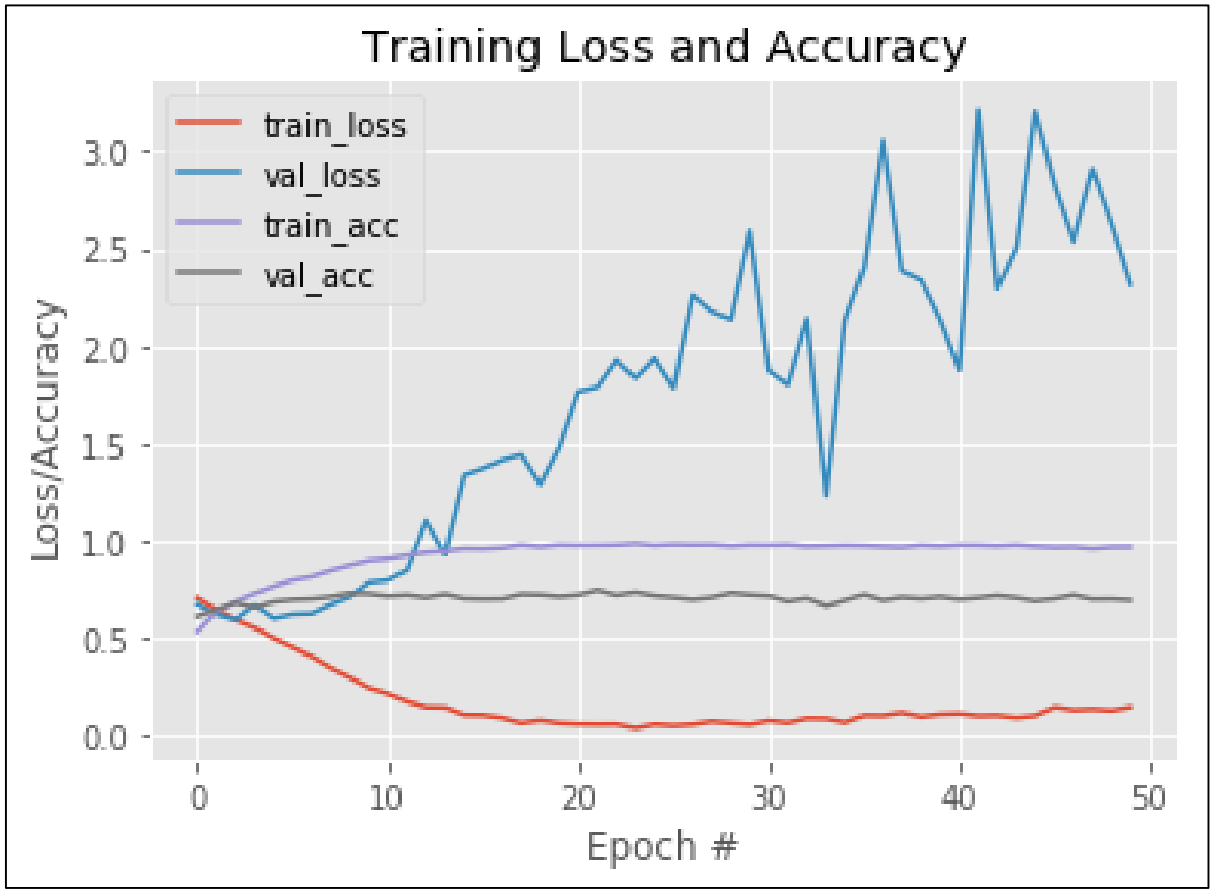
Lecture: Learning Curves

Ted Scully

Bias Vs Variance

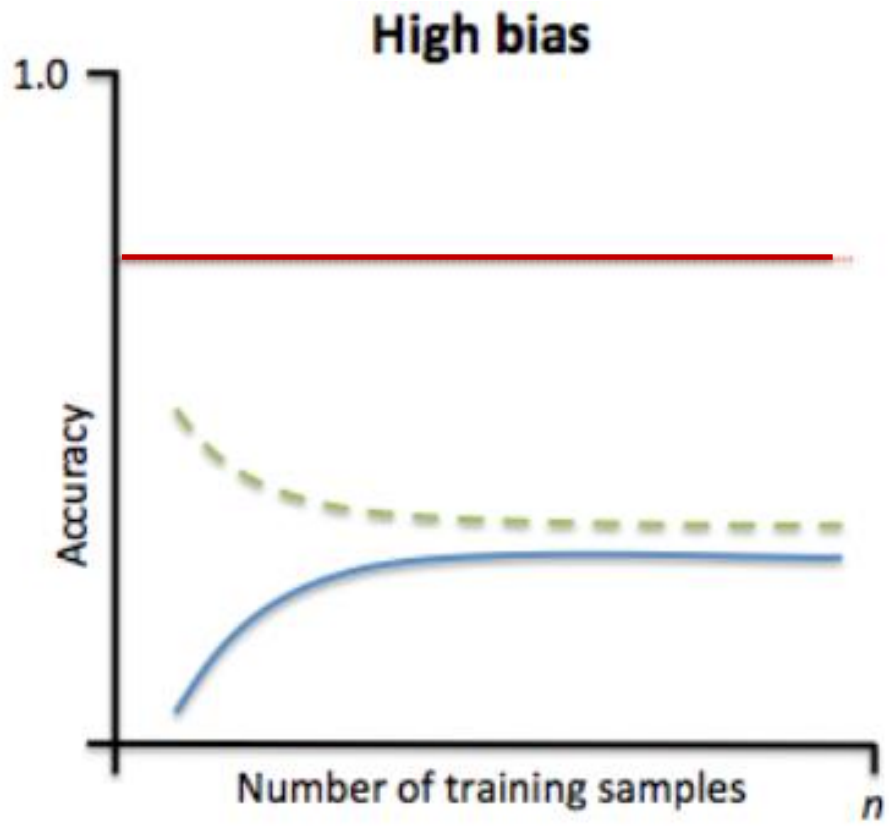
- ***Bias*** is the **inability** of an ML model to capture the **relationship between the features and the target**.
- ***Variance*** is the **sensitivity of a ML model** to changes in the dataset.
  - **High-variance** models may be able to represent their training set well, but may exhibit high variance in the results when tested on different samples of data.




Training Loss and Accuracy
train_loss
val_loss
train_acc
val_acc
Loss/Accuracy
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
10
20
30
40
50
Epoch #

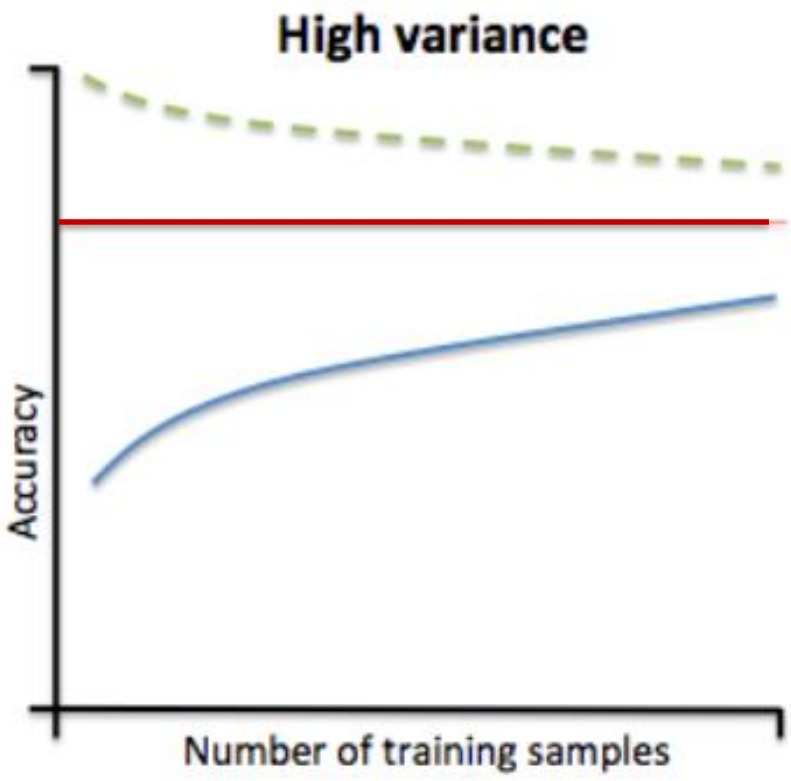
Learning Curves – High Bias

- With high bias the performance of the model on the training and cross validation set will be similar (**both will exhibit a poor accuracy or a high error**).
- Another interesting observation is that the addition of **additional data in a high bias scenario typically won't help improve the model performance**.



1.Use a more sophisticated model. Adding complexity to the model can help improve on bias. Deeper decision tree, larger neural network, etc.
2.Add more features. Adding additional features to the data could help improve a high-bias estimator. The existing features may not be strong enough predictors of the final class.

Learning Curves – High Variance



1. Try to reduce the complexity of your model. Increasing the regularization parameter, pruning a decision tree, increase value of k in kNN, reducing c value in SVM, etc. These are all examples that can reduce variance in a model.

2. Use fewer features. Using a feature selection technique may be useful, and decrease the over-fitting of the estimator.

3. Use more training samples. Adding training samples can reduce the effect of over-fitting, and lead to improvements in a high variance estimator.

Learning Curves in Scikit - Learn

- To build a learning curve in scikit learn we can use the **learning_curve** module in sklearn.learning_curve.

- **estimator** : ML algorithm that implements the “fit” and “predict” methods an object of that type which is cloned for each validation.

- **X** : training data (n_samples, n_features)

- **y**: class label vector length n_samples

- **cv** : Determines the cross-validation splitting strategy. Possible inputs for cv are:
  - None, to use the default 3-fold cross-validation,
  - integer, to specify the number of folds.
  - An object to be used as a cross-validation generator.

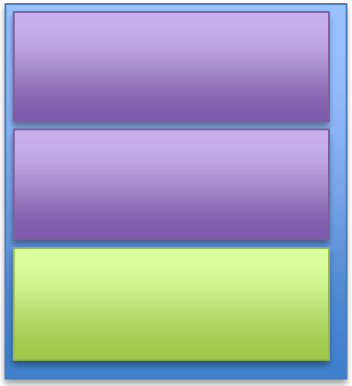
Learning Curves in Scikit - Learn

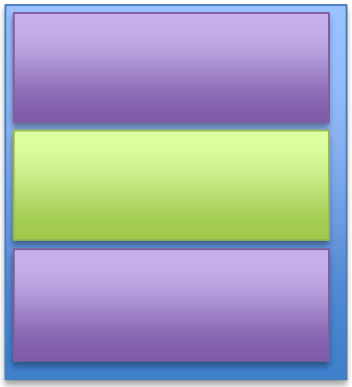
- Parameters of learning_curve :
- **train_sizes** : specifies how we should divide the training set for each point in the learning curve.
  - We can specify this using **np.linspace -** Returns evenly spaced numbers over a **specified interval**.
  - If we have 1000 training examples and wish our learning curve to have 10 values between 100 and 1000 then we can specify
  - **np.linspace(0.1, 1, 10)** -> **[0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.]**

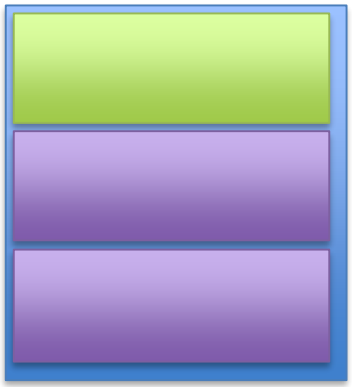
Learning Curves in Scikit - Learn

- The learning_curve method then returns the following:
  - **train_sizes_abs**: Numbers of training examples that has been used to generate the learning curve.
  - **train_scores** : array, shape (n_ticks, n_cv_folds) – Scores on training set
  - **test_scores** : array, shape (n_ticks, n_cv_folds) – CV score on test set

- It is import to understand that both train_scores and test_scores are both 2D NumPy arrays.







Learning Curves in Scikit - Learn

- The learning_curve method then returns the following:
  - **test_scores** : array, shape (n_ticks, n_cv_folds) – CV score on test set

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB

from sklearn.datasets import load_iris
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit

iris = load_iris()
X, y = iris.data, iris.target

# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
custom_cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=0)


estimator = GaussianNB()

title = "Learning Curve Iris (NB)"
plt.title(title)
ylim=(0.0, 1.01)
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB

from sklearn.datasets import load_iris
from sklearn.model_selection import learning
from sklearn.model_selection import ShuffleS

iris = load_iris()
X, y = iris.data, iris.target

# Cross validation with 100 iterations to get sm
# score curves, each time with 20% data randc
custom_cv = ShuffleSplit(n_splits=100, test_s


estimator = GaussianNB()

title = "Learning Curve Iris (NB)"
plt.title(title)
ylim=(0.0, 1.01)
```

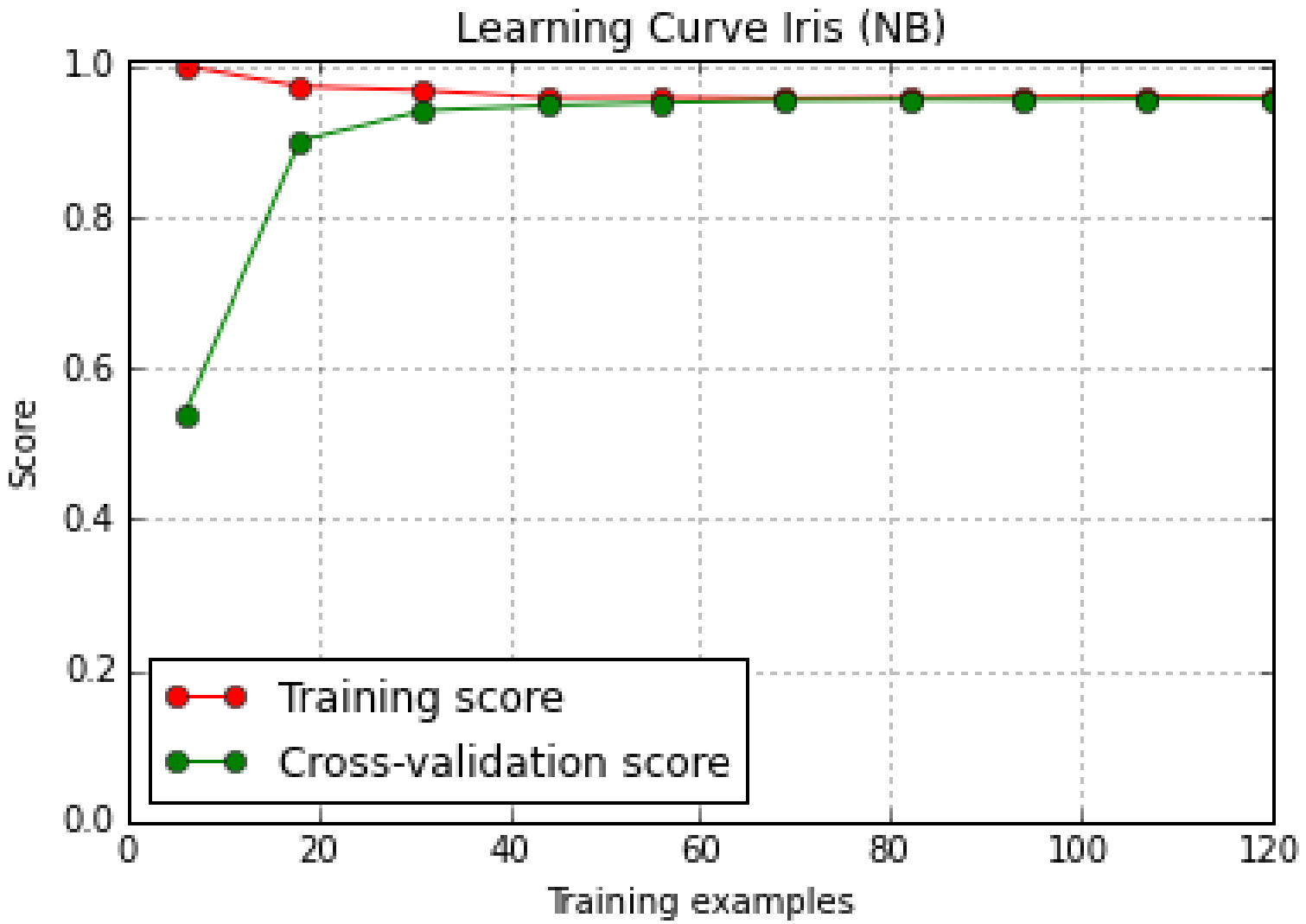
Notice we use a large number of folds (100) to make sure that the results are smooth. If we were to use normal cross fold validation with 100 folds, we may risk not having enough data to split into 100 different folds.

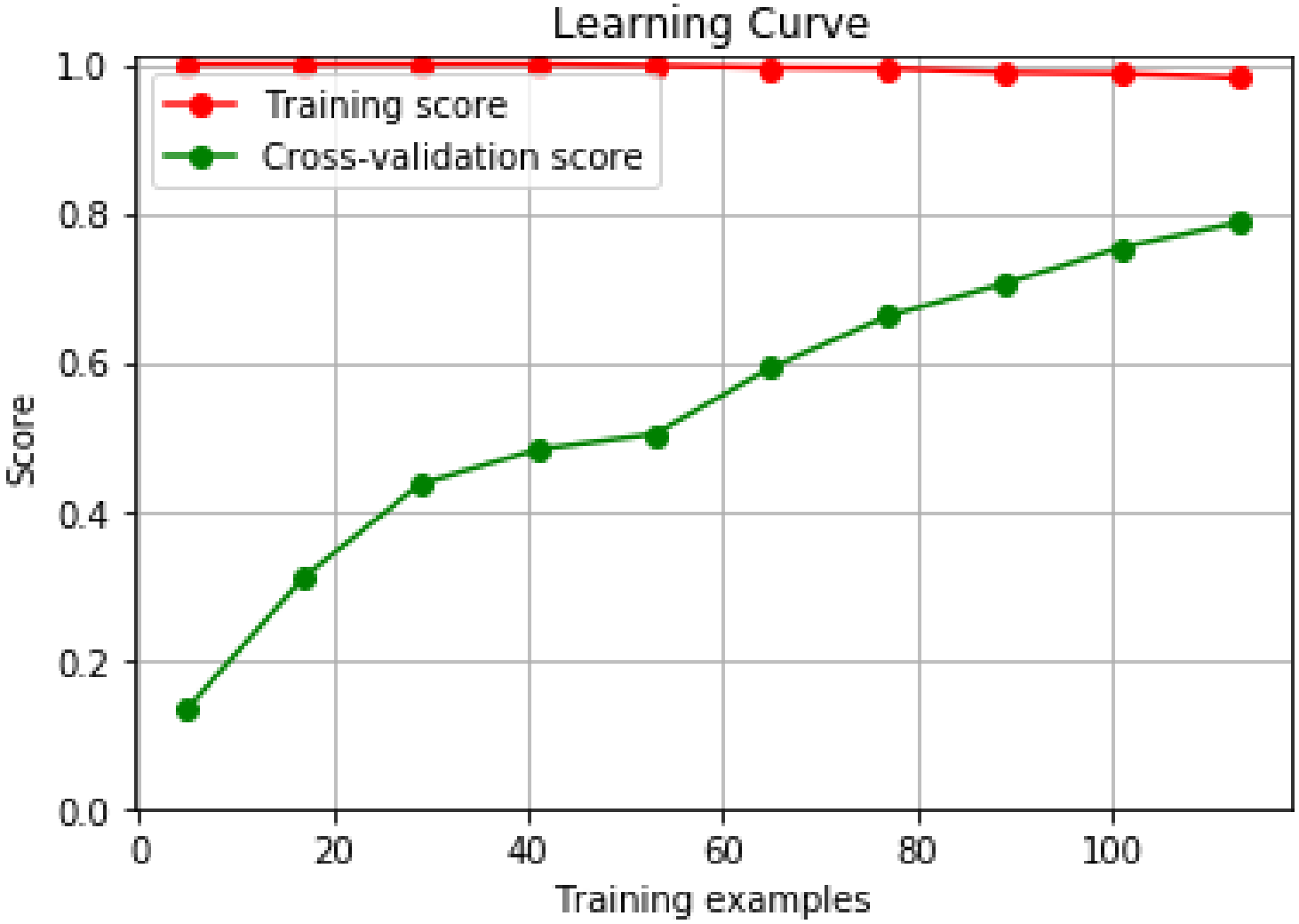
ShuffleSplit takes a slightly different approach. For each fold it randomly select 20% of the data for test and the rest for training. So it doesn't suffer from the same problem.

```
plt.ylim(ylim)
plt.xlabel("Training examples")
plt.ylabel("Score")

train_sizes=np.linspace(0.1, 1.0, 10)

train_sizes, train_scores, test_scores = learning_curve(estimator, X, y,
train_sizes=train_sizes, cv = custom_cv)

train_scores_mean = np.mean(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
plt.grid()


plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training score")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation score")

plt.legend(loc="best")
plt.show()
```

```
plt.ylim(ylim)
plt.xlabel("Training examples
plt.ylabel("Score")

train_sizes=np.linspace(0.1,

train_sizes, train_scores, test_scores = learning_curve(estimator, X, y,
train_sizes=train_sizes, cv = custom_cv)

train_scores_mean = np.mean(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
plt.grid()


plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training score")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation score")

plt.legend(loc="best")
plt.show()
```

Remember axis = 1 operates on the horizontal axis and calculates the mean for each row.
Each row corresponds to one tick (first row might correspond to 10% of data, the next row 20% of data, etc) and each value in a row is the accuracy achieved for each iteration of the cross fold (so in this example, each row would have 100 values).


Learning Curve Iris (NB)
Score
Training examples
Training score
Cross-validation score
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120


Learning Curve
Training score
Cross-validation score
Score
Training examples
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100

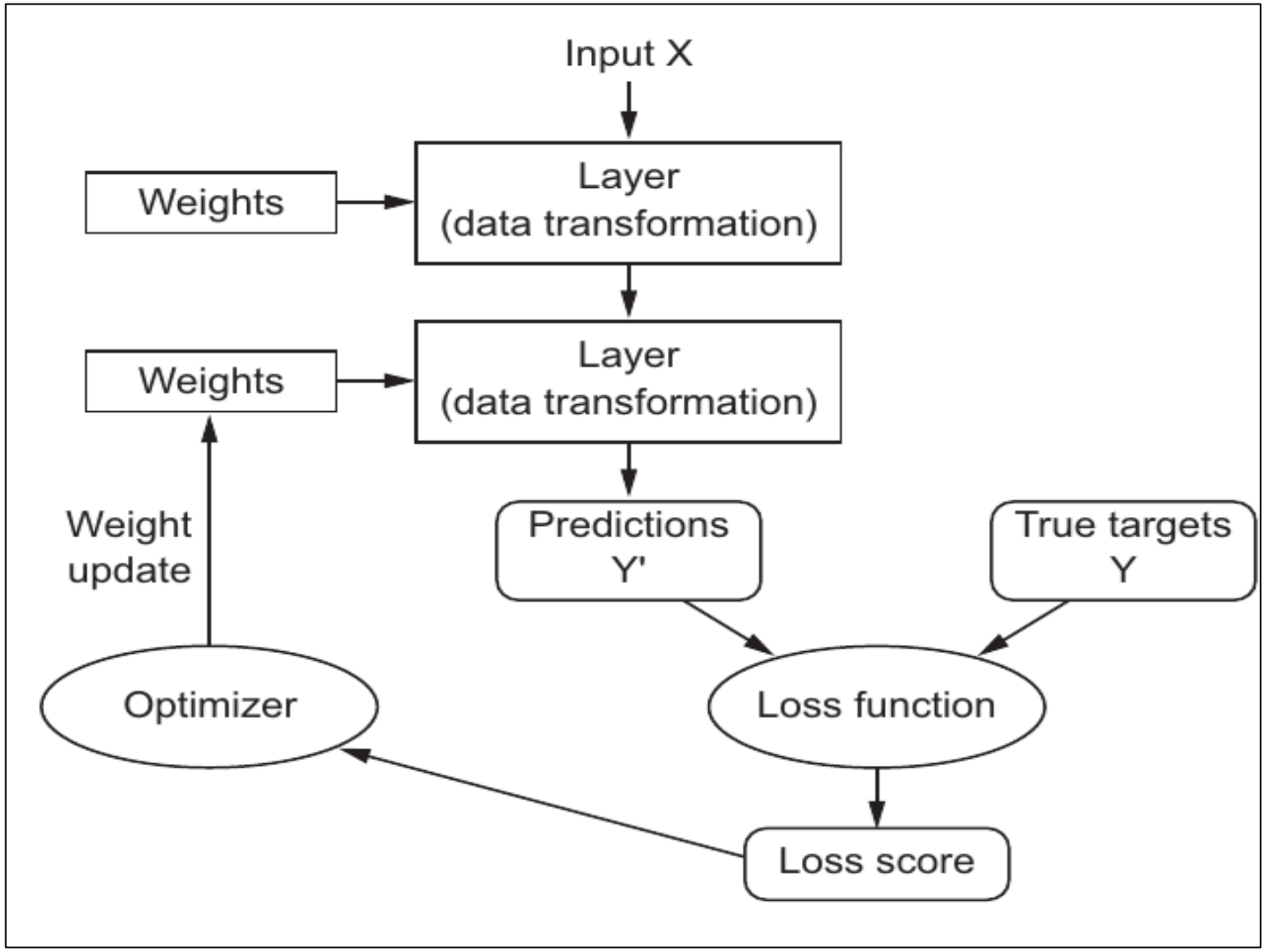
CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Machine Learning

Machine Learning

Lecture: Linear Regression

Ted Scully

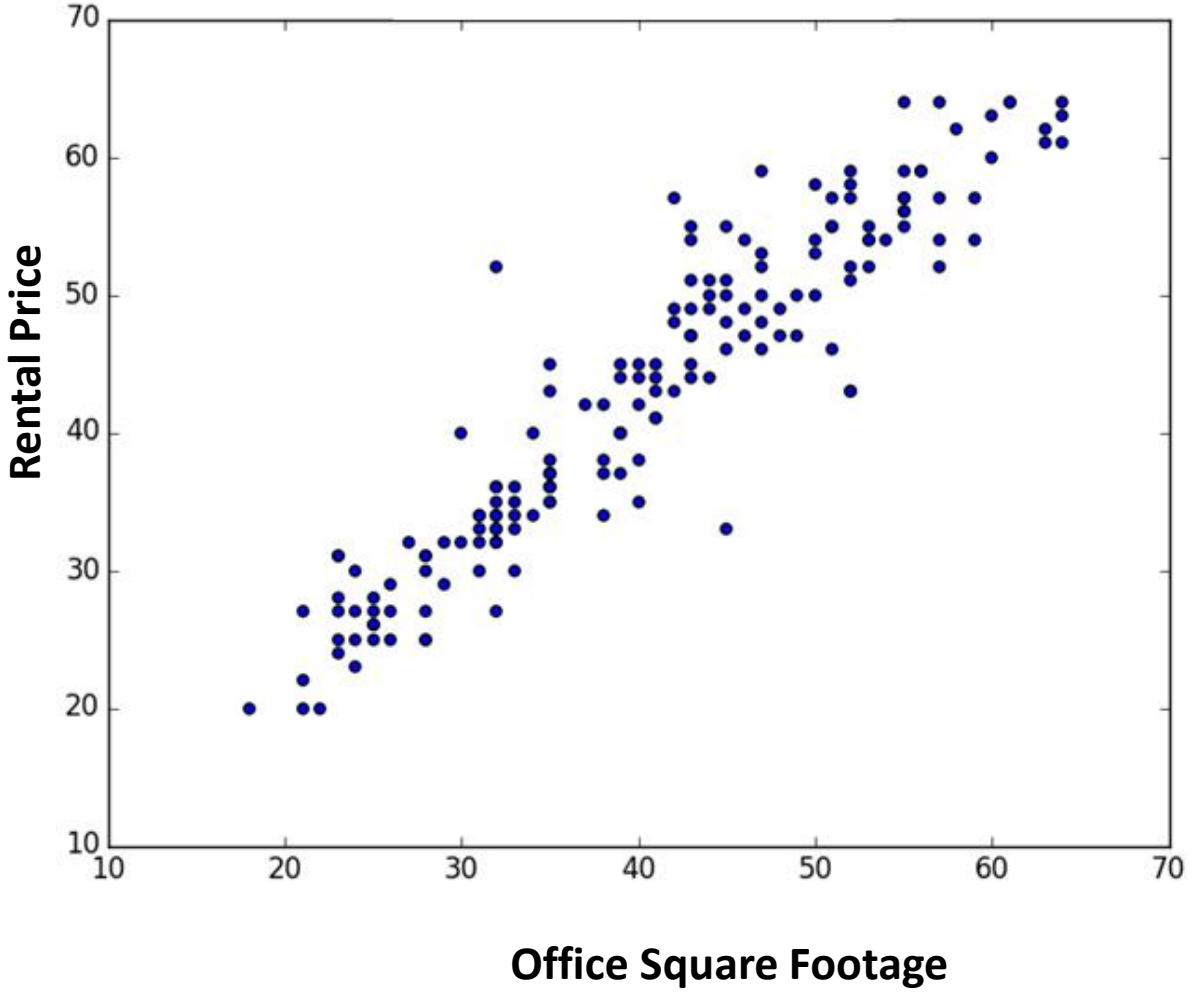

Input X
Weights
Layer
(data transformation)
Weights
Layer
(data transformation)
Weight
update
Predictions
Y'
True targets
Y
Optimizer
Loss function
Loss score

Introduction to Linear Regression

Linear regression attempts to **model the relationship between two variables** by fitting a linear equation to observed data.

The data shows us the relationship between the sq footage of an office and the rental price.

It could be useful to create a **function** that accepts the square footage of an office and allows us to estimate (based on this data) the associated rental price.

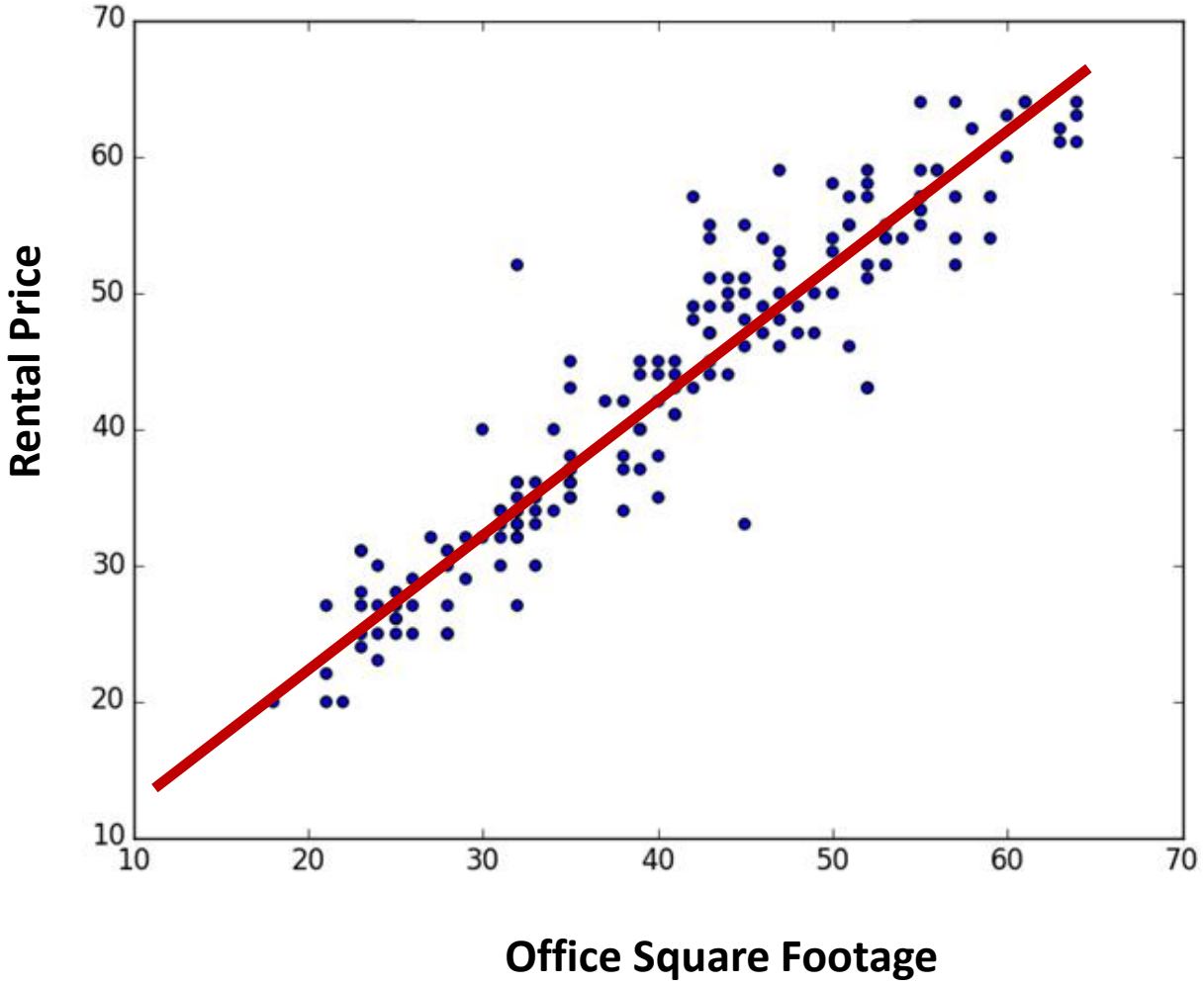


Introduction to Linear Regression

Linear regression attempts to **model the relationship between two variables** by fitting a linear equation to observed data.

The data shows us the relationship between the sq footage of an office and the rental price.

It could be useful to create a **function** that accepts the square footage of an office and allows us to estimate (based on this data) the associated rental price.



Notation

- **m** = Number of training examples
- **x** = Input variable (for example sq footage of an office)
- **y** = Output variable (for example the associated office rental price)

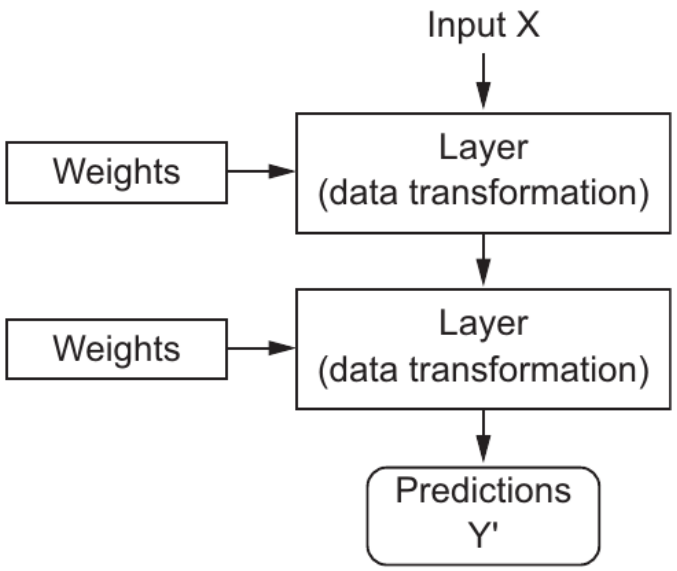
- **(x^i, y^i)** is the **i^{th}** training example from the dataset

- A linear regression line has an equation of the form:

  - ***h(x) = λ_1 x + b***

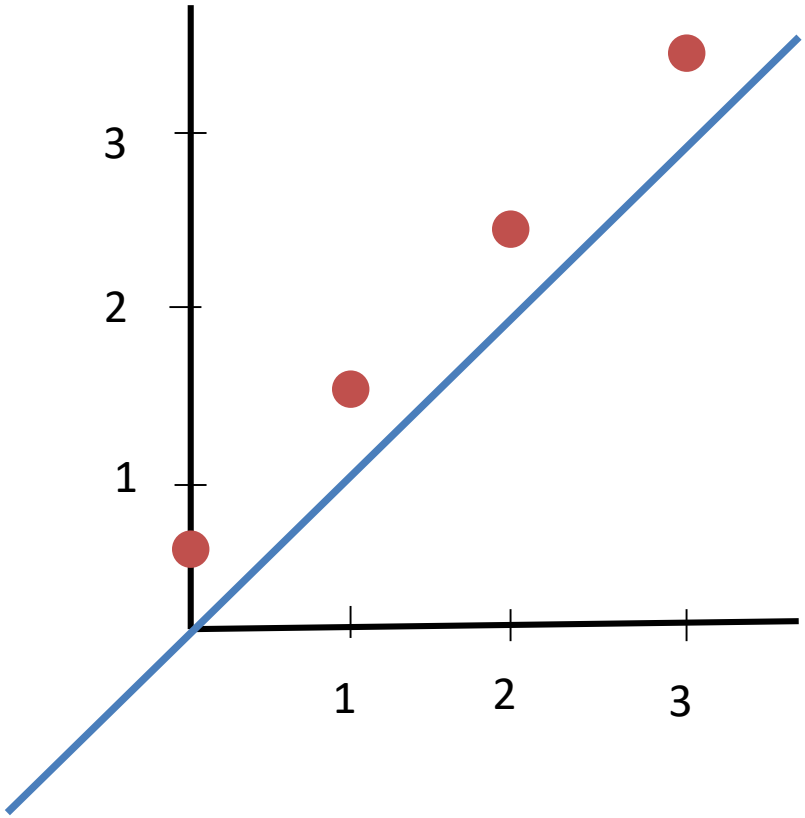
  - The **slope** of the line is $\boldsymbol{\lambda_1}$, and $\boldsymbol{b}$ is the **y** intercept (often referred to as bias).

- $h(x) = \lambda_1 x + b$



Formalising LR Problem

- The objective of a linear regression problem can be loosely described as assigning values to the variables λ_1 and ***b*** so as to **minimise predictive error**.
- So the first question we need to answer is **what do we mean by predictive error**? How would you measure it?



$$h(x) = \lambda_1 x + b$$

Let's assume we have the following values for our parameters:

b = 0

λ_1 **= 1**

Squared Error Cost/Loss Function

Here we obtain the difference between the actual value of $\mathbf{y^i}$ and our predicted value.
In this example $\mathbf{x^i}$ would be the size of a particular house from the dataset.
Our linear regression model produces the function **h**, which returns our estimated selling price for this house $\mathbf{h(x^i)}$.
We then subtract this from the actual selling price $\mathbf{y^i}$.

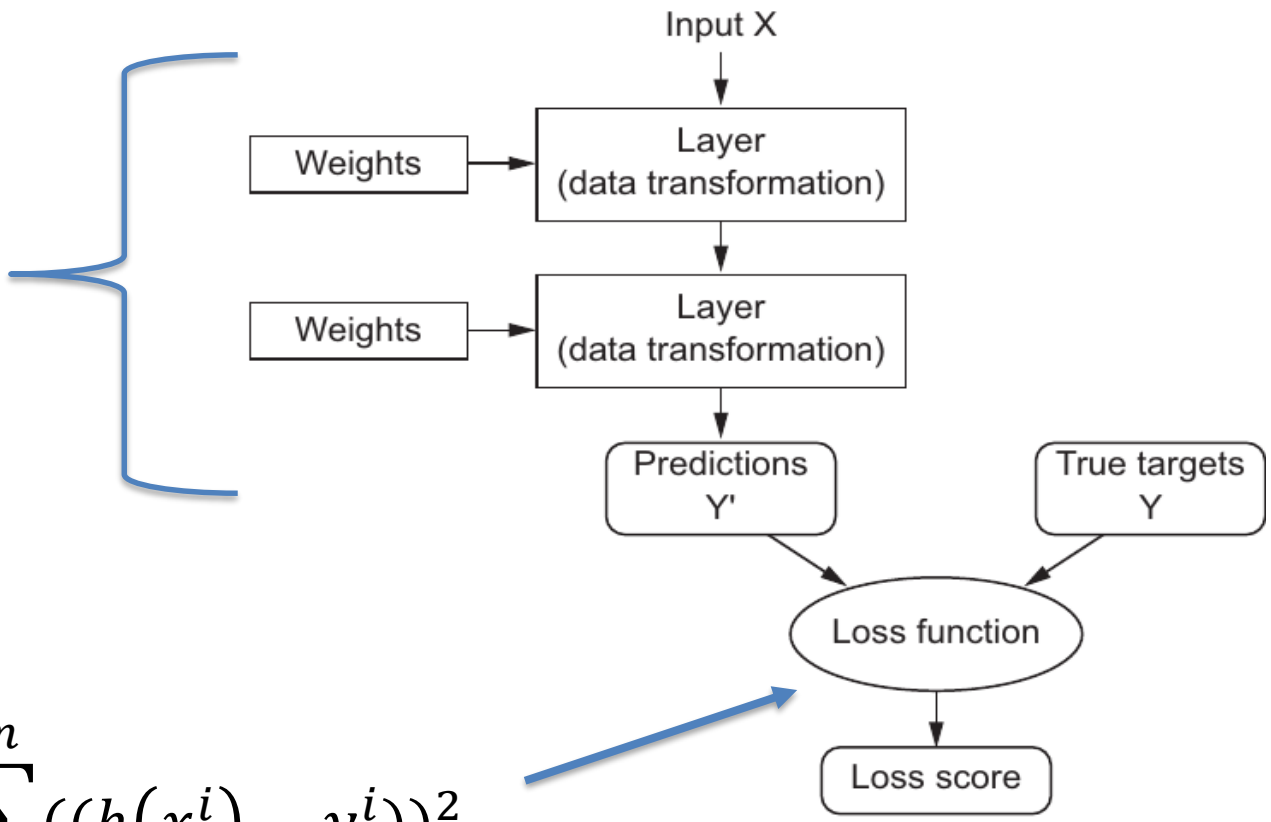
$$\frac{1}{2m}\sum_{i=0}^{m}((h(x^i)-y^i))^2$$

Squared Error Cost/Loss Function

We then sum the squared difference (squared error) for each value in our training set. We then average the value by dividing by m (the number of training set examples).

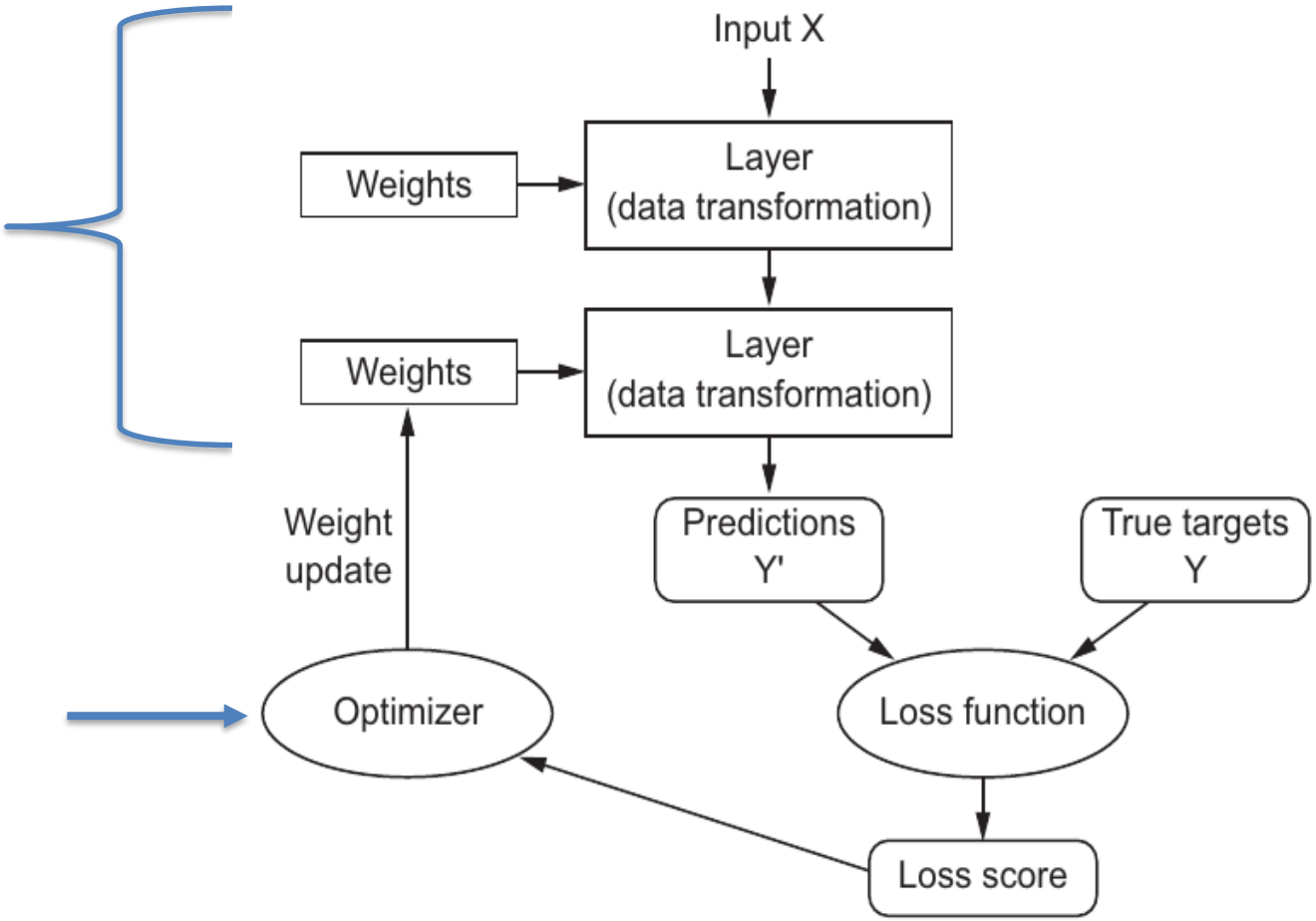
$$\frac{1}{2m}\sum_{i=0}^{m}((h(x^i)-y^i))^2$$

$h(x) = \lambda_1 x + b$



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

$h(x) = \lambda_1 x + b$



Gradient Descent

Optimizer

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

Squared Error Cost/Loss Function

Remember our function is $h(x) = \lambda_1 x + b$.
Therefore, our overall objective is to identify the values of b and λ_1 to plug into h such that we obtain the **minimal** average square error across all training examples. This is the cost function (lost function) for linear regression. For the moment we use these terms (cost function and loss function) interchangeably.

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

Squared Error Cost/Loss Function

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

Just as a shorthand I'm going to rewire the above formula as follows:

$$minimise_{\lambda_1, b}\ C(\lambda_1, b)$$

Gradient Descent and Linear Regression

- To understand and visualize the application of gradient descent we are going to **simplify the problem even further**.
- We are going to assume that the y-intercept (*b*) is always 0. Therefore, we only have one variable to worry about now (λ_1)

- ***h(x) = λ_1 x***

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

$$minimise_{\lambda_1}\ C(\lambda_1)$$

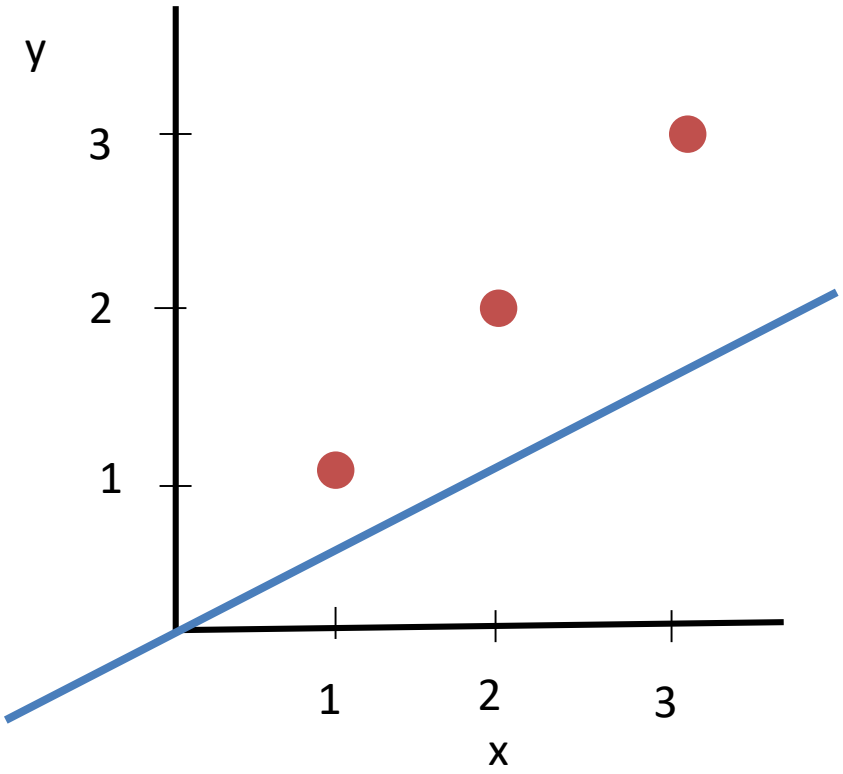
Over the next few slides we will pick a few different value for λ_1 and we will monitor the corresponding value of the cost function. The following is our simple dataset:

X	Y
1	1
2	2
3	3

X	Y
1	1
2	2
3	3

$h(x) = \lambda_1 x$

Function h(x)



Let's examine what happens when I give $\boldsymbol{\lambda_1}$ **a value of 0.5**.

We subsequent calculate the associated cost:

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

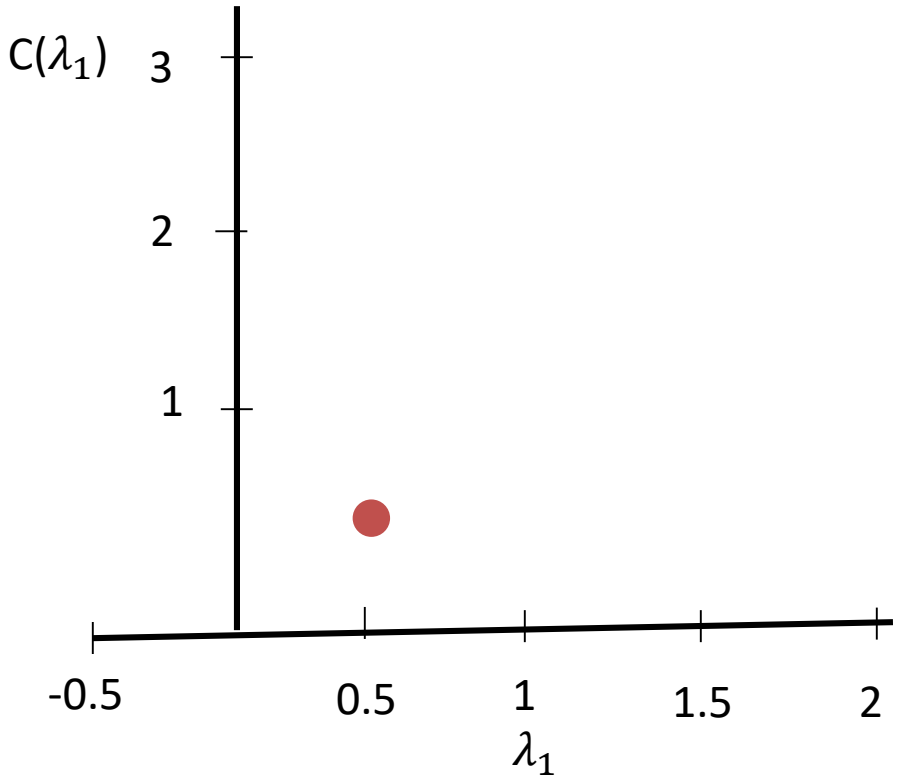
$$\frac{1}{6}(\,(0.5-1)^2 + (1-2)^2 + (1.5-3)^{\,2})$$

$$\frac{1}{6}(\,0.25 + 1 + 2.25)$$

$$= 0.58$$

Notice we define our line only using a single parameter λ_1 . Controls the slope and must pass through the origin.

Function $\mathbf{C(\lambda_1)}$

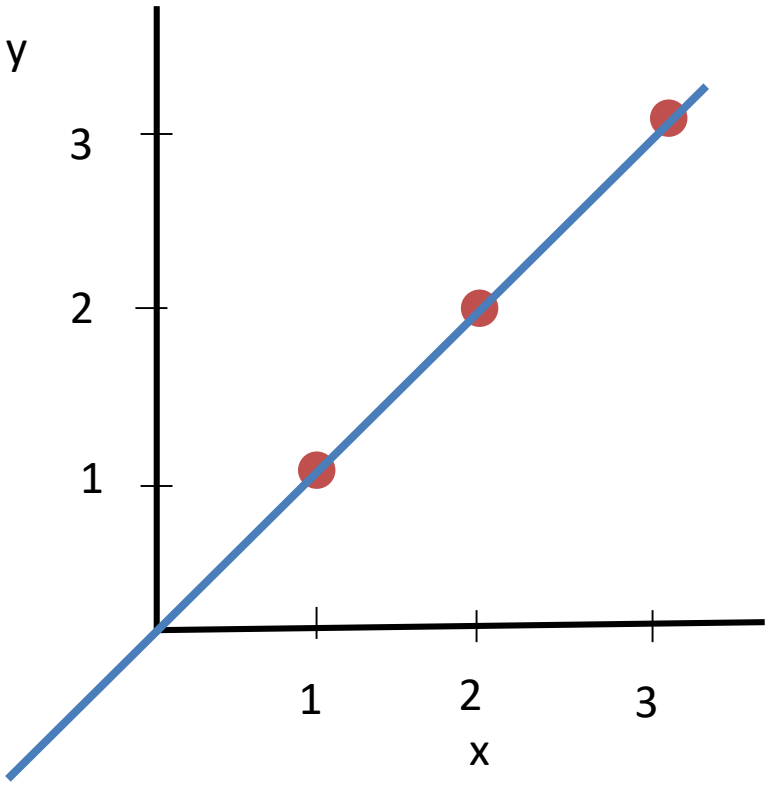


We are going to pick multiple values for λ_1 and map the relationship between C(λ_1) and λ_1 . When $\lambda_1 = 0.5$ then C(λ_1) = 0.58

X	Y
1	1
2	2
3	3

$h(x) = \lambda_1 x$

Function h(x)



Now we examine what happens when I give $\boldsymbol{\lambda_1}$ **a value of 1**.

This provides an excellent fit to the data.

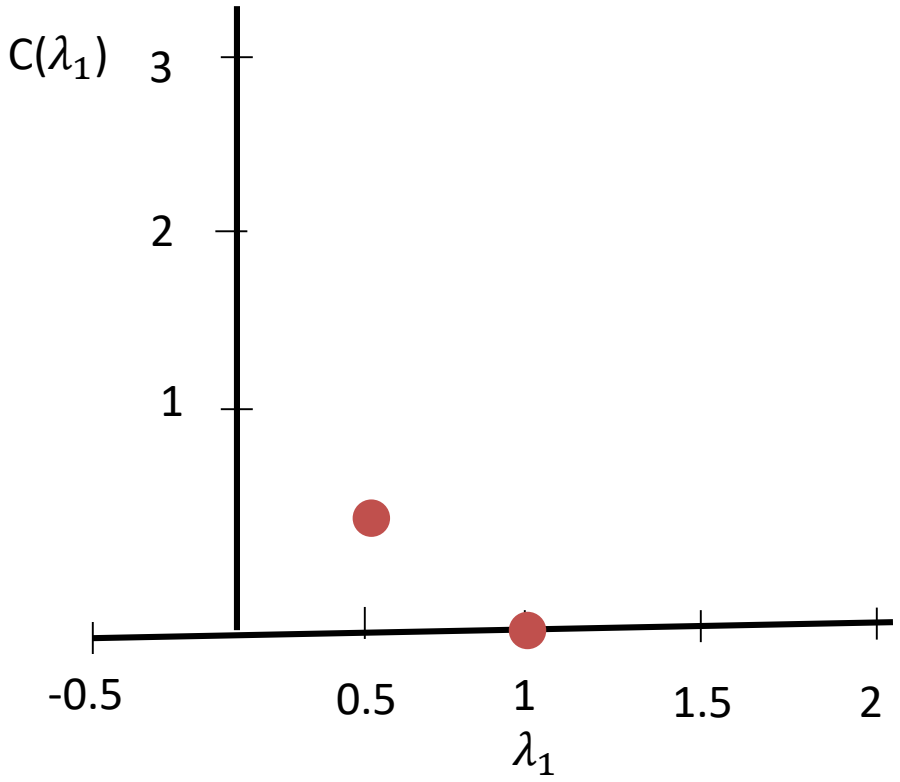
$$c(\lambda_1) = \frac{1}{2m}\sum_{i=0}^{m}((h(x^i) - y^i))^2$$

$$\frac{1}{6}(\,(1-1)^2 + (2-2)^2 + (3-3)^{\,2})$$

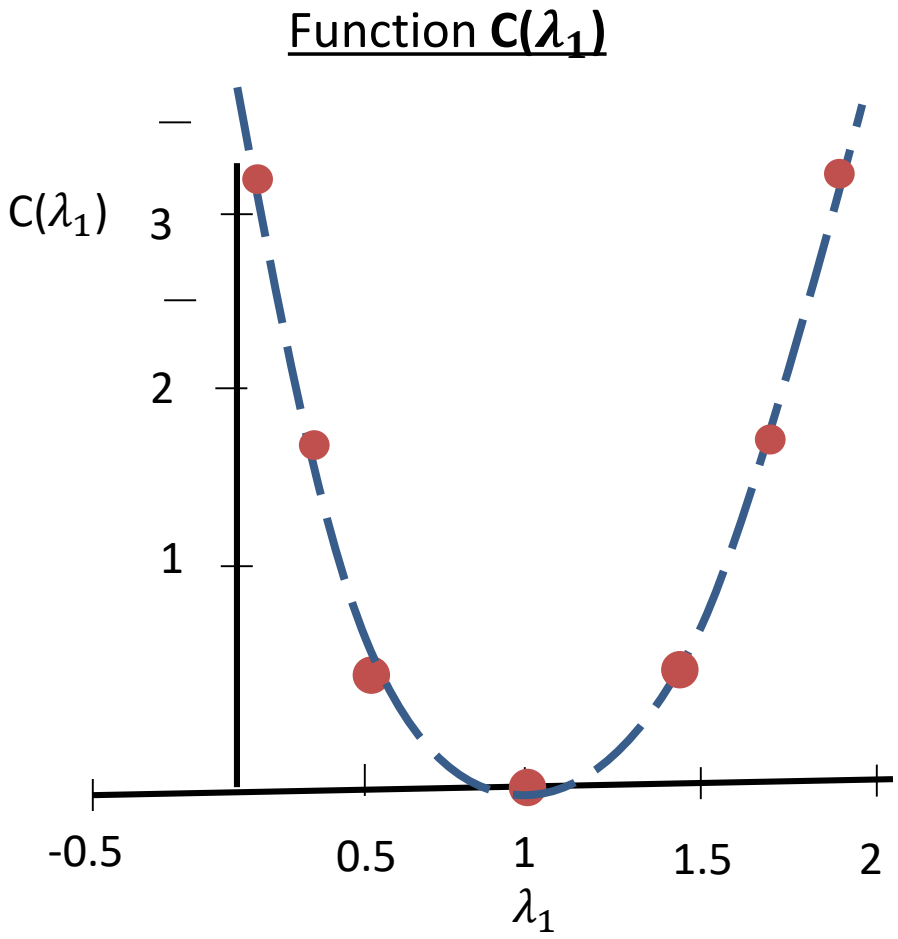
$$\frac{1}{6}(\,0)$$

$$= 0$$

Function $\mathbf{C(\lambda_1)}$



We are going to pick multiple values for λ_1 and map the relationship between C(λ_1) and λ_1 . When $\lambda_1 = 1$ then C(λ_1) = 0

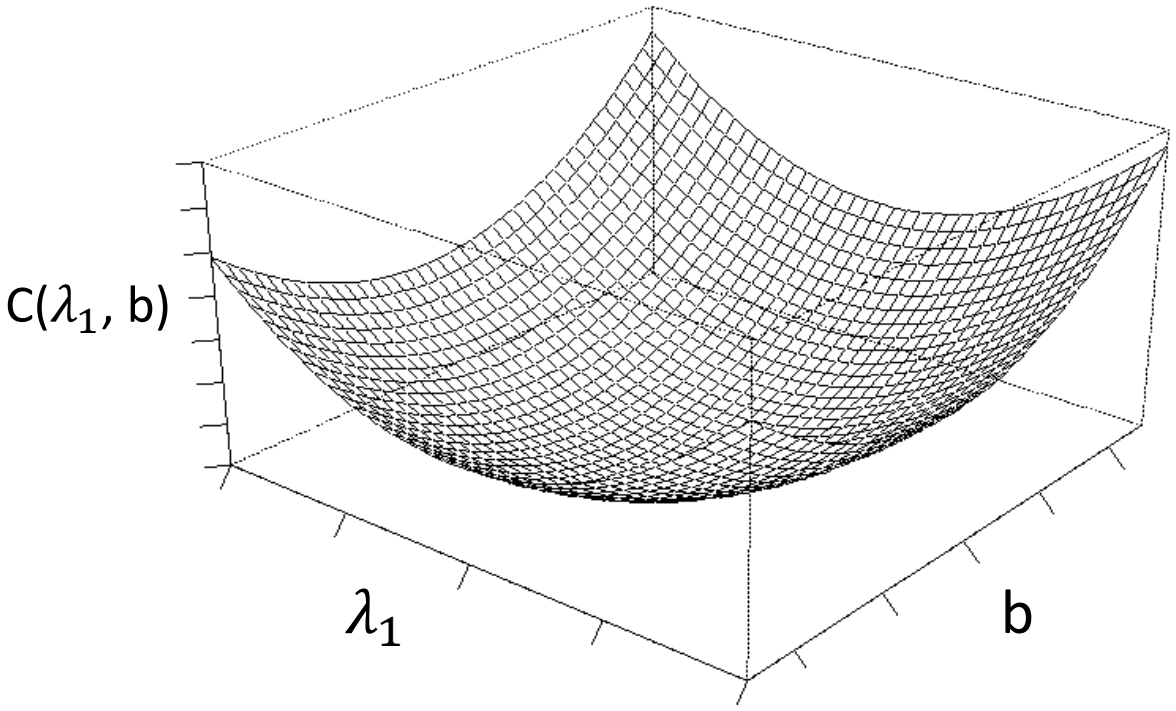


If we were to continue this process and map the shape of C(λ_1) then it would exhibit a regular **convex** shape as shown above.

$h(x) = \lambda_1 x + b$

Linear Regression – A Search Problem

- Now let's add the our additional parameter *b* back into our linear equation
- Below is an example **depiction of** $C(\lambda_1)$ when we have two parameters (λ_1 and *b*)



Gradient Decent – An Optimization Algorithm

- We have a function **C(λ_1, b)** and our objective is to determine the values of λ_1 and b that will give the **minimum** value of C(λ_1, b)
- Gradient Decent (Overview)
  - Start with a **random** value of λ_1 and *b*
  - Alter the value of λ_1 and *b* in order to continually reduce the value of C(λ_1, b)
  - Continue until we reach a minimum