Logistic Regression via gradient descent by Rohitash Chandra

Logistic regression

Linear regression is not a widely used model as real-world problems do not typically have a linear relationship when you look at the covariates (inputs or x) and the outcomes (y or outputs). Hence we need to look for better underlying models to capture the nonlinear relationships in the data.

We also need to consider the case of classification problems or pattern recognition where the goal is to identify patterns according to their class. In this slide, we will discuss logistic regression to model nonlinear data and to recognise patterns.

A logistic regression model is similar to a linear regression model, with the major difference that the activation is a sigmoid or logistic function rather than a linear function.

Below is an example of a model where the sum of the incoming or attached units $(w_1, w_2, w_3..w_N)$ over the inputs $(x_1, x_2, ...x_N)$ is computed and then the output (Z) goes through an activation function. A logistic activation function makes it a logistic regression model which is also known as the perceptron in the neural network literature.

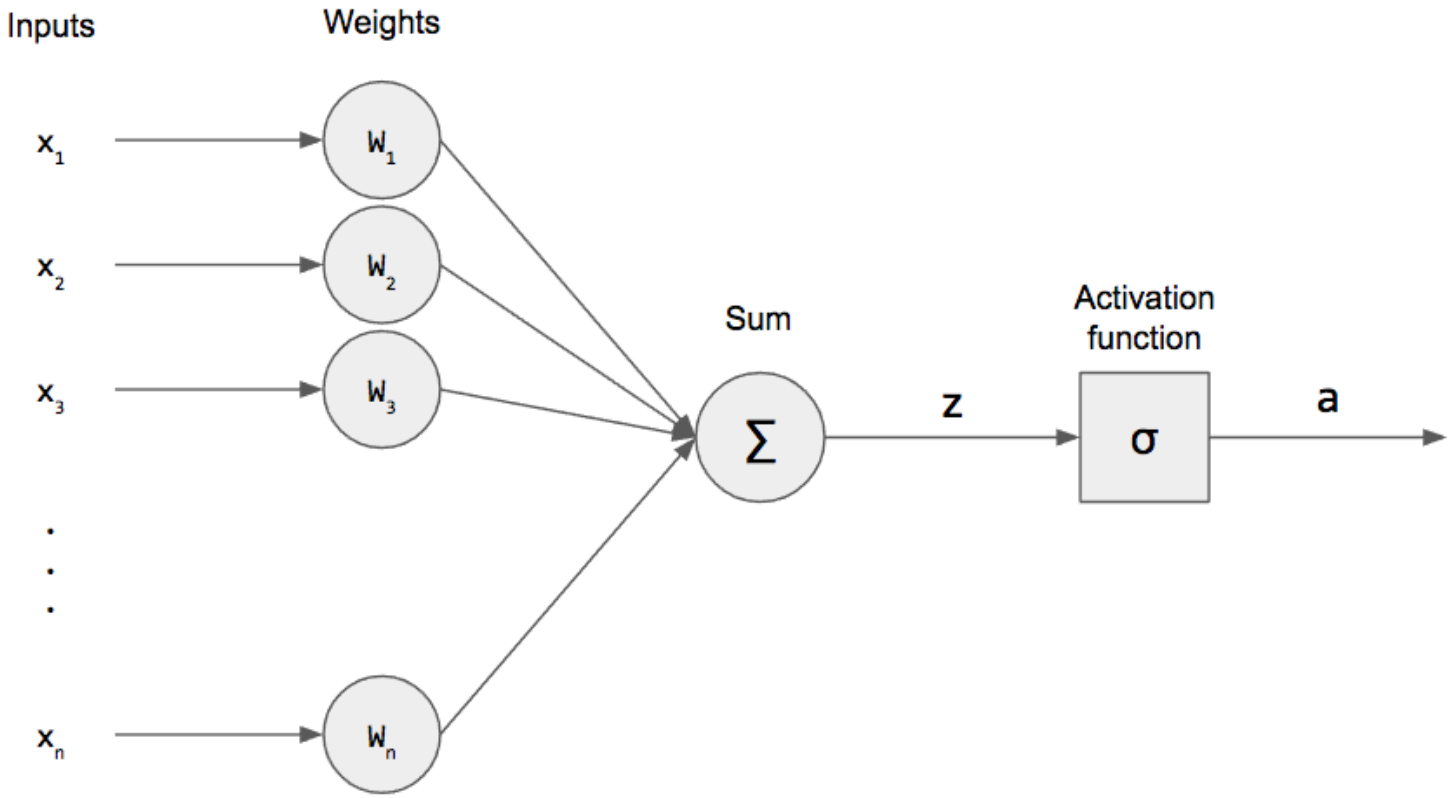


Figure: Linear regression model. Adapted from Nginx, n.d. Retrieved from https://static.edusercontent.com/files/AnSeb7ymghifF0PZjlCL4c7F.

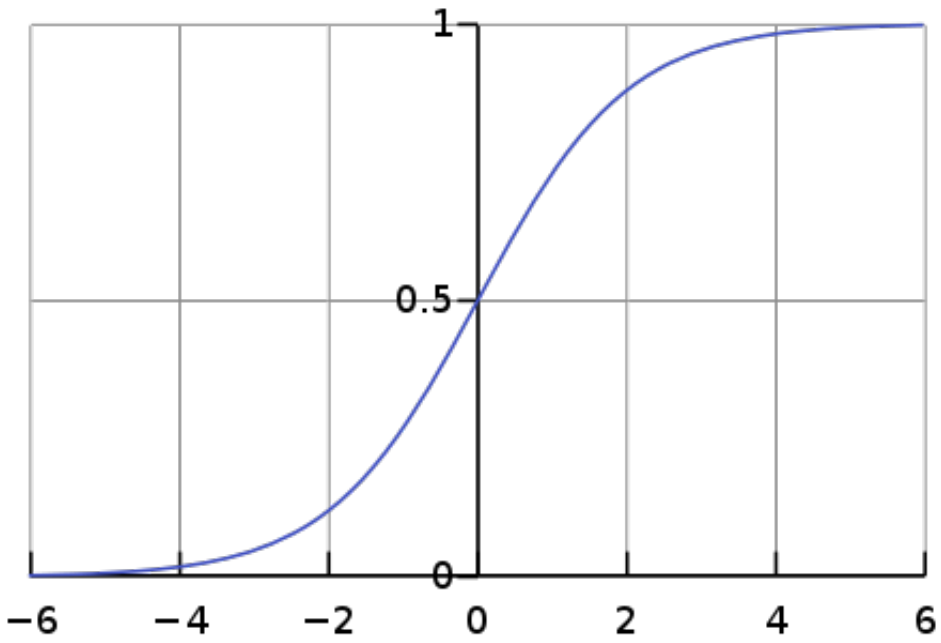


Figure: Sigmoid or logistic function. Adapted from "Sigmoid function" by Wikipedia, 2020. Retrieved from https://en.wikipedia.org/wiki/Sigmoid_function.

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$

Run PYTHON

```
1
2 import numpy as np
3 import math
4
5 x = np.linspace(-4, 4, 5)
6 z = 1/(1 + np.exp(-x))
7
8 print(x, z, ' x, z')
```

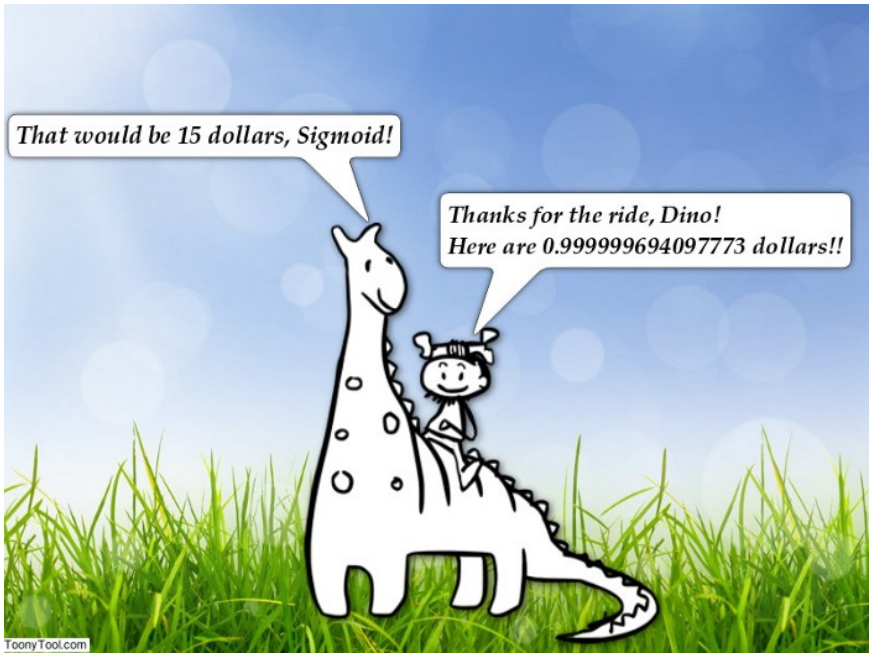
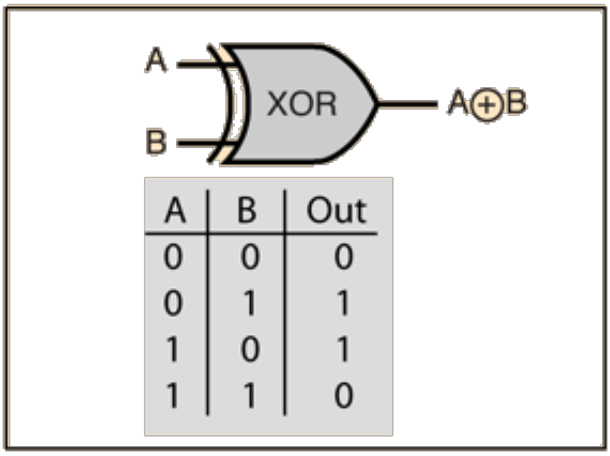


Figure: Sigmoid meme. Adapted from "Derivative of the sigmoid function" by Arunava, 2018. Retrieved from https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e.

As noted, a logistic regression model is also known as the perceptron, which is the building block of neural networks. The perceptron can either have a linear activation (that makes it equivalent to linear regression) or a logistic action (that makes it equivalent to logistic regression). In the case of neural networks (covered in Week 3), the logistic activation function is called the sigmoid activation function.

Below are videos that show how logistic regression or perceptron is trained with a simple example. As previously noted, the parameters are also named differently but essentially have the same meaning (weights or parameters or coefficients). The update of the parameters iteratively using gradients is known as training or learning, and in some cases, as optimisation. In statistics literature, the input data is called covariates, while in machine learning literature it is called features or training instances, and output is called class labels or outcomes.

Watch the below video that shows an example of how the parameters of the perceptron model (weights and bias) are training using gradient descent and an exclusive OR (XOR) gate problem. The XOR gate problem is a classic learning problem to demonstrate classification problems (with only 4 instances or samples in the data set and one outcome or class label). A and B are the features or covariates and Out is the class label. The perceptron can also be used for AND or OR gate, which are easier problems when compared to the XOR gate. Easier means that the problem can be trained faster with fewer iterations in weight update before the model gets to a point that it can correctly classify all the instances in the training data. Note there is no separate test data in these simple examples. The training data is also the test data set. More on training and testing data sets will be covered later.



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

Figure: Two input XOR gate. Adapted from "HyperPhysics" by R.Nave, n.d. Retrieved from http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/xor.html.

Watch the following videos on perceptron and logistic regression.

In the below video, note that the bias is a special type of weight (trainable parameter) with a fixed input (-1 or 1), whereas the weights have inputs $(x_1, x_2, ..x_N)$. The learning rate is a fixed variable chosen by the user, that determines the intensity of the weight update. Typically, the value of 0.1 makes a suitable learning rate but this can be evaluated for different problems, depending on what effect (in terms of classification performance) it has on the training and test data set.

Perceptron training

Udacity. (2015, February 23). *Perceptron Training* [online video]. Retrieved from https://www.youtube.com/watch?v=5g0TPrxKK6o.

The video below presents a real-world example using a logistic regression model (Perceptron Algorithm) for an email spam classifier problem. The class output is either spam or not spam, which is known as a binary classification where only two classes are present. Log-Loss Error is another way to capture how your model is training. You can use Sum-Squared-Error to do the same. More details about different error functions will be given later this week in a slide titled Measurement of error.

Logistic regression and the Perceptron Algorithm: A friendly introduction

Serrano, L. (2019, January 01). *Logistic Regression and the Perceptron Algorithm: A friendly introduction* [online video] Retrieved from https://www.youtube.com/watch?v=jbluHIgBmBo.

Below is the derivative of the sigmoid function:

$$\frac{ds}{dx} = \frac{1}{1+e^{-x}}$$

$$= \left(\frac{1}{1+e^{-x}}\right)^2 \frac{d}{dx}\left(1+e^{-x}\right)$$

$$= \left(\frac{1}{1+e^{-x}}\right)^2 e^{-x}(-1)$$

$$= \left(\frac{1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right)(-e^{-x})$$

$$= \left(\frac{1}{1+e^{-x}}\right)\left(\frac{-e^{-x}}{1+e^{-x}}\right)$$

$$= s(x)(1-s(x))$$

Further details about sigmoid derivative are here.

Consider data below from source: https://datatofish.com/logistic-regression-python/

gmat	gpa	work_experience	admitted
780	4	3	1
750	3.9	4	1
690	3.3	3	0
710	3.7	5	1
680	3.9	4	0
730	3.7	6	1
690	2.3	1	0
720	3.3	4	1
740	3.3	5	1
690	1.7	1	0
610	2.7	3	0
690	3.7	5	1
710	3.7	6	1
680	3.3	4	0
770	3.3	3	1
610	3	1	0
580	2.7	4	0
650	3.7	6	1
540	2.7	2	0
590	2.3	3	0
620	3.3	2	1
600	2	1	0
550	2.3	4	0
550	2.7	1	0
570	3	2	0
670	3.3	6	1
660	3.7	4	1
580	2.3	2	0
650	3.7	6	1
660	3.3	5	1
640	3	1	0
620	2.7	2	0
660	4	4	1
660	3.3	6	1
680	3.3	5	1
650	2.3	1	0
670	2.7	2	0
580	3.3	1	0
590	1.7	4	0
690	3.7	5	1

Run PYTHON

```
1
2 #source: https://datatofish.com/logistic-regression-python/
3 import pandas as pd
4 from sklearn.model_selection import train_test_split
5 from sklearn.linear_model import LogisticRegression
6 from sklearn import metrics
7 import seaborn as sn
8 import matplotlib.pyplot as plt
9
10 candidates = {'gmat': [780,750,690,710,680,730,690,720,740,690,610,690,7
11               'gpa': [4,3.9,3.3,3.7,3.9,3.7,2.3,3.3,3.3,1.7,2.7,3.7,3.7,
12               'work_experience': [3,4,3,5,4,6,1,4,5,1,3,5,6,4,3,1,4,6,2,
13               'admitted': [1,1,0,1,0,1,0,1,1,0,0,1,1,0,1,0,0,1,0,0,1,0,0
14               }
```

Further reading: https://realpython.com/logistic-regression-python/

Logistic regression from scratch code below

Run PYTHON

```
1 #Source: https://machinelearningmastery.com/implement-logistic-regressio
2
3 from math import exp
4
5 # Make a prediction with coefficients
6 def predict(row, coefficients):
7     yhat = coefficients[0]
8     for i in range(len(row)-1):
9         yhat += coefficients[i + 1] * row[i]
10    return 1.0 / (1.0 + exp(-yhat))
11
12 # Estimate logistic regression coefficients using stochastic gradient de
13 def coefficients_sgd(train, l_rate, n_epoch):
14    coef = [0.0 for i in range(len(train[0]))]
```

Measurement of error

Watch the video and read the given content on measurement of error.

Tutorial video

Watch the tutorial video on measurement of error by clicking on the link below.

How to measure error?

There are a number of methods to measure errors in classification and regression problems.

There are different types of problems that would have different types of calculations for loss. Below are some prominent methods for calculating error. For classification problems, typically classification performance is reported in machine learning papers while in regression or time series problems use of the mean-squared-error or root-mean-squared error is quite prominent. It is useful to know other ways of calculating errors.

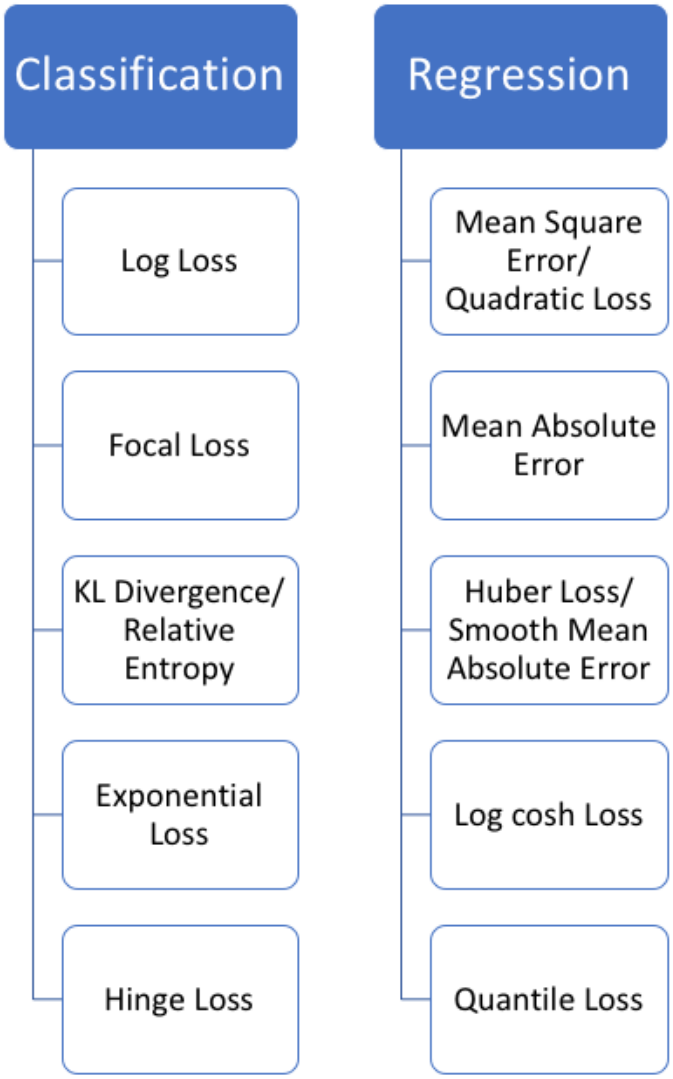


Figure: Losses in classification and regression. Adapted from "5 Regression Loss Functions All Machine Learners Should Know" by P. Grover, 2018. Retrieved from https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0.

Below are some examples of different ways to capture errors for different models.

Regression error functions

Some of the regression error functions which are very popular are:

- MSE - Mean Squared Error
- MAPE - Mean Absolute Percentage Error
- MAE - Mean Absolute Error

Consider the figure below that shows a linear regression model fitting into data (outcomes).

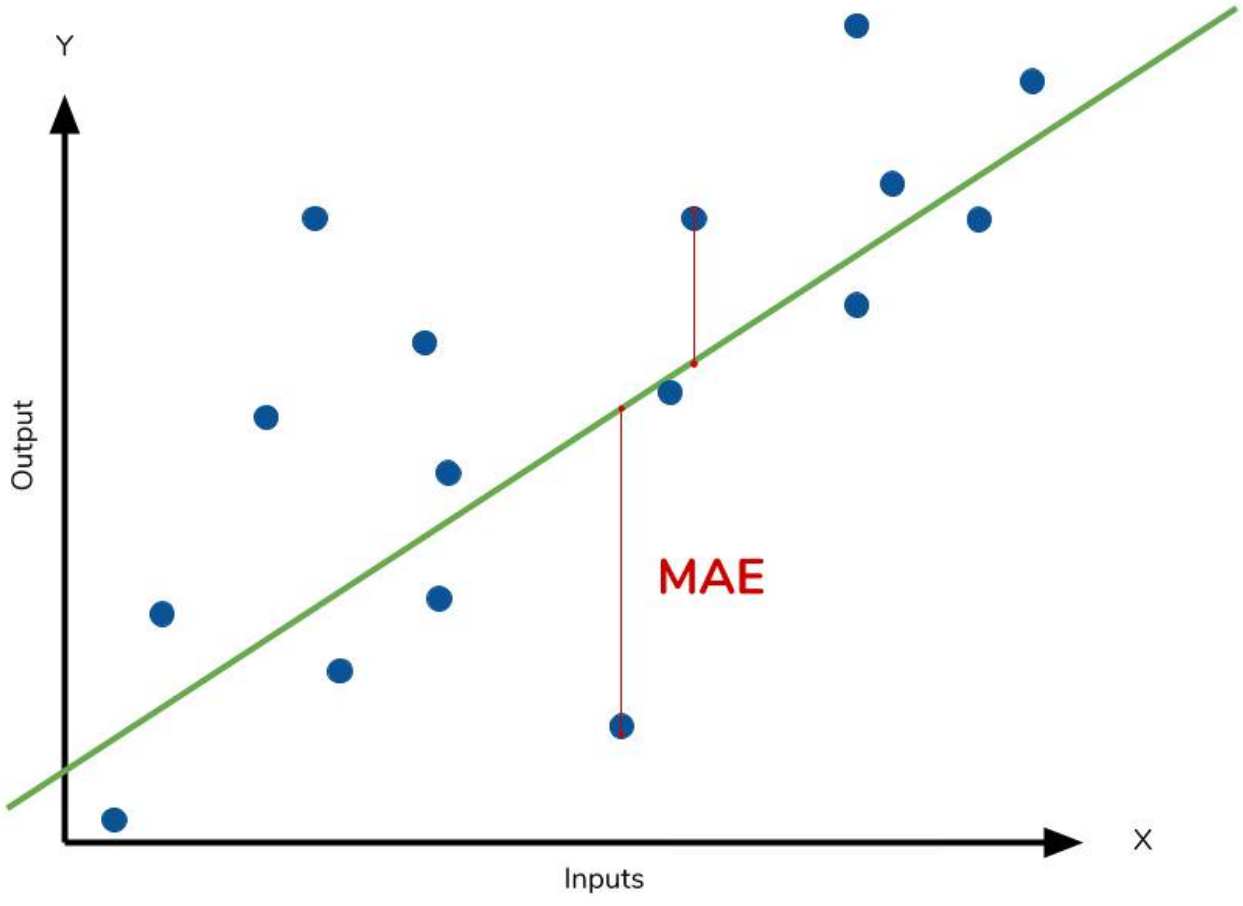
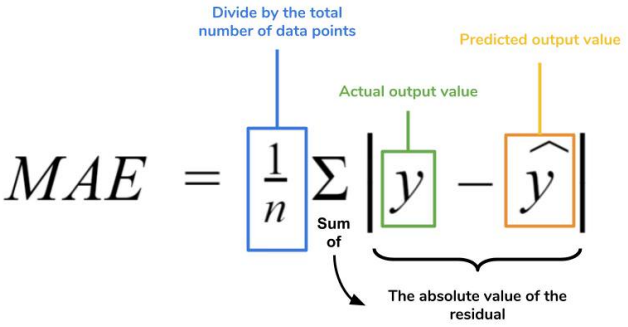
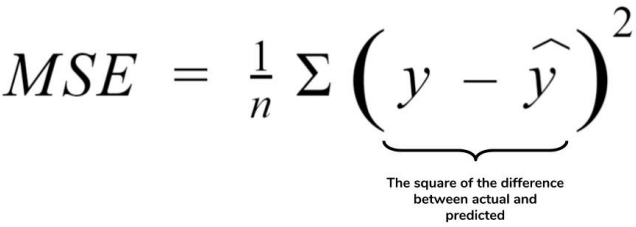


Figure: A linear regression model fitting into data. Adapted from "Tutorial: Understanding Regression Error Metrics in Python" by Pascual, n.d. Retrieved from https://www.dataquest.io/blog/understanding-regression-error-metrics/.

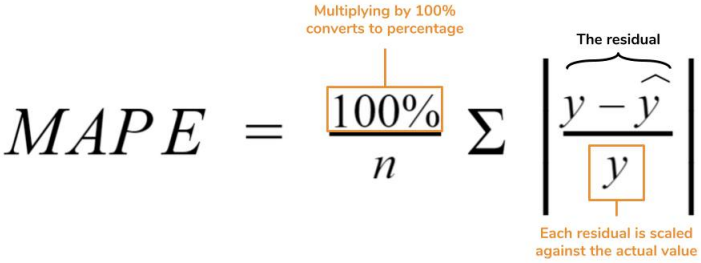
MAE function is



MSE function is



MAPE function is



RMSE function is

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{T}}.$$

Pascual, C. (n.d). Tutorial: Understanding Regression Error Metrics in Python. Retrieved from https://www.dataquest.io/blog/understanding-regression-error-metrics/.

While it may be standard to use RMSE, MSE, MAE for regression problems, log loss could be used for visualisation of how the error is decreasing over time in regression problems.

https://medium.com/analytics-vidhya/understanding-the-loss-function-of-logistic-regression-ac1eec2838ce

Furthermore, we note that its standard to use log loss mostly for classification problems. Note logistic regression model is typically used for classification but it can also be used for regression. We also note that SSE, RMSE, MSE can also be used to show error trend while training in classification problems.

What can be used and what is used as a standard can be different. These days, logistic regression models are used as a standard for classification problems, and it features log loss error function.

It is important to note the difference in terms and usage of models and error functions that are mostly used and become industry standards.

Run PYTHON

```
import numpy as np
def loss(h, y):
    return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

h= np.random.rand(5)
print(h, ' h')
y= np.random.rand(5)
print(y, ' y')

log_loss = loss(h, y) # case of regression or prediction problem
print(log_loss)

y_ = np.ones(5)
print(y_, ' y_')
```

The next lesson presents a notebook with examples of loss functions.

Bias and variance

Read the following content on bias and variance in order to understand the common terms used in model prediction.

Bias

The difference between the prediction of the values by the model and the actual value (from the data set) is known as the bias. A high bias gives a large error in training and testing data and in order to avoid the problem of underfitting, the training method should ensure low bias. A low bias, on the other hand, will result in overfitting. Strategies are required to deal with training in order to ensure good performance on generalisation or test data sets.

Variance

The variability of model prediction for a given data point which tells us the spread of our data is called the variance of the model. A model with high variance has a very complex fit to the training data and thus is not able to fit accurately on data that it hasn't seen before. As a result, such models perform very well on training data but have high error rates on test data.

When a model is high on variance, it is said to be **overfitting of data**. Overfitting is fitting the training set accurately via complex curve and high order hypothesis but is not the solution as the error with unseen data is high. While training a data model variance should be kept low.

The figure below provides an example of overfitting as all the points in the data in blue are covered by the model outputs in red. This is a problem as a model that fits the training data set extremely well typically has problems in giving a good test or generalisation performance.

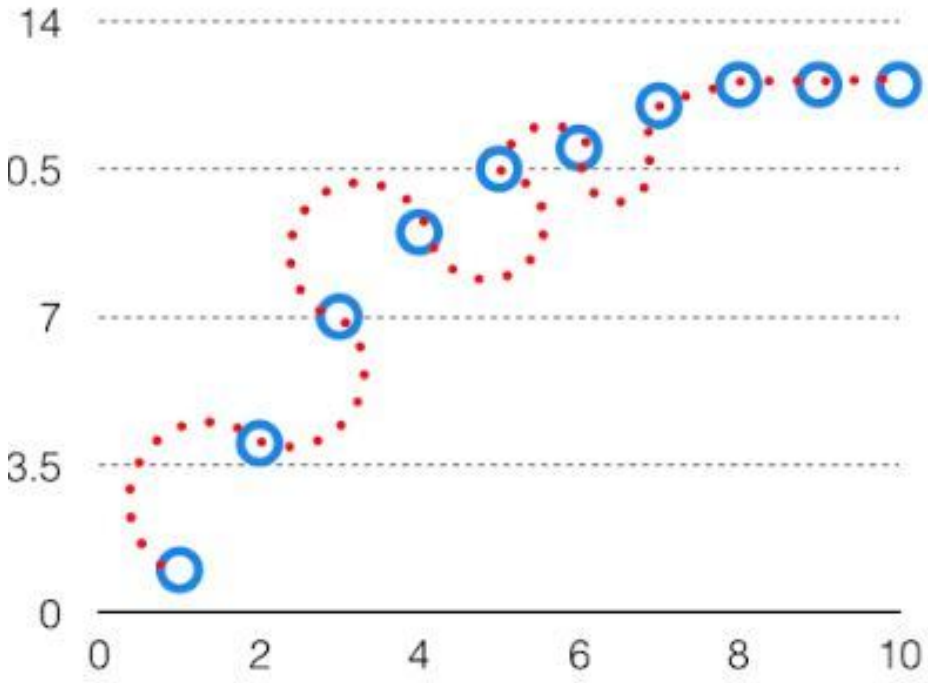


Figure: Overfitting of data points. Adapted from "Bias-Variance Trade off – Machine Learning" by Geeksforgeeks, n.d. Retrieved from https://www.geeksforgeeks.org/ml-bias-variance-trade-off/?ref=leftbar-rightbar.

Let's examine the trade-offs between bias and variance and overfitting and underfitting a model.

Bias variance trade-off

The below figure shows a trade-off between variance and bias. Because the model captures some of the data points in blue and leaves the others, this could typically provide better generalisation performance when compared to the previous example.

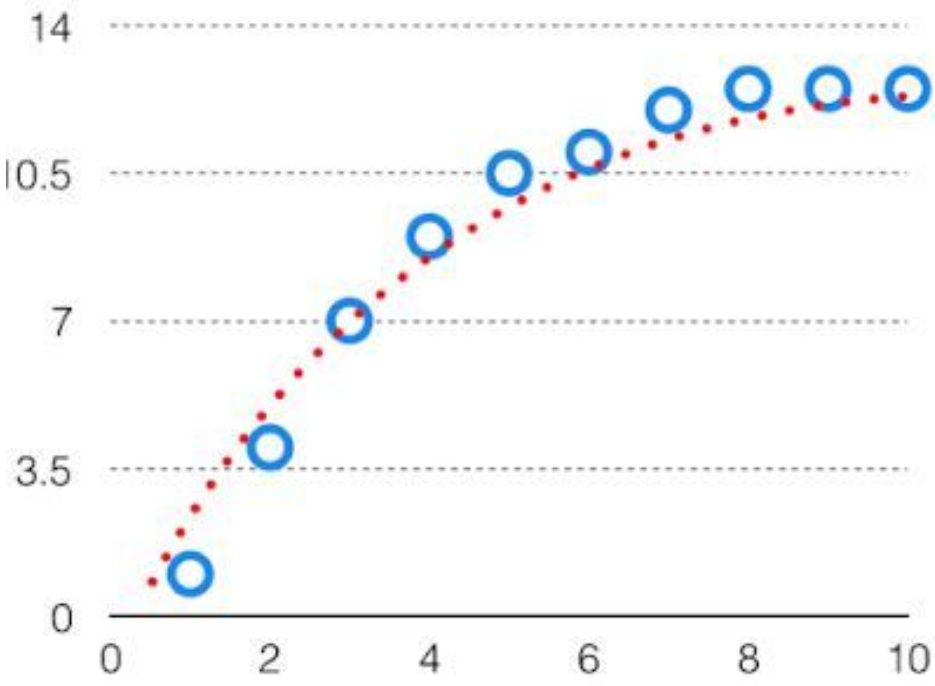


Figure: Bais Variance trade-off. Adapted from "Bias-Variance Trade off – Machine Learning" by Geeksforgeeks, n.d. Retrieved from https://www.geeksforgeeks.org/ml-bias-variance-trade-off/?ref=leftbar-rightbar.

Trade-off between overfitting and underfitting a model

The below figure shows that a trade-off between overfitting and underfitting a model is needed in order to have a good test or generalisation performance. It shows the bias and variance and how they relate to the increase/decrease of error in the y-axis. It is difficult to know when it's best to stop training and hence several trial experiments are needed.

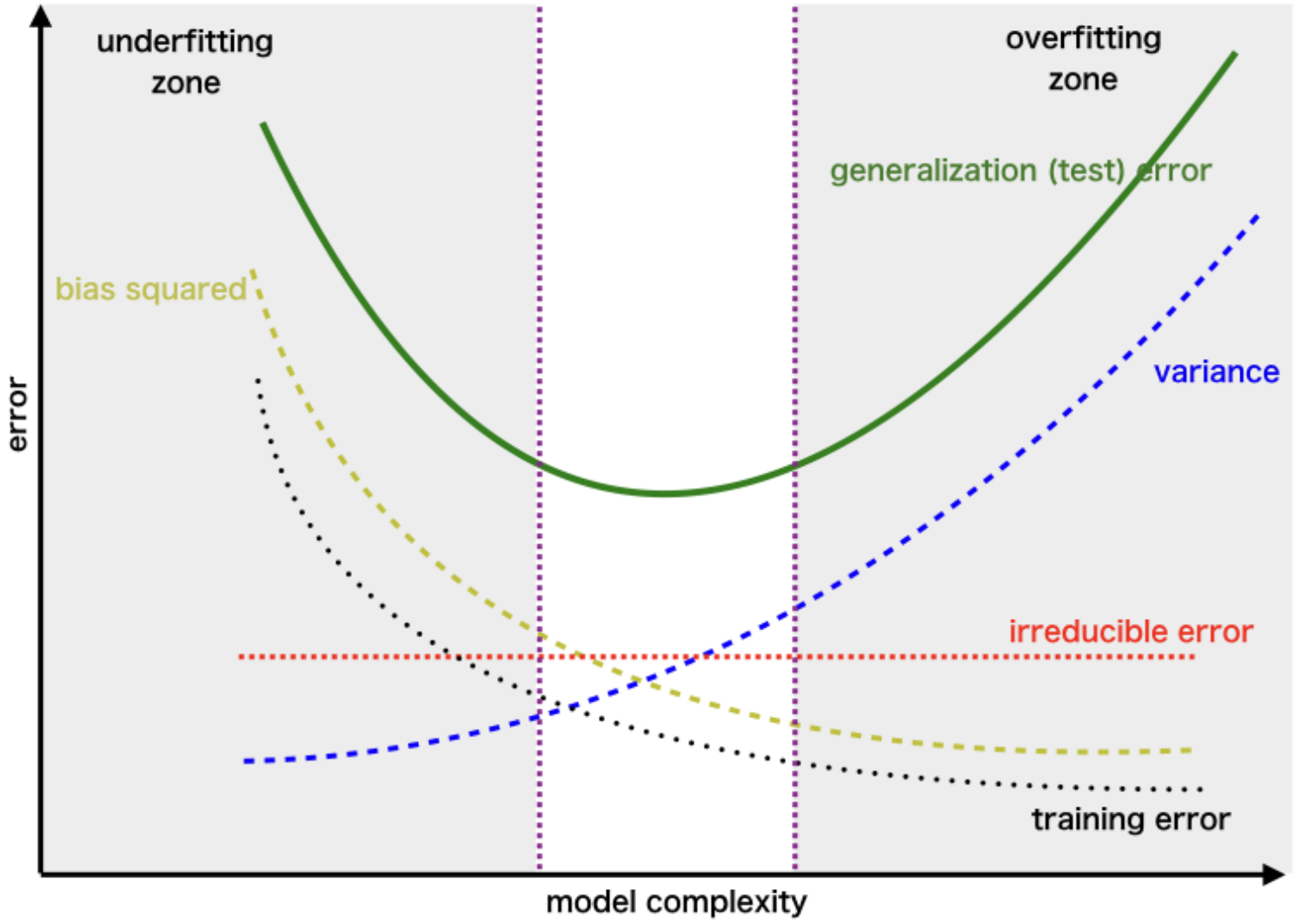


Figure: Model complexity vs error. Adapted from "Bias-Variance Trade off – Machine Learning" by Geeksforgeeks, n.d. Retrieved from https://www.geeksforgeeks.org/ml-bias-variance-trade-off/?ref=leftbar-rightbar.

Typically a **validation data set** is used that tests the model performance during training against the validation data set in order to stop training to address the bias-variance problem. The validation data set is typically taken from the test data set. Hence, the validation data set is a partially seen data set; it's not used for training but used to determine when to stop training given some error level is achieved on the validation data set. Further information is given here.

Cross validation

Read the following content on cross validation and complete the associated activities.

In any sports event, you have training sessions and actual competition sessions. In machine learning, we have similar situations. In a sporting event, consider that you need to test how strong your team will be playing against a team you have not met before. During training, you play against your teammates.

Similarly, in machine learning, you need to test how good your model will be when provided with a data set it has not seen prior to the training. This shows the quality or validity of your model so that you can convince clients, investors, or the company that uses your model to adopt it. In this slide, you will learn how to cross validate the models.

Cross validation is a means to check the quality of your model's predictions on unseen or test data. In many cases, there are no separate training and testing datasets. Hence, you need to divide your single data set into training and testing batches. However, it is important to know which section of the data you should use for training and which for testing.

There are different methods of cross-validation. N-fold cross-validation is one of the most popular ones in the data mining literature.

Watch the video below that gives an overview of cross validation.

Machine learning fundamentals: Cross validation

Statquest. (2018, August 24). *Machine Learning Fundamentals: Cross Validation* [online video]. Retrieved from https://www.youtube.com/watch?v=fSytzGwwBVw.

N-fold cross validation

In this case, N refers to the number of groups to split the given data set; hence, the procedure is called N-fold cross-validation. At times, a specific value for N is chosen, such as N=10 which would make it 10-fold cross-validation. The method is also called k-fold cross-validation. The below diagrams show how you can use cross validation strategy to divide the data. You can do multiple experiments to test the validity of your model and provide the mean and standard decisions in your results. This will also show statistical validity to indicate a measure of uncertainty in your model predictions.

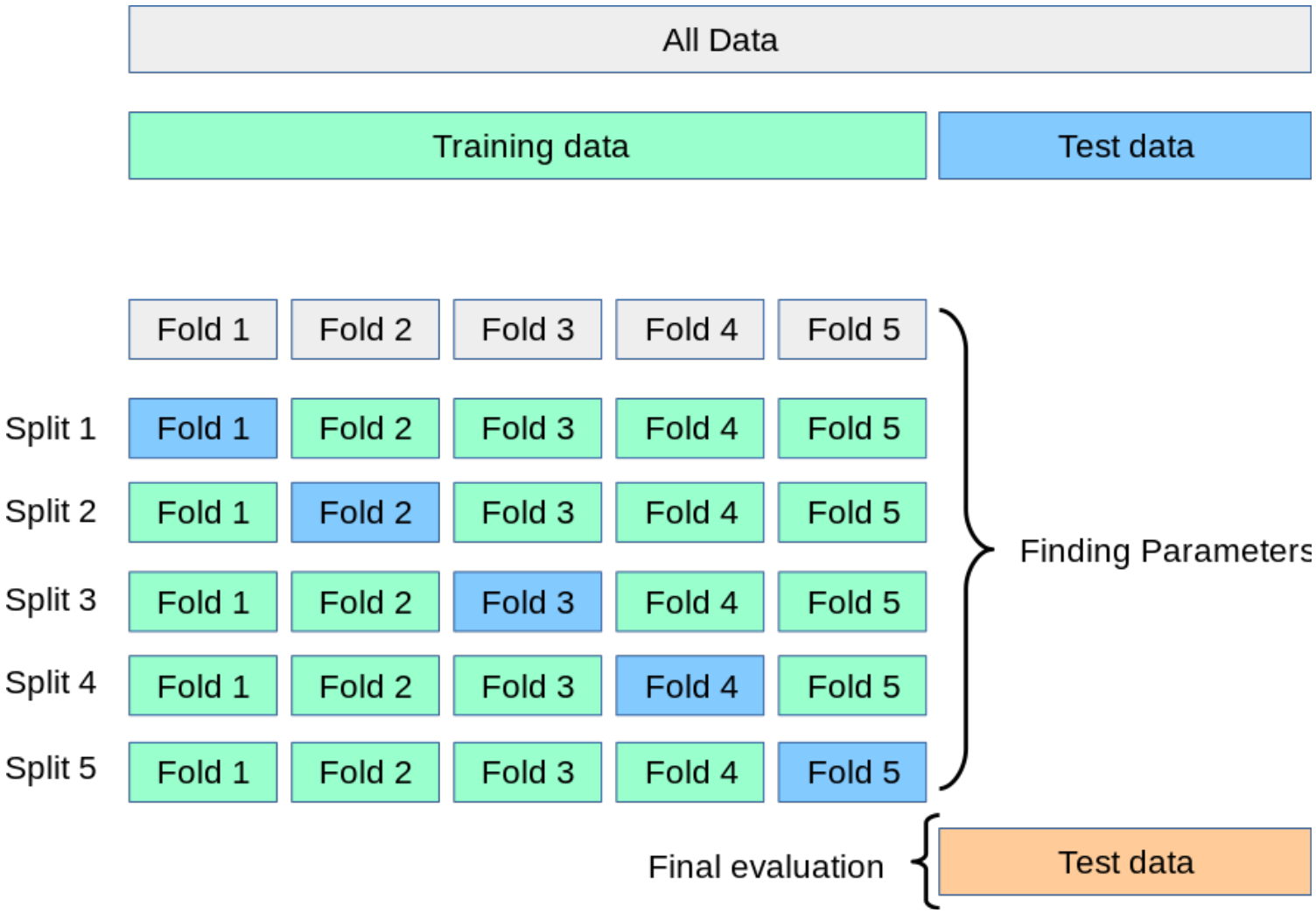


Figure N-cross validation. Adapted from "Cross-validation: evaluating estimator performance" by scikit-learn, n.d. Retrieved from https://scikit-learn.org/stable/modules/cross_validation.html.

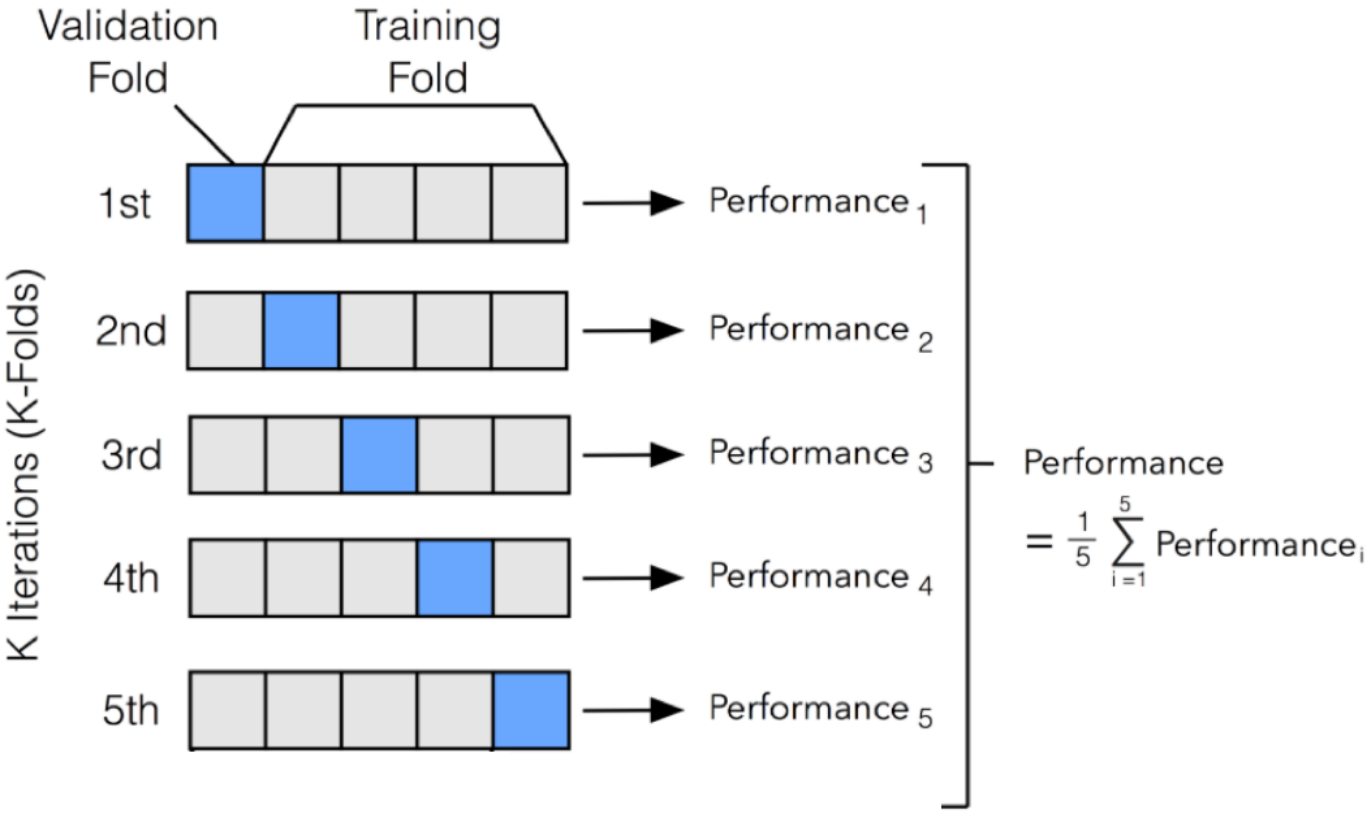


Figure: N-cross validation approach. Adapted from "Model selection" by Github, n.d. Retrieved from http://ethen8181.github.io/machine-learning/model_selection/model_selection.html.

The validation set approach

The validation set approach consists of randomly splitting the data into two sets, one set is used to train the model and the remaining other set is used to test the model.

This approach works as follows:

- Build (train) the model on the training data set
- Apply the model to the test data set to predict the outcome of new unseen observations
- Quantify the prediction error as the mean squared difference between the observed and the predicted outcome values.

The first code splits the `swiss` data set so that 80% is used to train a linear regression model and 20% is used to evaluate the model performance. The second code creates X (features) and y (response) using train/test split with different random state values and checks the classification accuracy of the model.

Practise the cross validation using R. Run the code and see the outputs.

Run R

```
1
2 library(tidyverse)
3 library(caret)
4
5 # Load the data
6 data("swiss")
7 # Inspect the data
8 sample_n(swiss, 3)
9
10
11 # Split the data into training and test set
12 set.seed(123)
13 training.samples <- swiss$Fertility %>%
14   createDataPartition(p = 0.8, list = FALSE)
```

Run PYTHON

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
# read in the iris data
iris = load_iris()

# create X (features) and y (response)
X = iris.data
y = iris.target
# use train/test split with different random_state values
# we can change the random_state values that changes the accuracy scores
# the accuracy changes a lot
# this is why testing accuracy is a high-variance estimate
```

Leave one out cross validation (LOOCV)

Similar to the n-fold cross-validation strategy, we have LOOCV with some differences as given below:

1. Leave one out data point and build the model on the rest of the data set.
2. Test the model against the data point that is left out in step 1 and record the test error associated with the prediction.
3. Repeat the process for all data points.
4. Compute the overall prediction error by taking the average of all these test error estimates recorded in step 2.

Practise the 'leave one out cross-validation' using R. Run the code and see the outputs.

Run R

```r

library(tidyverse)
library(caret)

# Load the data
data("swiss")
# Inspect the data
sample_n(swiss, 3)
# Define training control
train.control <- trainControl(method = "LOOCV")
# Train the model
model <- train(Fertility ~., data = swiss, method = "lm",
               trControl = train.control)
# Summarize the results
```

K-fold cross-validation

The k-fold cross-validation method evaluates the model performance on a different subset of the training data and then calculates the average prediction error rate. The algorithm is as follows:

1. Randomly split the data set into k-subsets (or k-fold) (for example 5 subsets).
2. Reserve one subset and train the model on all other subsets.
3. Test the model on the reserved subset and record the prediction error.
4. Repeat this process until each of the k subsets has served as the test set.
5. Compute the average of the k recorded errors.

This is called the cross-validation error serving as the performance metric for the model.

Run the code and examine the outputs.

Run PYTHON

```
1
2
3 from sklearn import datasets, linear_model
4
5 from sklearn.model_selection import cross_validate
6
7 from sklearn.model_selection import KFold
8 from sklearn import metrics
9
10
11 #https://scikit-learn.org/stable/modules/generated/sklearn.model_selecti
12 # simulate splitting a dataset  into 5 folds
13 diabetes = datasets.load_diabetes()
14 X = diabetes.data[:150]
```

Run the below R code to understand the function of k-fold cross-validation.

```r
#Source: STHDA. (2018). Cross-Validation Essentials in R. http://www.sth

library(tidyverse)
library(caret)

# Load the data
data("swiss")
# Inspect the data
sample_n(swiss, 3)

# Define training control
set.seed(123)
train.control <- trainControl(method = "cv", number = 10)
# Train the model
```

Validation set approach

Validation sets are useful in preventing models from overfitting. The below image explains the validation set approach where the data is split into train, validation and test sets. The validation data is used to provide an unbiased evaluation of a model on the training data set. The model occasionally **sees** validation data to determine when to stop but never uses it for training.

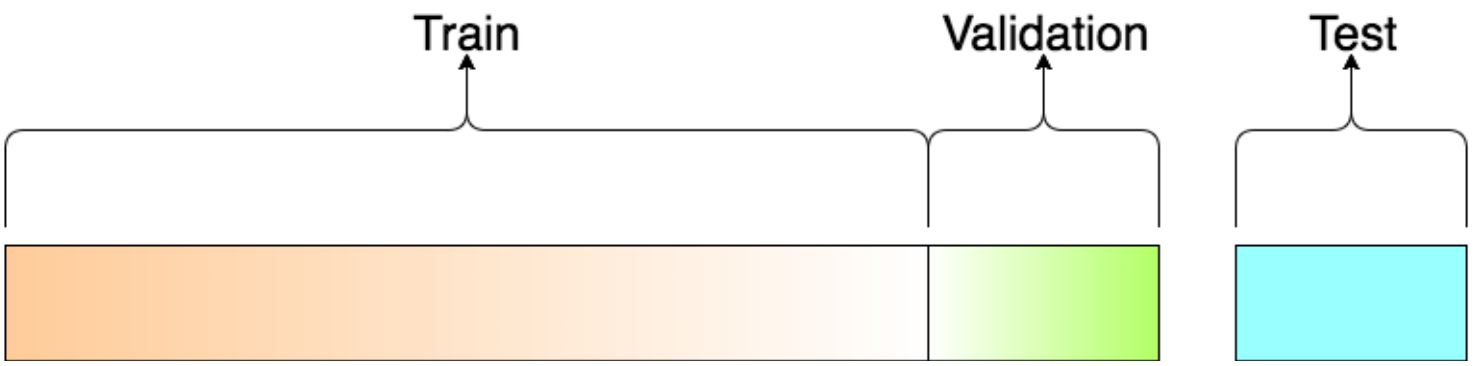


Figure: About Train. Adapted from "Validation and Test Sets in Machine Learning" by Towards data science, 2017. Retrieved from https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7.

Regularization

A major problem in the linear and logistic regression model is overfitting and to avoid it, we need to use additional techniques (e.g. cross-validation, regularization, early stopping, and pruning). Some of these methods will be covered in Week 4 for the case of neural networks.

A way for bias-variance trade off is regularization which is a way for tuning the complexity of the model that helps in datasets that feature noise and prevents over fitting.

In the case of linear and logistic models, the major forms are by adding a type of penalty in the cost function with lasso (L1) and ridge (L2) regularisation methods. We need to note we have a parameter, known as the regularisation parameter which is greater than 0 and needs to be manually tuned. Consider the error surface below.

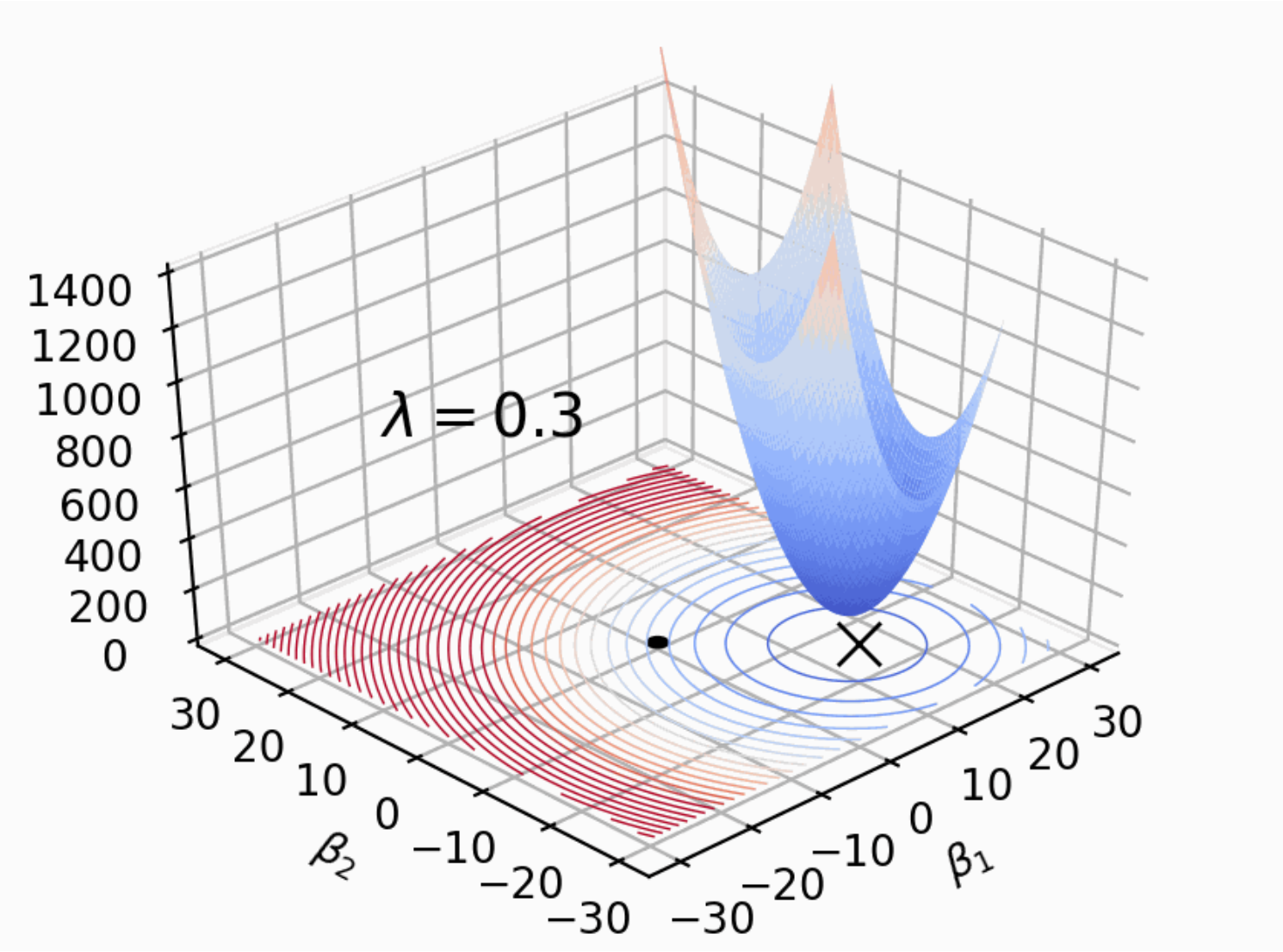


Figure Source: https://explained.ai/regularization/index.html

Lasso (L1) regularization: consider the following penalty added to the cost function:

$$L_1 = (wx + b - y)^2 + \lambda|w|$$

We need to note that the derivative of the absolute value is 1 or -1, except where w =0

$$\frac{d|w|}{dw} = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \end{cases}$$

Hence our new gradients would be

$$\begin{aligned} w_{\text{new}} &= w - \eta \frac{\partial L_1}{\partial w} \\ &= w - \eta \cdot \left[2x(wx + b - y) + \lambda \frac{d|w|}{dw}\right] \\ &= \begin{cases} w - \eta \cdot \left[2x(wx + b - y) + \lambda\right] & w > 0 \\ w - \eta \cdot \left[2x(wx + b - y) - \lambda\right] & w < 0 \end{cases} \end{aligned}$$

$$w_{\text{new}} = \begin{cases} (w - H) - \lambda, & w > 0 \quad \text{—— (1.1)} \\ (w - H) + \lambda, & w < 0 \quad \text{—— (1.2)} \end{cases}$$

Ridge (L2) regularisation:

Consider the following penalty added to the cost function

$$L_2 = (wx + b - y)^2 + \lambda w^2$$

$$\begin{aligned} w_{\text{new}} &= w - \eta \frac{\partial L_2}{\partial w} \\ &= w - \eta \cdot [2x(wx + b - y) + 2\lambda w] \end{aligned}$$

Source and more information: https://towardsdatascience.com/intuitions-on-l1-and-l2-regularisation-235f2db4c261

Elastic net regularisation: Combination of ridge and Lasso regularisation.

Source: https://www.youtube.com/watch?v=Xm2C_gTAl8c&t=402s

Run PYTHON

```python
#Source: https://scikit-learn.org/stable/auto_examples/linear_model/plot
# Authors: Alexandre Gramfort <alexandre.gramfort@inria.fr>
#          Mathieu Blondel <mathieu@mblondel.org>
#          Andreas Mueller <amueller@ais.uni-bonn.de>
# License: BSD 3 clause

import numpy as np
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.preprocessing import StandardScaler

X, y = datasets.load_digits(return_X_y=True)
```

Further reading:

1. Discussion for R with logistic regression: https://www.machinegurning.com/rstats/regularised-logistic-regression/
2. Scikit-learn: https://www.bogotobogo.com/python/scikit-learn/scikit-learn_logistic_regression.php
3. Elastic net: https://www.machinecurve.com/index.php/2020/01/21/what-are-l1-l2-and-elastic-net-regularization-in-neural-networks/
4. https://courses.cs.washington.edu/courses/cse446/13sp/slides/logistic-regression-gradient.pdf
5. log loss http://users.umiacs.umd.edu/~hcorrada/PML/homeworks/HW04_solutions.pdf