

Week 2: Logistic Regression and Evaluation

Introduction

This week introduces you to data mining and machine learning. You will develop and evaluate machine learning models using logistic regression models with gradient descent learning algorithms. You will then evaluate the performance of the models using evaluation metrics. The exercises will provide you with opportunities to reflect on your acquired knowledge and skills.

Find out about the topics covered in this week by watching the below video.

Recommended readings

We recommend you to **first go through the lessons and then see the related topics in the textbook**. Note that **the textbook readings are not mandatory**. They are meant to provide additional information for this week's lessons.

The following chapters of the course textbook will help you with the topics covered this week:

Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Ed.). O'Reilly Media, Inc.

- Chapter 1: The Machine Learning Landscape
- Chapter 2: End-to-End Machine Learning Project
- Chapter 3: Classification
- Chapter 4: Training Models

Mitchell, T. (1997). Machine Learning. McGraw-Hill.

- Chapter 4: Neural Networks

All readings are available from the course [Leganto reading list](#).

This week's lessons were prepared by Dr. R. Chandra. If you have any questions or comments, please email directly: rohitash.chandra@unsw.edu.au.

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.

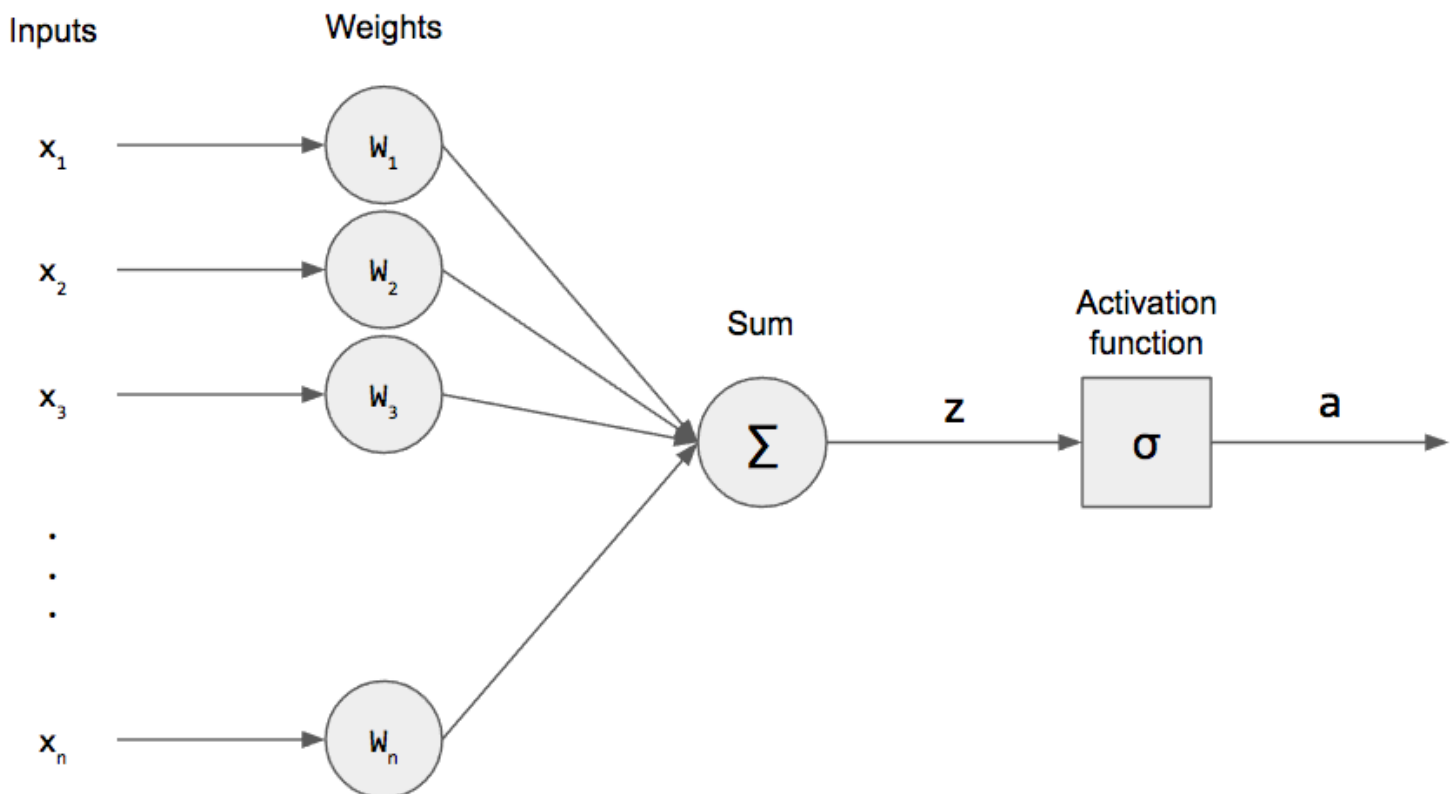


Figure: Linear regression model. Adapted from Nginx, n.d. Retrieved from <https://static.edusercontent.com/files/AnSeb7ymghifF0PZjICL4c7F>.

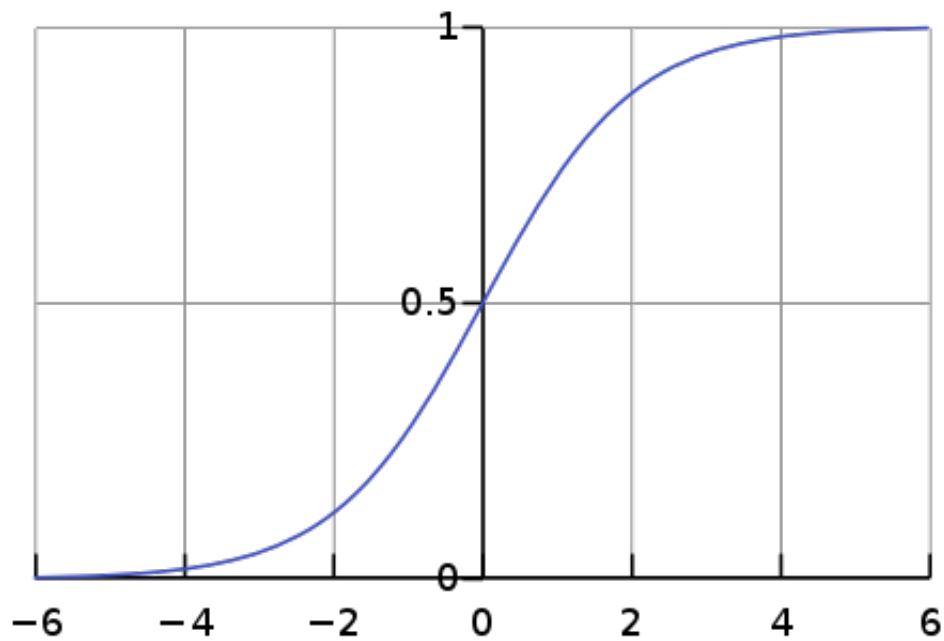


Figure: Sigmoid or logistic function. Adapted from "Sigmoid function" by Wikipedia, 2020. Retrieved from https://en.wikipedia.org/wiki/Sigmoid_function.

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

```
import numpy as np
import math

x = np.linspace(-4, 4, 5)
z = 1/(1 + np.exp(-x))

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.

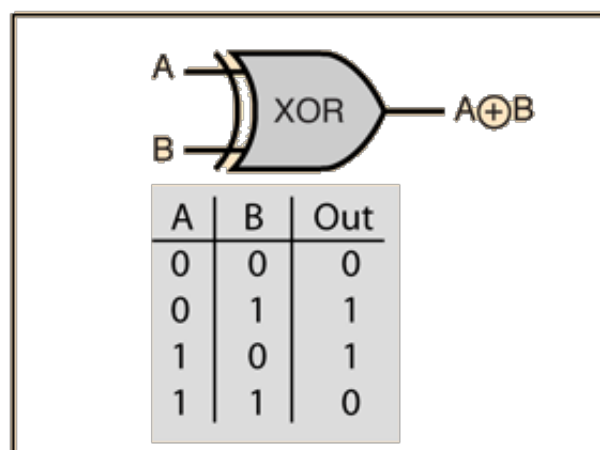


Figure: Two input XOR gate. Adapted from "HyperPhysics" by R.Nave, n.d. Retrieved from <http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/xor.html>.

i 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

An error occurred.

Try watching this video on [www.youtube.com](https://www.youtube.com/watch?v=5g0TPrxKK6o), or enable JavaScript if it is disabled in your browser.

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

An error occurred.

Try watching this video on [www.youtube.com](https://www.youtube.com/watch?v=jbluHlgBmBo), or enable JavaScript if it is disabled in your browser.

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

Below is the derivative of the sigmoid function:

$$\begin{aligned}
\frac{ds}{dx} &= \frac{1}{1 + e^{-x}} \\
&= \left(\frac{1}{1 + e^{-x}} \right)^2 \frac{d}{dx} (1 + e^{-x}) \\
&= \left(\frac{1}{1 + e^{-x}} \right)^2 e^{-x} (-1) \\
&= \left(\frac{1}{1 + e^{-x}} \right) \left(\frac{1}{1 + e^{-x}} \right) (-e^{-x}) \\
&= \left(\frac{1}{1 + e^{-x}} \right) \left(\frac{-e^{-x}}{1 + e^{-x}} \right) \\
&= s(x) (1 - s(x))
\end{aligned}$$

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

```
#source: https://datatofish.com/logistic-regression-python/
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import seaborn as sn
```

```

import matplotlib.pyplot as plt

candidates = {'gmat': [780,750,690,710,680,730,690,720,740,690,610,690,710,680,770,610,580,650,540,
                    'gpa': [4,3.9,3.3,3.7,3.9,3.7,2.3,3.3,3.3,1.7,2.7,3.7,3.7,3.3,3.3,3,2.7,3.7,2.7,2.3,3
                    'work_experience': [3,4,3,5,4,6,1,4,5,1,3,5,6,4,3,1,4,6,2,3,2,1,4,1,2,6,4,2,6,5,1,2,4
                    'admitted': [1,1,0,1,0,1,0,1,1,0,0,1,1,0,1,1,0,0,1,0,0,0,1,0,0,0,0,1,1,0,1,1,0,0,1,1,1,0,
                    }

df = pd.DataFrame(candidates,columns= ['gmat', 'gpa','work_experience','admitted'])

X = df[['gmat', 'gpa','work_experience']]
y = df['admitted']

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=0) #in this case,

logistic_regression= LogisticRegression()
logistic_regression.fit(X_train,y_train)

y_pred_train=logistic_regression.predict(X_train)

y_pred=logistic_regression.predict(X_test)

confusion_matrix = pd.crosstab(y_train, y_pred_train, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
print('Accuracy: ',metrics.accuracy_score(y_train, y_pred_train))
plt.savefig('confmat_train.png')
plt.clf()

confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
print('Accuracy: ',metrics.accuracy_score(y_test, y_pred))
plt.savefig('confmat_tes.png')

new_candidates = {'gmat': [590,740,680,610,710],
                  'gpa': [2,3.7,3.3,2.3,3],
                  'work_experience': [3,4,6,1,5]
                  }

df2 = pd.DataFrame(new_candidates,columns= ['gmat', 'gpa','work_experience'])
y_pred=logistic_regression.predict(df2)

print (df2)
print (y_pred)

```


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

Logistic regression from scratch code below

```
#Source: https://machinelearningmastery.com/implement-logistic-regression-stochastic-gradient-descent-from-scratch-in-python/

from math import exp

# Make a prediction with coefficients
def predict(row, coefficients):
    yhat = coefficients[0]
    for i in range(len(row)-1):
        yhat += coefficients[i + 1] * row[i]
    return 1.0 / (1.0 + exp(-yhat))

# Estimate logistic regression coefficients using stochastic gradient descent
def coefficients_sgd(train, l_rate, n_epoch):
    coef = [0.0 for i in range(len(train[0]))]
    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
            yhat = predict(row, coef)
            error = row[-1] - yhat
            sum_error += error**2
            coef[0] = coef[0] + l_rate * error * yhat * (1.0 - yhat)
            #print(row, yhat, error, sum_error)
            for i in range(len(row)-1):
                coef[i + 1] = coef[i + 1] + l_rate * error * yhat * (1.0 - yhat) * row[i]
        print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
    return coef

# Calculate coefficients
dataset = [[2.7810836,2.550537003,0],
           [1.465489372,2.362125076,0],
           [3.396561688,4.400293529,0],
           [1.38807019,1.850220317,0],
           [3.06407232,3.005305973,0],
           [7.627531214,2.759262235,1],
           [5.332441248,2.088626775,1],
           [6.922596716,1.77106367,1],
           [8.675418651,-0.242068655,1],
           [7.673756466,3.508563011,1]]

l_rate = 0.3
n_epoch = 10
coef = coefficients_sgd(dataset, l_rate, n_epoch)
print(coef)
```

Code challenge is to convert this by using numpy arrays.



Source: <https://machinelearningmastery.com/implement-logistic-regression-stochastic-gradient-descent-scratch-python/>




Dataset: <https://www.kaggle.com/kumargh/pima-indians-diabetes-csv?select=pima-indians-diabetes.csv>

Linear and Logistic Regression

This code slide does not have a description.

Logistic regression using scikit-learn

 Apply logistic regression using the scikit-learn library.

Running this code will help you understand logistic regression using the scikit-learn library. The purpose of this example is for you to learn how to use existing data sets in building models, not just models from scratch as demonstrated previously but building models using well-known machine learning libraries such as scikit-learn. More details of the scikit-learn library can be found [here](#).

There are other examples of data sets, such as the Heart Disease data set which can be used. UCI machine learning repository is popularly used for data mining and machine learning. You can download and apply logistic regression using scikit-learn.

- [Heart Disease data set](#)
- Bank-note data
- Wisconsin Breast Cancer Dataset

We are restricting to binary classification datasets in this example.

Logistic regression using gradient descent in R



Apply logistic regression using gradient descent in R.

Let's run an example of logistic regression using gradient descent in R. Note that you are free to choose either Python or R for assessments, but it is useful to see examples in both languages.

The code further demonstrates the use of gradient descent learning for logistic regression, showing how the gradients are computed and used for weight update. You can see the gradient function and sigmoid function that is used for training the model. Note that a synthetic training example is created as you run the code and you will see the following visualisation.

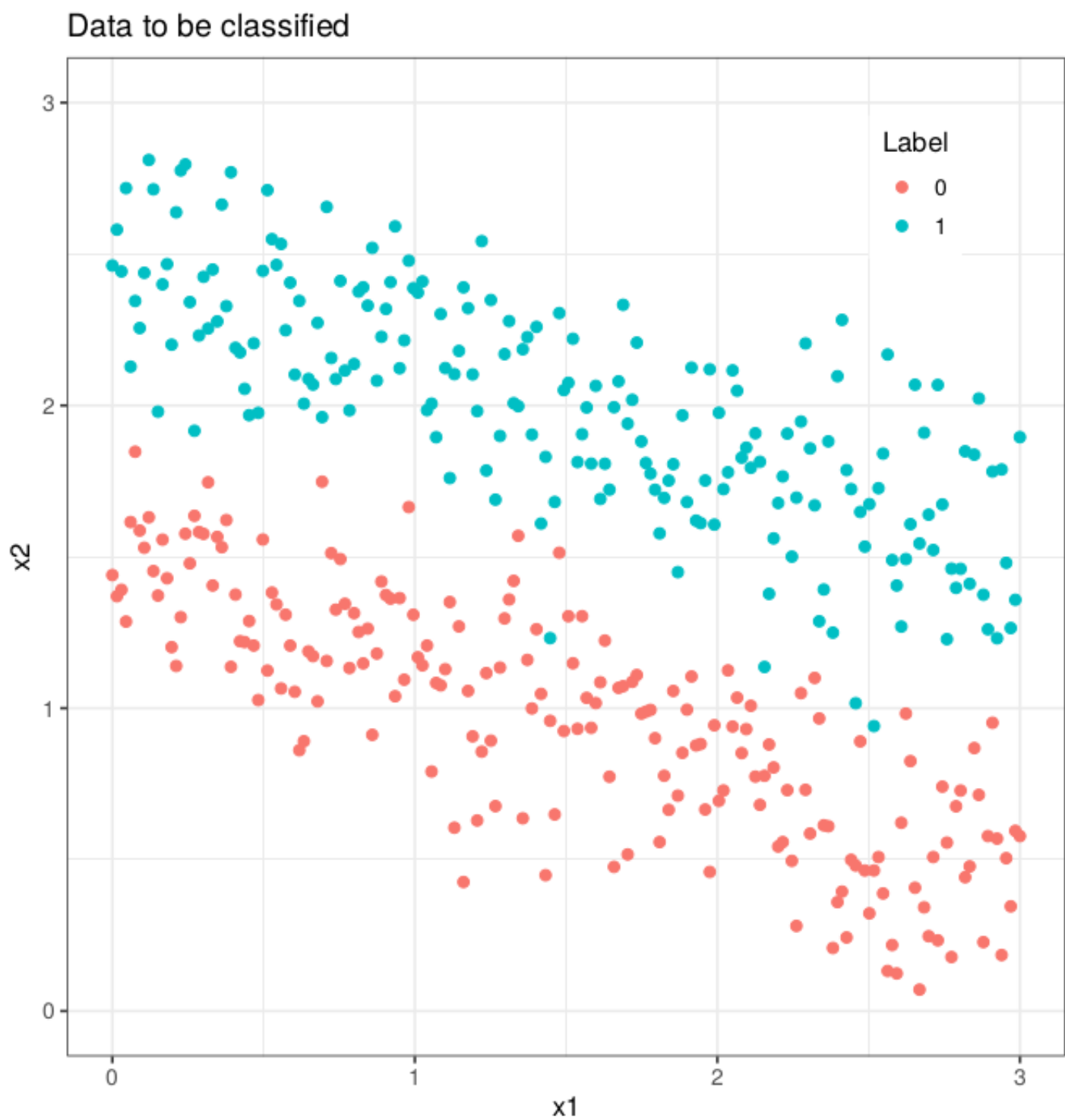


Figure: Data to be classified. Adapted from "Logistic Regression from Scratch in R" by M. Jun, 2019. Retrieved from <https://towardsdatascience.com/logistic-regression-from-scratch-in-r-b5b122fd8e83>.

The result after running the code is shown below.

Decision Boundary for Logistic Regression

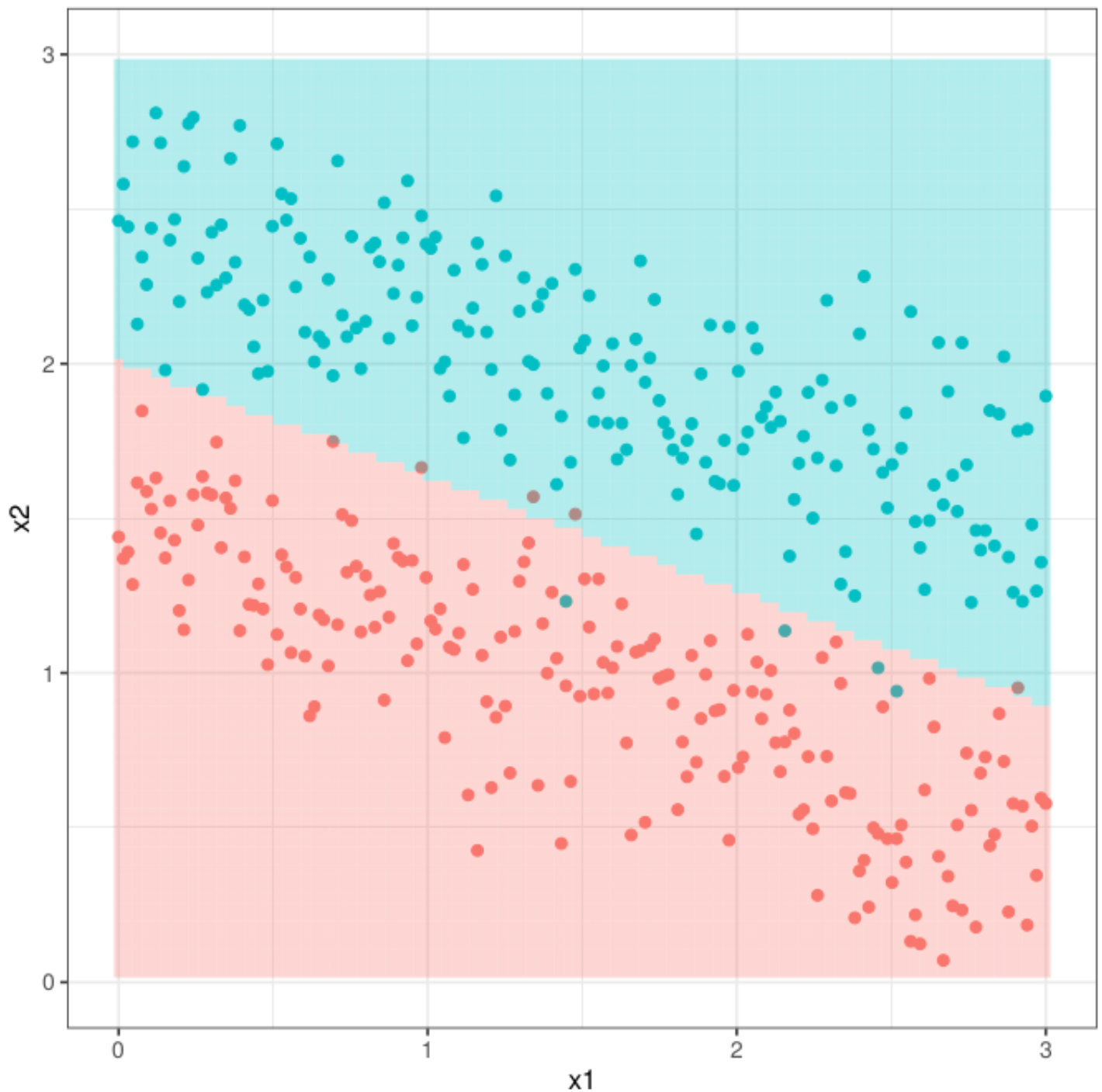


Figure: Application of logistic regression on the data Adapted from "Logistic Regression from Scratch in R" by M. Jun, 2019. Retrieved from <https://towardsdatascience.com/logistic-regression-from-scratch-in-r-b5b122fd8e83>.

You can print and visualise the data set in other ways and can try using the Iris, Wine, and Heart Disease examples that have been mentioned in the previous Python example.



Practise logistic regression using R.

```
#source: https://github.com/JunWorks/Logistic-Regression-from-scratch-in-R/blob/master/logistic_reg
# tutorial source: https://towardsdatascience.com/logistic-regression-from-scratch-in-r-b5b122fd8e83
```

```

library(ggplot2)
library(dplyr)

N <- 200 # number of points per class
D <- 2 # dimensionality, we use 2D data for easy visualization
K <- 2 # number of classes, binary for logistic regression
X <- data.frame() # data matrix (each row = single example, can view as xy coordinates)
y <- data.frame() # class labels

set.seed(56)

for (j in (1:K)){
  # t, m are parameters of parametric equations x1, x2
  t <- seq(0,1,length.out = N)
  # add randomness
  m <- rnorm(N, j+0.5, 0.25)
  Xtemp <- data.frame(x1 = 3*t , x2 = m - t)
  ytemp <- data.frame(matrix(j-1, N, 1))
  X <- rbind(X, Xtemp)
  y <- rbind(y, ytemp)
}

data <- cbind(X,y)
colnames(data) <- c(colnames(X), 'label')

# lets visualize the data:
ggplot(data) + geom_point(aes(x=x1, y=x2, color = as.character(label)), size = 2) +
  scale_colour_discrete(name = "Label") +
  ylim(0, 3) + coord_fixed(ratio = 1) +
  ggtitle('Data to be classified') +
  theme_bw(base_size = 12) +
  theme(legend.position=c(0.85, 0.87))

#sigmoid function, inverse of logit
sigmoid <- function(z){1/(1+exp(-z))}

#cost function
cost <- function(theta, X, y){
  m <- length(y) # number of training examples
  h <- sigmoid(X %*% theta)
  J <- (t(-y)%*%log(h)-t(1-y)%*%log(1-h))/m
  J
}

#gradient function
grad <- function(theta, X, y){
  m <- length(y)

  h <- sigmoid(X%*%theta)
  grad <- (t(X)%*%(h - y))/m
  grad
}

```



```

}

logisticReg <- function(X, y){
  #remove NA rows
  X <- na.omit(X)
  y <- na.omit(y)
  #add bias term and convert to matrix
  X <- mutate(X, bias =1)
  #move the bias column to col1
  X <- as.matrix(X[, c(ncol(X), 1:(ncol(X)-1))])
  y <- as.matrix(y)
  #initialize theta
  theta <- matrix(rep(0, ncol(X)), nrow = ncol(X))
  #use the optim function to perform gradient descent
  costOpti <- optim(theta, fn = cost, gr = grad, X=X, y=y)
  #return coefficients
  return(costOpti$par)
}

logisticProb <- function(theta, X){
  X <- na.omit(X)
  #add bias term and convert to matrix
  X <- mutate(X, bias =1)
  X <- as.matrix(X[,c(ncol(X), 1:(ncol(X)-1))])
  return(sigmoid(X%*%theta))
}

logisticPred <- function(prob){
  return(round(prob, 0))
}

# training
theta <- logisticReg(X, y)
prob <- logisticProb(theta, X)
pred <- logisticPred(prob)

# generate a grid for decision boundary, this is the test set
grid <- expand.grid(seq(0, 3, length.out = 100), seq(0, 3, length.out = 100))
# predict the probability
probZ <- logisticProb(theta, grid)
# predict the label
Z <- logisticPred(probZ)
gridPred = cbind(grid, Z)

# decision boundary visualization
ggplot() + geom_point(data = data, aes(x=x1, y=x2, color = as.character(label)), size = 2, show.l
  geom_tile(data = gridPred, aes(x = grid[, 1],y = grid[, 2], fill=as.character(Z)), alpha = 0.3, s
  ylim(0, 3) +
  ggtitle('Decision Boundary for Logistic Regression') +
  coord_fixed(ratio = 1) +
  theme_bw(base_size = 12)

```

Now see example with Caret R machine learning library.

```
#source http://www.sthda.com/english/articles/36-classification-methods-essentials/151-logistic-reg

library(tidyverse)
library(caret)
theme_set(theme_bw())

# Load the data and remove NAs
data("PimaIndiansDiabetes2", package = "mlbench")
PimaIndiansDiabetes2 <- na.omit(PimaIndiansDiabetes2)
# Inspect the data
sample_n(PimaIndiansDiabetes2, 3)
# Split the data into training and test set
set.seed(123)
training.samples <- PimaIndiansDiabetes2$diabetes %>%
  createDataPartition(p = 0.8, list = FALSE)
train.data <- PimaIndiansDiabetes2[training.samples, ]
test.data <- PimaIndiansDiabetes2[-training.samples, ]

# Fit the model
model <- glm( diabetes ~., data = train.data, family = binomial)
# Summarize the model
summary(model)
# Make predictions
probabilities <- model %>% predict(test.data, type = "response")
predicted.classes <- ifelse(probabilities > 0.5, "pos", "neg")
# Model accuracy
head(probabilities)
head(predicted.classes)

mean(predicted.classes == test.data$diabetes) #Proportion of correctly classified observations:

#Logistic regression is limited to only two-class classification problems.
# There is an extension, called multinomial logistic regression,
#for multiclass classification problem (Chapter @ref(multinomial-logistic-regression)).
```

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.

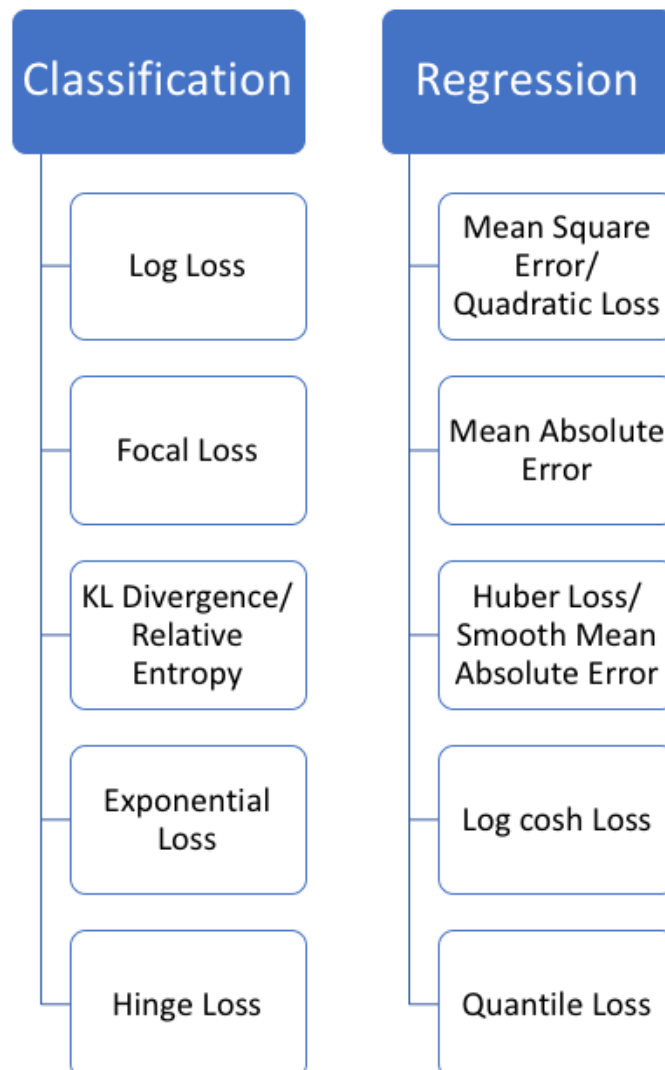


Figure: Losses in classification and regression. Adapted from "5 Regression Loss Functions All Machine Learners Should Know" by P. Grover, 2018. Retrieved from <https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0>.

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).

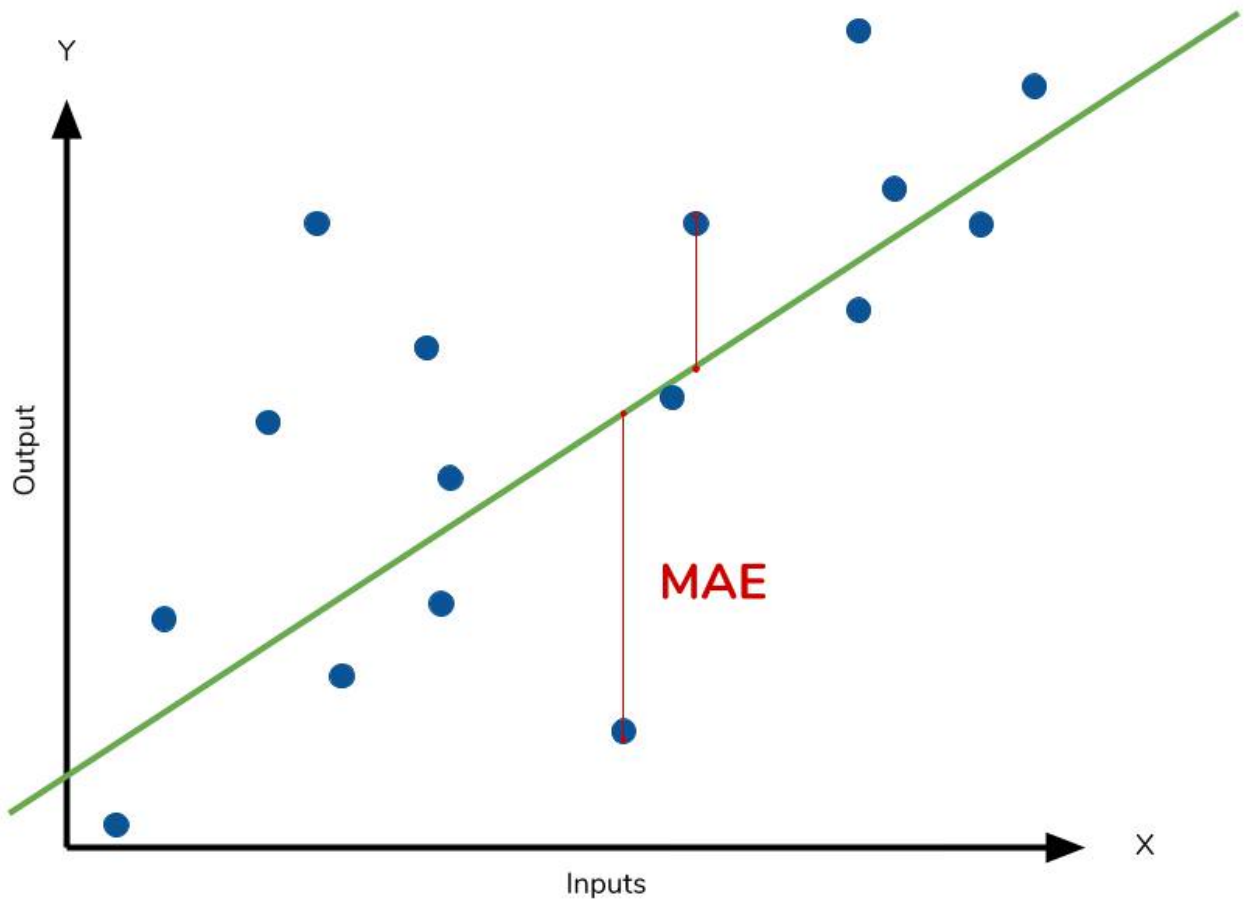


Figure: A linear regression model fitting into data. Adapted from "Tutorial: Understanding Regression Error Metrics in Python" by Pascual, n.d. Retrieved from <https://www.dataquest.io/blog/understanding-regression-error-metrics/>.

MAE function is

$$MAE = \frac{1}{n} \sum \left| y - \hat{y} \right|$$

Divide by the total number of data points

Actual output value

Predicted output value

Sum of

The absolute value of the residual

MSE function is

$$MSE = \frac{1}{n} \sum \left(y - \hat{y} \right)^2$$

The square of the difference between actual and predicted

MAPE function is

$$MAPE = \frac{100\%}{n} \sum \left| \frac{y - \hat{y}}{y} \right|$$

Multiplying by 100% converts to percentage

The residual

Each residual is scaled against the actual value

RMSE function is

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

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.

```
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_')
log_loss = loss(h, y_) # case of classification problem (assume class one 1s)
print(log_loss)
```

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

Error or loss functions



Explore the Jupyter notebook and try using it with different data sets.

This is a Jupyter notebook that demonstrates several loss functions using Python code and data. You can try different data sets or generate different forms of data to see additional trends in the calculation of loss and compare them with existing data and graphs.

Source: Jupyter nbviewer. (n.d). Regression loss. Retrieved from https://nbviewer.jupyter.org/github/groverpr/Machine-Learning/blob/master/notebooks/05_Loss_Functions.ipynb

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.

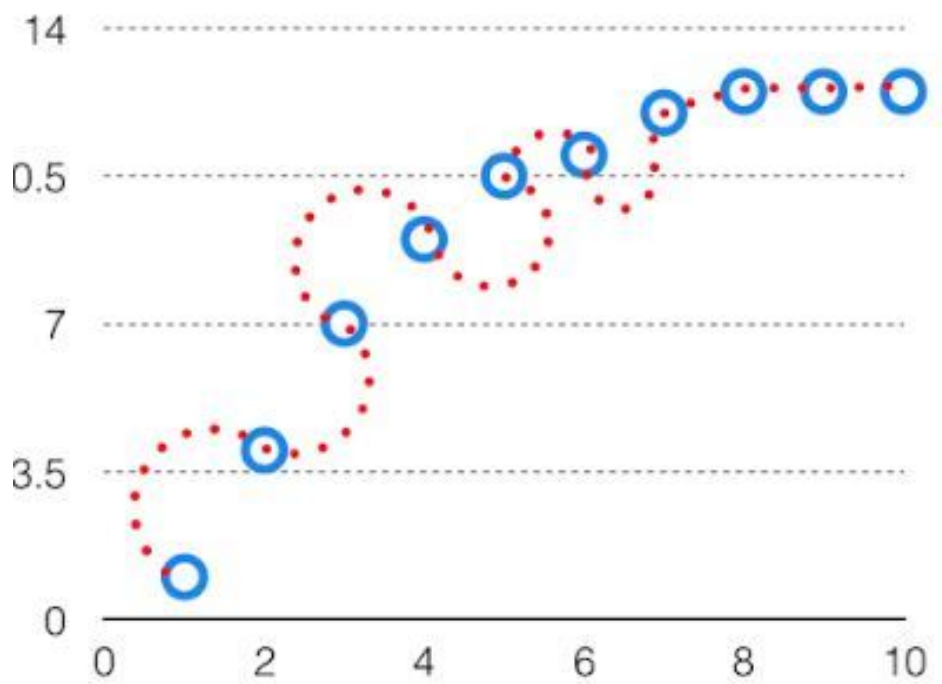


Figure: Overfitting of data points. Adapted from "Bias-Variance Trade off – Machine Learning" by Geeksforgeeks, n.d. Retrieved from <https://www.geeksforgeeks.org/ml-bias-variance-trade-off/?ref=leftbar-rightbar>.



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.

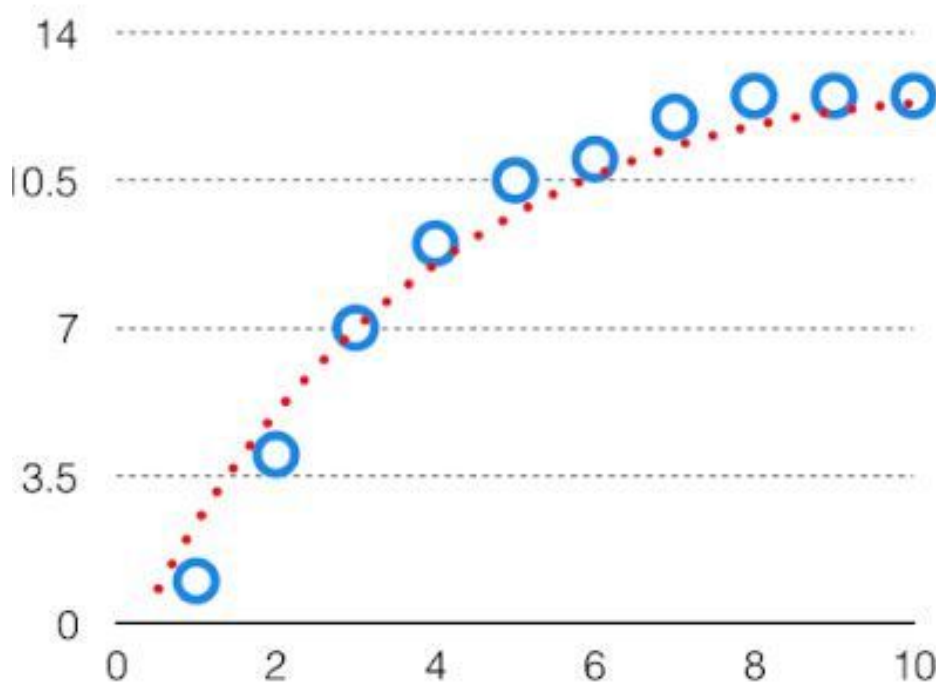




Figure: Bias Variance trade-off. Adapted from "Bias-Variance Trade off – Machine Learning" by Geeksforgeeks, n.d. Retrieved from <https://www.geeksforgeeks.org/ml-bias-variance-trade-off/?ref=leftbar-rightbar>.

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.

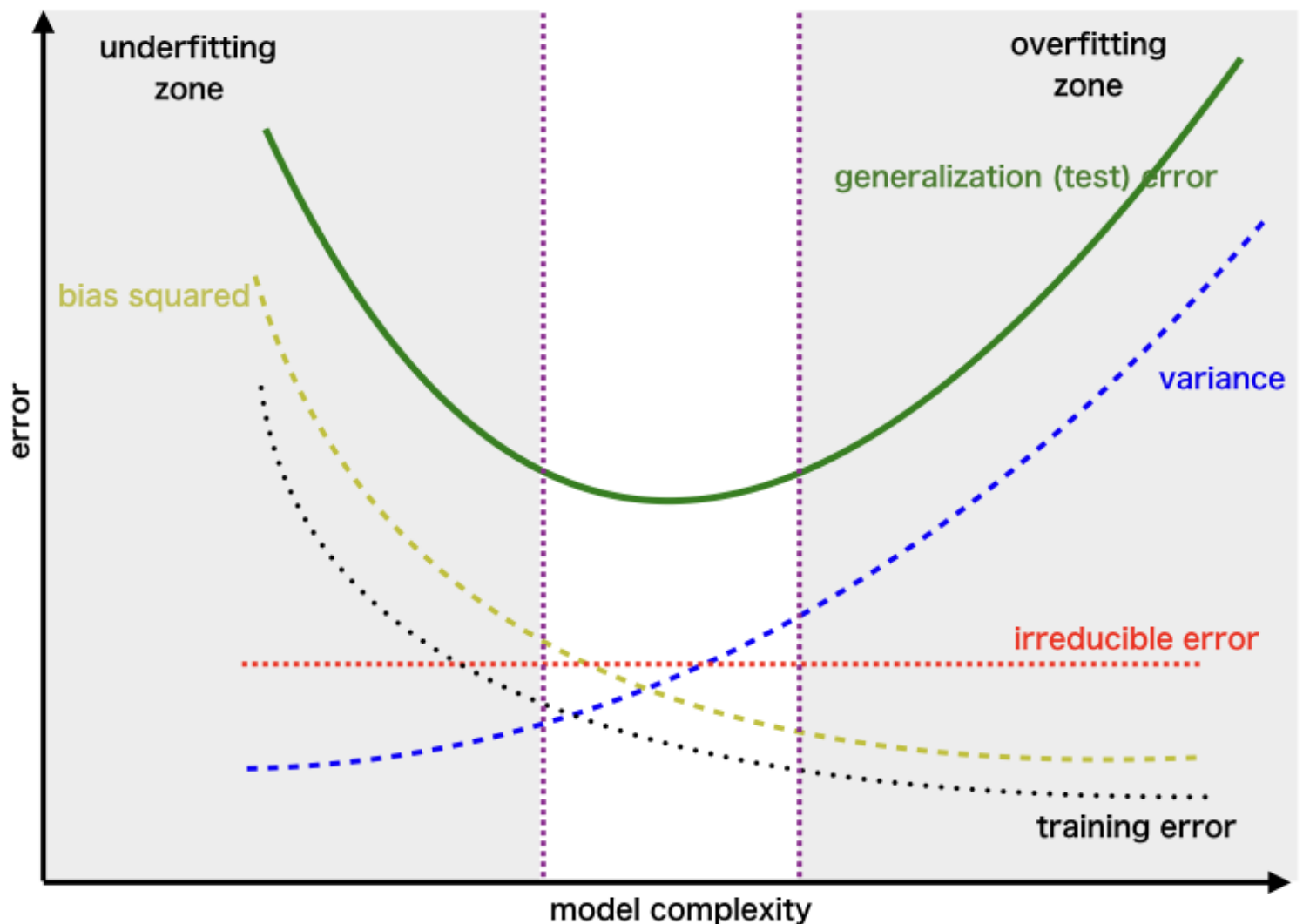


Figure: Model complexity vs error. Adapted from "Bias-Variance Trade off – Machine Learning" by Geeksforgeeks, n.d. Retrieved from <https://www.geeksforgeeks.org/ml-bias-variance-trade-off/?ref=leftbar-rightbar>.

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.

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.

N-fold is also known as 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.

The diagram shows how you can use a 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 deviations in your results. This will also show statistical validity to indicate a measure of uncertainty in your model predictions.

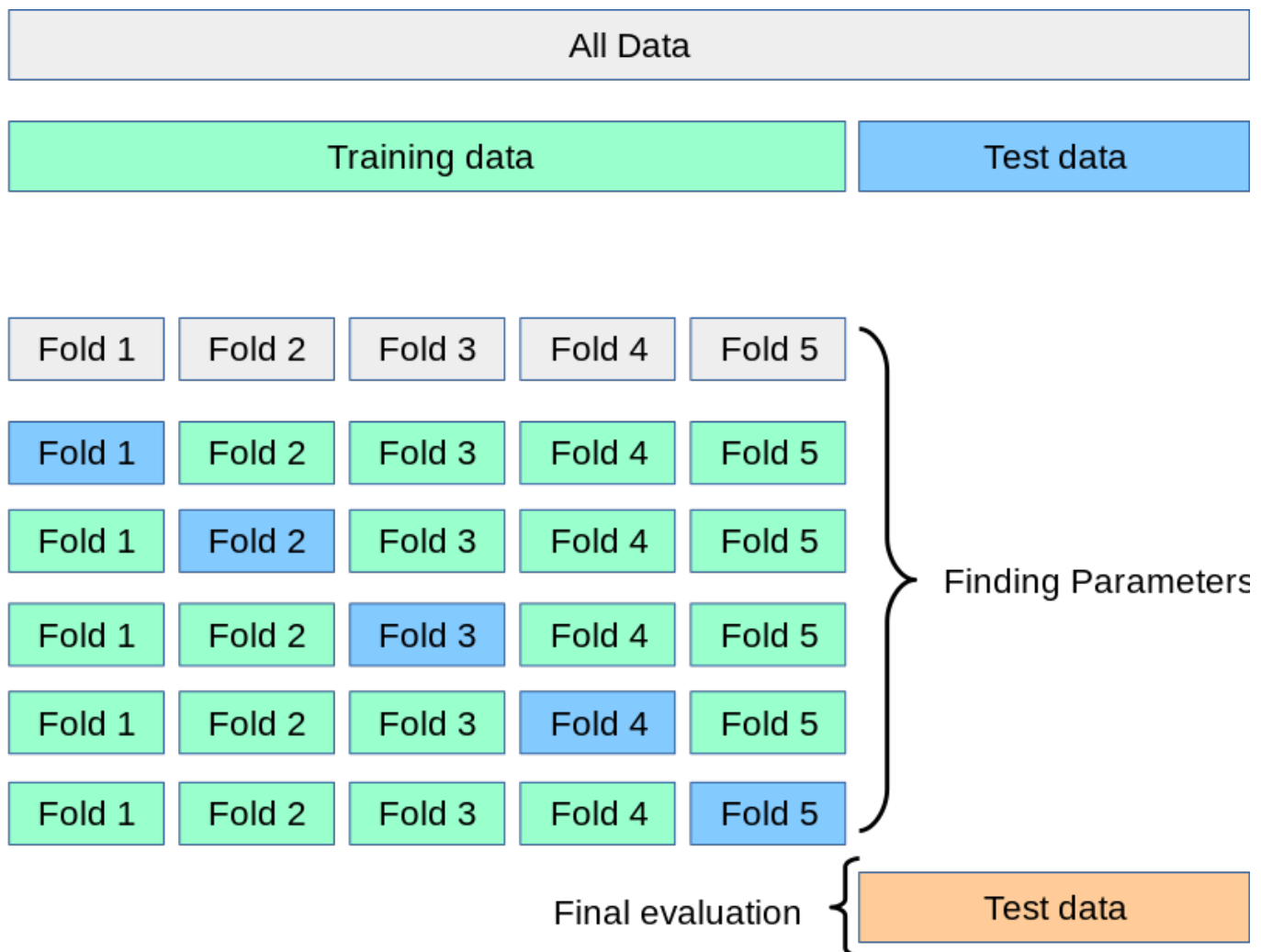


Figure N-cross validation. Adapted from "Cross-validation: evaluating estimator performance" by scikit-learn, n.d. Retrieved from https://scikit-learn.org/stable/modules/cross_validation.html.

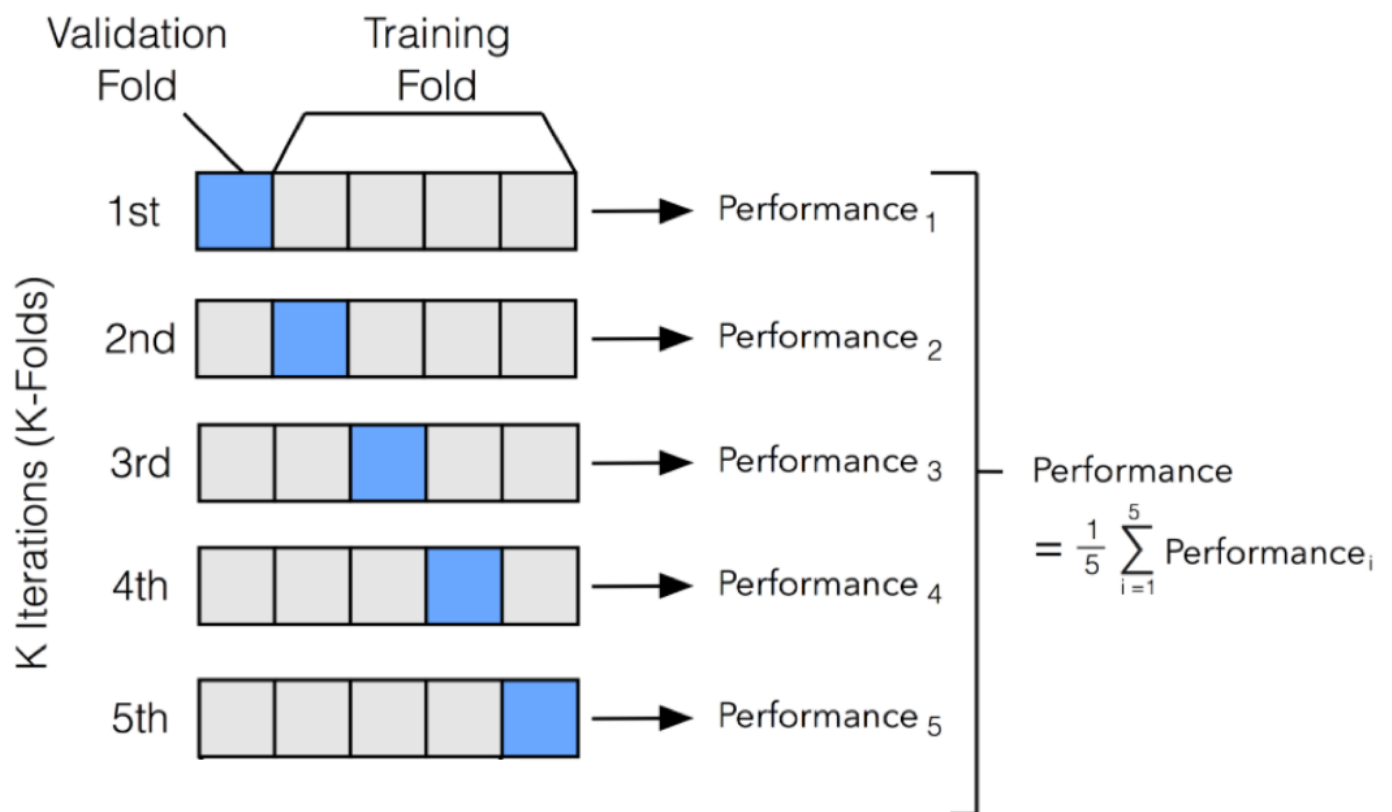




Figure: N-cross validation approach. Adapted from "Model selection" by Github, n.d. Retrieved from http://ethen8181.github.io/machine-learning/model_selection/model_selection.html.

```
from sklearn import datasets, linear_model

from sklearn.model_selection import cross_validate

from sklearn.model_selection import KFold
from sklearn import metrics

#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html
# simulate splitting a dataset into 5 folds
diabetes = datasets.load_diabetes()
X = diabetes.data[:150]

y = diabetes.target[:150]
lasso = linear_model.Lasso()
#Single metric evaluation using cross_validate

cv_results = cross_validate(lasso, X, y, cv=5)
res = sorted(cv_results.keys())
print(res)
ind_res = cv_results['test_score']
print(ind_res, ' CV res')

scores = cross_validate(lasso, X, y, cv=5, scoring=('r2', 'neg_mean_squared_error'), return_train_s
print(scores['test_neg_mean_squared_error'])
print(scores['train_r2'])
```



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

```
#Source: STHDA. (2018). Cross-Validation Essentials in R. http://www.sthda.com/english/articles/38-

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
```

```
model <- train(Fertility ~., data = swiss, method = "lm",  
              trControl = train.control)  
# Summarize the results  
print(model)
```

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.

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.

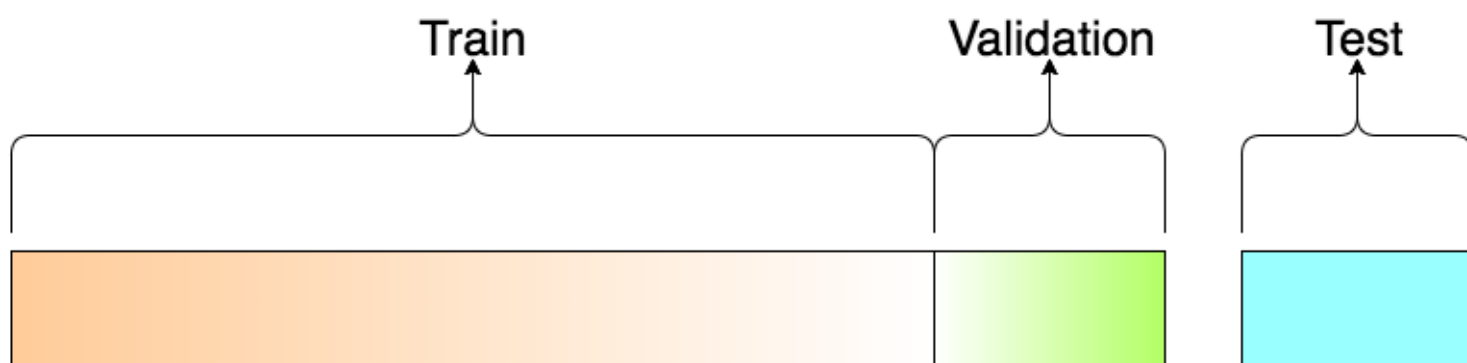


Figure: About Train. Adapted from "Validation and Test Sets in Machine Learning" by Towards data science, 2017. Retrieved from <https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7>.

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.

```
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
print(model)
```


Confusion matrix



Read the following content on confusion matrix and complete the associated activities.

Another technique to analyse and evaluate the performance of a model is a confusion matrix.

The confusion matrix is a way to evaluate the performance of classification methods. It gives a better indication of the performance of the model rather than just the classification performance, which does not indicate how well the model performs for the different classes.

The confusion matrix shows the ways in which our classification model is confused when it makes predictions. This gives us insight not only into the errors but the types of errors made by the model.

The confusion matrix table features two dimensions ("actual" and "predicted"), and identical sets of "classes" in both dimensions. Below is an example of the confusion matrix that features the number of True Positives, True Negatives, False Positives, and False Negatives gathered from the model output when compared to the actual data.

		Actual class	
		Cat	Non-cat
Predicted class	Cat	5 True Positives	2 False Positives
	Non-cat	3 False Negatives	3 True Negatives

$$BM = TPR + TNR - 1$$

markedness (MK) or deltaP

$$MK = PPV + NPV - 1$$

Sources: Fawcett (2006),^[1] Powers (2011),^[2] Ting (2011),^[3] and CAWCR^[4] Chicco & Jurman (2020)^[5]. Tharwat (2018)^[6].

The final table of confusion would contain the average values for all classes combined.

Let us define an experiment from **P** positive instances and **N** negative instances for some condition. The four outcomes can be formulated in a 2x2 *confusion matrix*, as follows:

		True condition			
		Condition positive	Condition negative		
Predicted condition	Total population	Condition positive	Condition negative	Prevalence $= \frac{\sum \text{Condition positive}}{\sum \text{Total population}}$	Accuracy (ACC) = $\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$
	Predicted condition positive	True positive	False positive, Type I error	Positive predictive value (PPV), Precision = $\frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\sum \text{False positive}}{\sum \text{Predicted condition positive}}$
	Predicted condition negative	False negative, Type II error	True negative	False omission rate (FOR) = $\frac{\sum \text{False negative}}{\sum \text{Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\sum \text{True negative}}{\sum \text{Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power $= \frac{\sum \text{True positive}}{\sum \text{Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\sum \text{False positive}}{\sum \text{Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{TPR}{FPR}$	Diagnostic odds ratio (DOR) = $\frac{LR+}{LR-}$ $F_1 \text{ score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
		False negative rate (FNR), Miss rate $= \frac{\sum \text{False negative}}{\sum \text{Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) $= \frac{\sum \text{True negative}}{\sum \text{Condition negative}}$	Negative likelihood ratio (LR-) $= \frac{FNR}{TNR}$	

Figure: Confusion matrix. Adapted from "Confusion matrix" by Wikipedia, 2020. Retrieved from https://en.wikipedia.org/wiki/Confusion_matrix.

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Let's look at some examples below:

```
# assume 1 is positive
Actual Predicted Confusion-Matrix
1      1      TP
1      0      FN
0      0      TN
0      1      FP
```

```
# assume 0 is positive
Actual Predicted Confusion-Matrix
1      1      TN
1      0      FP
0      0      TP
0      1      FN
```

Below is a Python script for the confusion matrix with an example of actual and predicted values from a binary classification problem. We note that by default, scikit-learn takes 0 as positive.

Update: <https://edstem.org/au/courses/6212/discussion/626872>

```
# Python script for confusion matrix creation.
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

actual = [1, 1, 0, 1, 0, 0, 1, 0, 0, 0]
predicted = [1, 0, 0, 1, 0, 0, 1, 1, 1, 0]

#results = confusion_matrix(actual, predicted) # in this case, class 0 is taken as a postive class
results = confusion_matrix(actual, predicted, labels=[1,0])
# in this case, class 1 is taken as a postive class

print('Confusion Matrix :')
print(results)
print('Accuracy Score :',accuracy_score(actual, predicted) )
```

```
print('Report : ')\nprint(classification_report(actual, predicted) )
```



Practise confusion matrix in R. Run the code and examine the outputs.

In R, the Caret machine learning library provides the implementation for the confusion matrix and can be used as shown in the example below.

```
# example of a confusion matrix in R\nlibrary(caret)\n\nexpected <- factor(c(1, 1, 0, 1, 0, 0, 1, 0, 0, 0))\npredicted <- factor(c(1, 0, 0, 1, 0, 0, 1, 1, 1, 0))\nresults <- confusionMatrix(data=predicted, reference=expected)\n# in this case, class 0 is taken as a postive class\nprint(results)
```



You can find examples of confusion matrix in R [here](#).



Source: Geeksforgeeks. (n.d). Confusion Matrix in Machine Learning. Retrieved from <https://www.geeksforgeeks.org/confusion-matrix-machine-learning/>.





StatQuest. (2018, October 29). *Machine Learning Fundamentals: The Confusion Matrix* [online video]. Retrieved from <https://www.youtube.com/watch?v=Kdsp6soqA7o>.

Below, see code for multi-class problem:

```
#source: https://datascience.stackexchange.com/questions/40067/confusion-matrix-three-classes-pytho
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import numpy as np

y_true = ['Cat', 'Dog', 'Rabbit', 'Cat', 'Cat', 'Rabbit']
y_pred = ['Dog', 'Dog', 'Rabbit', 'Dog', 'Dog', 'Rabbit']

classes=['Cat', 'Dog', 'Rabbit']

cmat = confusion_matrix(y_true, y_pred, labels=['Cat', 'Dog', 'Rabbit'])

print(cmat, ' is confusion matrix')

#-----
```

```

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    import itertools
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

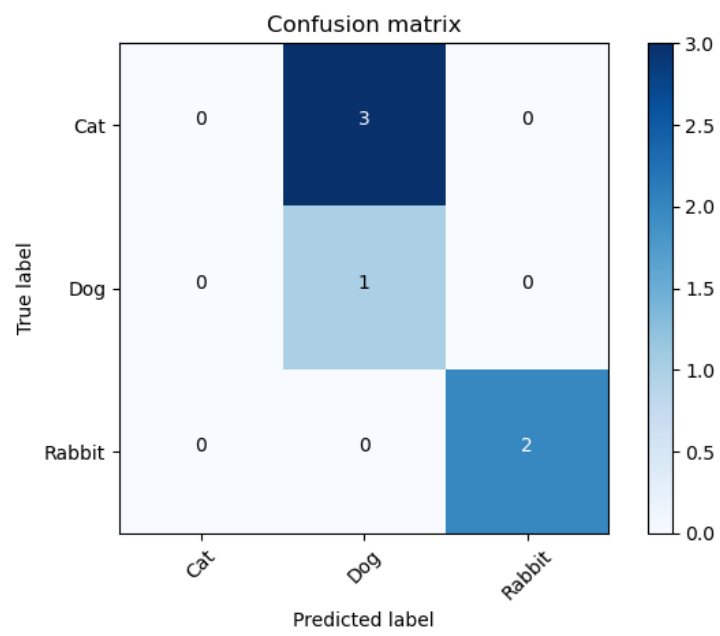
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    plt.savefig('cmat.png')

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cmat, classes=['Cat', 'Dog', 'Rabbit'],
                      title='Confusion matrix')

```

The above code gives output below:



Area Under The Curve (AUC) and Receiver Operating Characteristics (ROC) curve



Read the following content on Area Under The Curve (AUC) and Receiver Operating Characteristics (ROC) curve.

The **Area Under The Curve (AUC)** and **Receiver Operating Characteristics (ROC)** curve are used to visualise the performance of a multi-class classification problem. The AUC and ROC curve are also written as **Area Under the Receiver Operating Characteristics (AUROC)**..

The ROC is a probability curve while AUC represents the degree or measure of separability. It tells to what degree the model is capable of distinguishing between classes. Hence, the higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s (i.e., the respective outcomes or classes).

The ROC curve shows the performance of a classification model at all classification thresholds which plot two parameters based on True Positives (TP), True Negatives (TN), False Positives (FN), and False Negatives (FN).

True Positive Rate (TPR) : $TPR = TP / (TP + FN)$

False Positive Rate (FPR): $FPR = FP / (FP + TN)$

Consider the output of the model below that calculates probability (likelihood) of repaying a loan. Notice how the TPR and FPR changes when the threshold (t) changes.

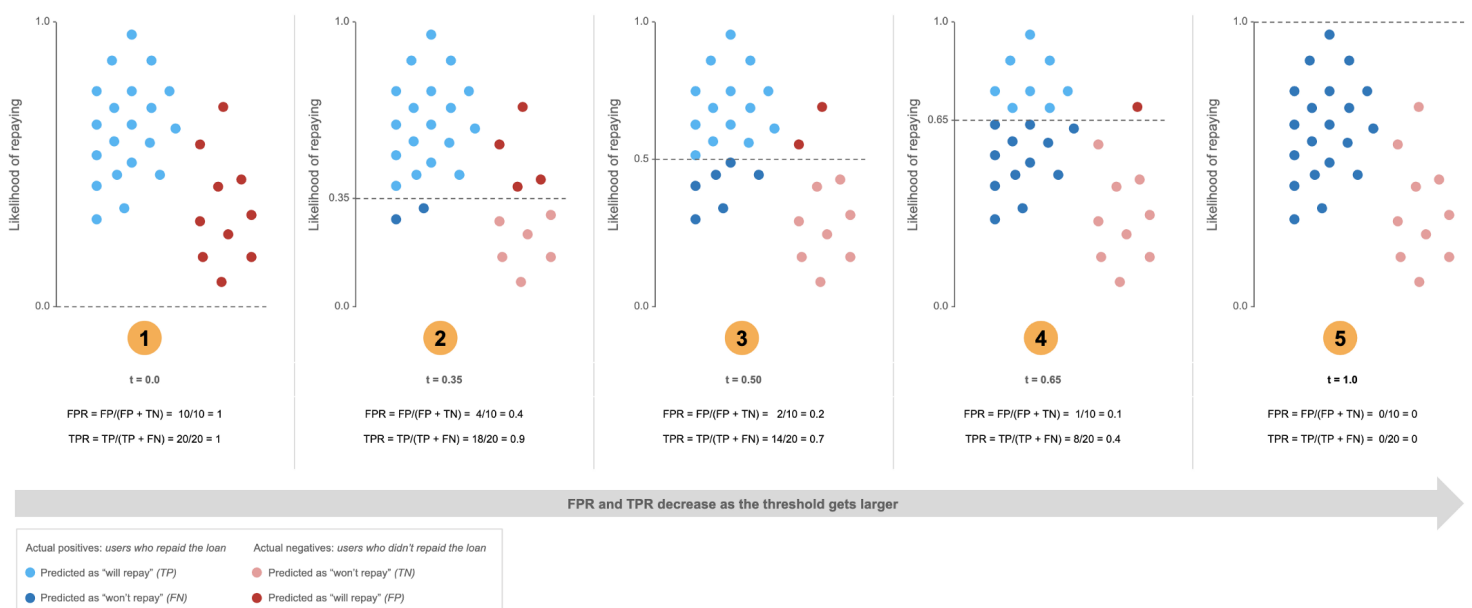


Figure Source: <https://towardsdatascience.com/understanding-the-roc-curve-in-three-visual-steps-795b1399481c>

The ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both FN and TP as shown below.

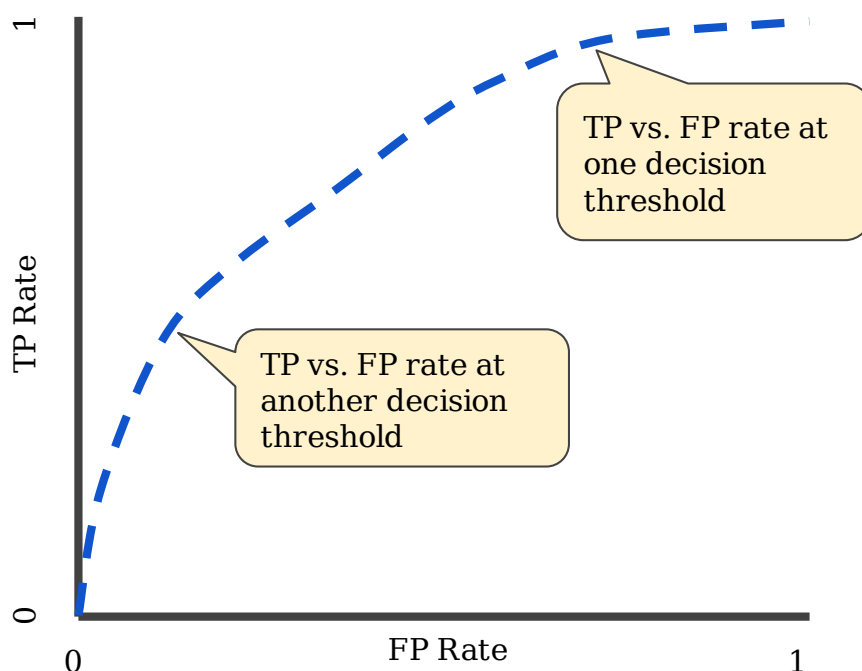


Figure: ROC curve. Adapted from "Classification: ROC Curve and AUC" by Google, n.d. Retrieved from <https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc>.

Now see how the ROC curve is constructed taking into account the values for TPR and FPR for different thresholds.

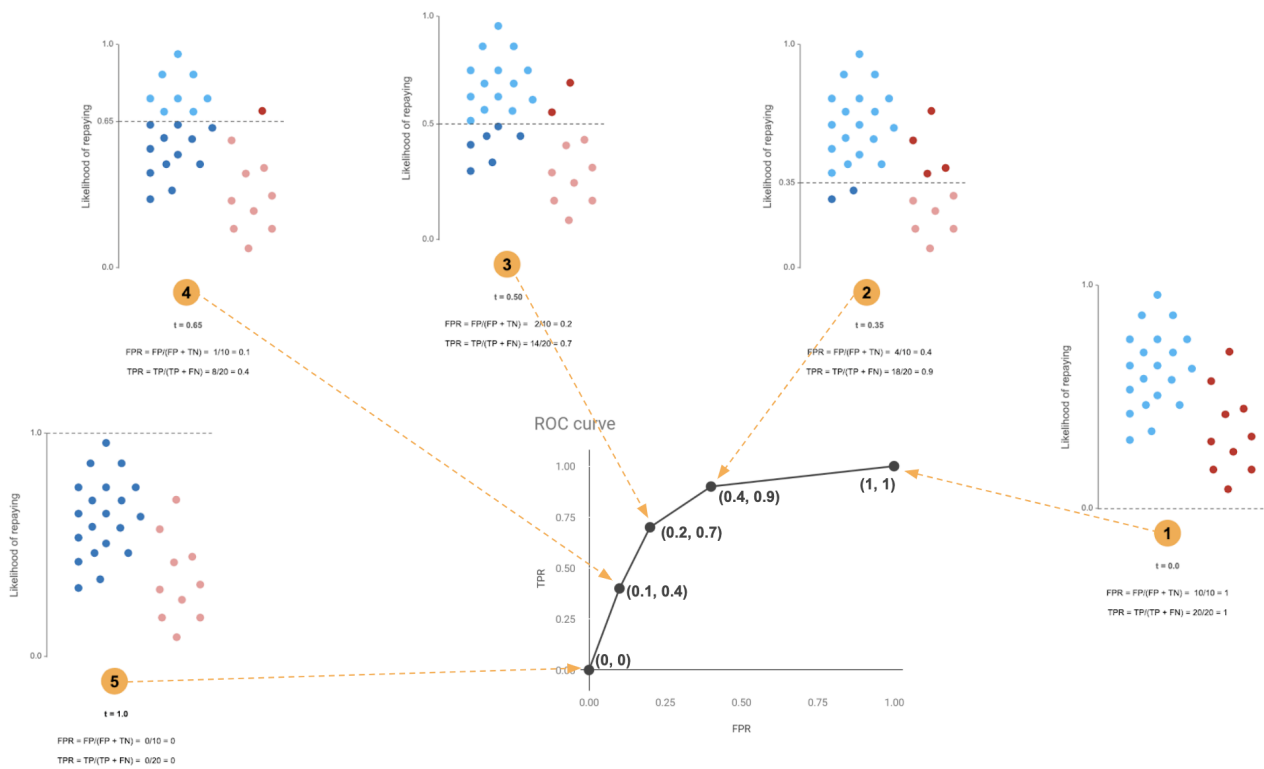


Figure Source: <https://towardsdatascience.com/understanding-the-roc-curve-in-three-visual-steps-795b1399481c>

Consider the following visualisation that shows how the threshold changes the true positive and false positive rate for the ROC.



Now consider the visualisation below that shows how AUC becomes given your model predictions.



"When the model can perfectly separate the two outcomes, the ROC curve forms a right angle and the AUC becomes 1." ~ Figure Source: <https://paulvanderlaken.com/2019/08/16/roc-auc-precision-and-recall-visually-explained/>



Statquest. (2019, July 11). *ROC and AUC, Clearly Explained!* [online video]. Retrieved from <https://www.youtube.com/watch?v=4jRBRDbJemM>.



Apply AUC and ROC using Python.

Understand and run the below codes on AUC and ROC curve using scikit-learn machine learning library with some examples to explain the concepts further.

```
# source: https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classificat
# roc curve and auc
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=2, random_state=1)
# split into train/test sets
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=2)
# generate a no skill prediction (majority class)
```

```

ns_probs = [0 for _ in range(len(testy))]
# fit a model
model = LogisticRegression(solver='lbfgs')
model.fit(trainX, trainy)
# predict probabilities
lr_probs = model.predict_proba(testX)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# calculate scores
ns_auc = roc_auc_score(testy, ns_probs)
lr_auc = roc_auc_score(testy, lr_probs)
# summarize scores
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('Logistic: ROC AUC=%.3f' % (lr_auc))
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(testy, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(testy, lr_probs)
# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
pyplot.legend()
# show the plot
pyplot.savefig('plot.png')

```

References

1. Fawcett, T. (2006). An introduction to ROC analysis. *Pattern recognition letters*, 27(8), 861-874. https://www.researchgate.net/profile/Tom_Fawcett/publication/222511520_Introduction_to_ROC_analysis/links/5ac7844ca6fdcc8bfc7fa47e/Introduction-to-ROC-analysis.pdf
2. Ferri, C., Hernández-Orallo, J., & Salido, M. A. (2003, September). Volume under the ROC surface for multi-class problems. In *European conference on machine learning* (pp. 108-120). Springer, Berlin, Heidelberg. https://link.springer.com/chapter/10.1007/978-3-540-39857-8_12
3. Landgrebe, T. C., & Duin, R. P. (2007). Approximating the multiclass ROC by pairwise analysis. *Pattern recognition letters*, 28(13), 1747-1758. <https://doi.org/10.1016/j.patrec.2007.05.001>
4. Multiclass ROC: <https://stackoverflow.com/questions/45332410/sklearn-roc-for-multiclass-classification>

Precision-Recall and F1 Score

Precision-recall curve

The field of information retrieval is based on finding documents based on queries to measure precision and recall which have been applied for machine learning for evaluating binary classification models.

The **precision** is a ratio of the number of true positives divided by the sum of the true positives and false positives. Precision describes how good a model is at predicting the positive class and is referred to as the positive predictive value.

The **recall** is the ratio of the number of true positives divided by the sum of the true positives and the false negatives, also known as sensitivity.

Precision-recall is useful in finding an imbalance in the observations between the two classes. Specifically, there are many examples of no event (class 0) and only a few examples of an event (class 1).



~ Figure Source: <https://paulvanderlaken.com/2019/08/16/roc-auc-precision-and-recall-visually-explained/>

Precision-Recall curve is very useful for class-imbalance problems, i.e when one class outcomes are significantly greater than the other: eg you have 90 % positive examples and 10 % negative examples in a given dataset. That creates a challenge for the model. Below the figure shows that both distributions start with 1000 outcomes. Later the blue distribution is then reduced to 50 and we can see that the curve changes shape drastically when compared to the ROC curve. We note that the AUC value mostly stays (around 0.9) when the change occurs.



~ Figure Source: <https://paulvanderlaken.com/2019/08/16/roc-auc-precision-and-recall-visually-explained/>



Apply the precision-recall curve using Python.

The below code shows the precision-recall curve and F1 score:

```
# source: https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classificat

# precision-recall curve and f1
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from matplotlib import pyplot

# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=2, random_state=1)
# split into train/test sets
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=2)
# fit a model
model = LogisticRegression(solver='lbfgs')
model.fit(trainX, trainy)
# predict probabilities
lr_probs = model.predict_proba(testX)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# predict class values
yhat = model.predict(testX)
lr_precision, lr_recall, _ = precision_recall_curve(testy, lr_probs)
lr_f1, lr_auc = f1_score(testy, yhat), auc(lr_recall, lr_precision)
# summarize scores
print('Logistic: f1=%.3f auc=%.3f' % (lr_f1, lr_auc))
# plot the precision-recall curves
no_skill = len(testy[testy==1]) / len(testy)
pyplot.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')
```

```

pyplot.plot(lr_recall, lr_precision, marker='.', label='Logistic')
# axis labels
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
# show the legend
pyplot.legend()
# show the plot
pyplot.savefig('plot_AUC.png')

```

Now we see ROC in R

```

#source: https://stackoverflow.com/questions/4903092/calculate-auc-in-r

true_Y = c(1,1,1,1,2,1,2,1,2,2)
probs = c(1,0.999,0.999,0.973,0.568,0.421,0.382,0.377,0.146,0.11)

getROC_AUC = function(probs, true_Y){
  probsSort = sort(probs, decreasing = TRUE, index.return = TRUE)
  val = unlist(probsSort$x)
  idx = unlist(probsSort$ix)

  roc_y = true_Y[idx];
  stack_x = cumsum(roc_y == 2)/sum(roc_y == 2)
  stack_y = cumsum(roc_y == 1)/sum(roc_y == 1)

  auc = sum((stack_x[2:length(roc_y)]-stack_x[1:length(roc_y)-1])*stack_y[2:length(roc_y)])
  return(list(stack_x=stack_x, stack_y=stack_y, auc=auc))
}

aList = getROC_AUC(probs, true_Y)

stack_x = unlist(aList$stack_x)
stack_y = unlist(aList$stack_y)
auc = unlist(aList$auc)

plot(stack_x, stack_y, type = "l", col = "blue", xlab = "False Positive Rate", ylab = "True Positive Rate")
axis(1, seq(0.0,1.0,0.1))
axis(2, seq(0.0,1.0,0.1))
abline(h=seq(0.0,1.0,0.1), v=seq(0.0,1.0,0.1), col="gray", lty=3)
legend(0.7, 0.3, sprintf("%3.3f",auc), lty=c(1,1), lwd=c(2.5,2.5), col="blue", title = "AUC")

```

F1 score

The F1 score is a score to give further information, which is calculated by **2*((precision*recall)/(precision+recall))** and also called the F Score or the F Measure.

F1 score conveys the balance between precision and recall.



Apply the class imbalance problem using Python.

Run the below code and you will get AUC curve for the class imbalance problem:

```
# roc curve and auc on an imbalanced dataset
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=2, weights=[0.99,0.01], random_state=1)
# split into train/test sets
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=2)
# generate a no skill prediction (majority class)
ns_probs = [0 for _ in range(len(testy))]
# fit a model
model = LogisticRegression(solver='lbfgs')
model.fit(trainX, trainy)
# predict probabilities
lr_probs = model.predict_proba(testX)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# calculate scores
ns_auc = roc_auc_score(testy, ns_probs)
lr_auc = roc_auc_score(testy, lr_probs)
# summarize scores
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('Logistic: ROC AUC=%.3f' % (lr_auc))
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(testy, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(testy, lr_probs)
# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
pyplot.legend()
# show the plot
pyplot.savefig('imbalance_AUC.png')
```

References

1. Davis, J., & Goadrich, M. (2006, June). The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning* (pp. 233-240).https://dl.acm.org/doi/pdf/10.1145/1143844.1143874?casa_token=foicvb37FmlAAAAA:YMuoxFwhutpqUr1V_tXBaGJ8GpfN8529IXM72QM_K2wiC4ojOKH2XpF0-rl3ezNVFP_-14mQjhSugmg
2. Flach, P. A., & Kull, M. (2015, December). Precision-Recall-Gain Curves: PR Analysis Done Right. In *NIPS* (Vol. 15). http://people.cs.bris.ac.uk/~flach/PRGcurves/PRcurves_ISL.pdf
3. Powers, D. M. (2020). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*.
<https://arxiv.org/pdf/2010.16061.pdf>

1.

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 overfitting.

In the case of linear and logistic models, the major forms are by adding a type of penalty in the cost function with a 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>

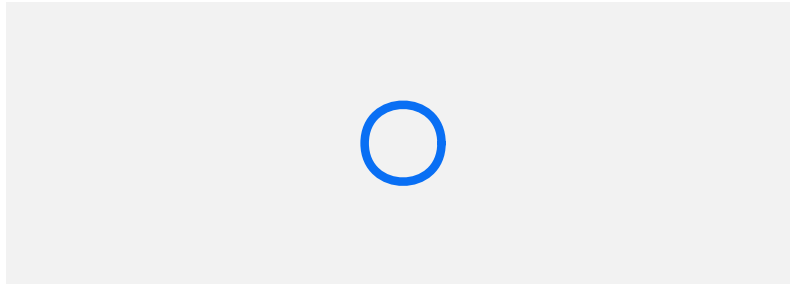
Lasso (L1) regularization: consider the following penalty added to the cost function:



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



Hence our new gradients would be



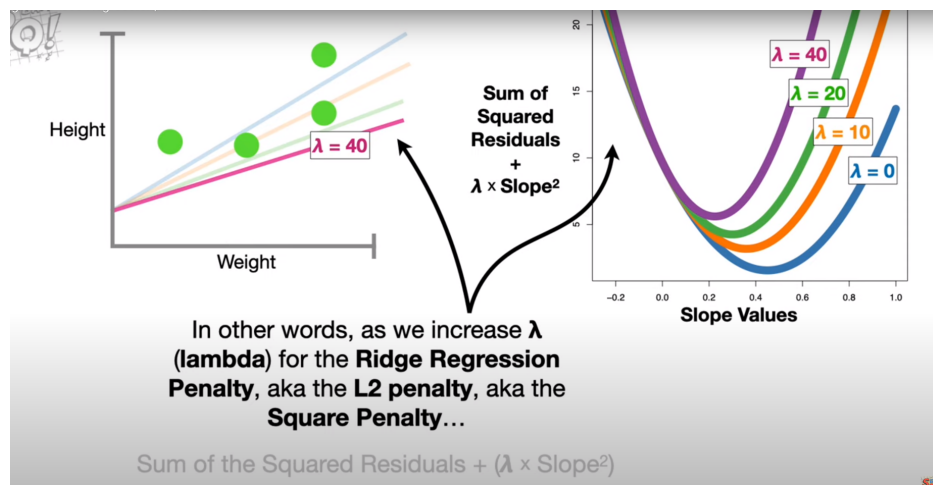
Ridge (L2) regularisation:

Consider the following penalty added to the cost function



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_gTAI8c&t=402s

```
#Source: https://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_l1_l2_sparsity.ht
# Authors: Alexandre Gramfort <alexandre.gramfort@inria.fr>
#          Mathieu Blondel <mathieu@mbondel.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)

X = StandardScaler().fit_transform(X)

# classify small against large digits
y = (y > 4).astype(np.int)

l1_ratio = 0.5 # L1 weight in the Elastic-Net regularization

fig, axes = plt.subplots(3, 3)

# Set regularization parameter
for i, (C, axes_row) in enumerate(zip((1, 0.1, 0.01), axes)):
    # turn down tolerance for short training time
    clf_l1_LR = LogisticRegression(C=C, penalty='l1', tol=0.01, solver='saga')
    clf_l2_LR = LogisticRegression(C=C, penalty='l2', tol=0.01, solver='saga')
    clf_en_LR = LogisticRegression(C=C, penalty='elasticnet', solver='saga',
                                  l1_ratio=l1_ratio, tol=0.01)

    #documentation for solver choices
    #https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
    clf_l1_LR.fit(X, y)
    clf_l2_LR.fit(X, y)
    clf_en_LR.fit(X, y)
```

```

coef_l1_LR = clf_l1_LR.coef_.ravel()
coef_l2_LR = clf_l2_LR.coef_.ravel()
coef_en_LR = clf_en_LR.coef_.ravel()

# coef_l1_LR contains zeros due to the
# L1 sparsity inducing norm

sparsity_l1_LR = np.mean(coef_l1_LR == 0) * 100
sparsity_l2_LR = np.mean(coef_l2_LR == 0) * 100
sparsity_en_LR = np.mean(coef_en_LR == 0) * 100

print("C=%.2f" % C)
print("{:<40} {:.2f}%".format("Sparsity with L1 penalty:", sparsity_l1_LR))
print("{:<40} {:.2f}%".format("Sparsity with Elastic-Net penalty:",
                               sparsity_en_LR))
print("{:<40} {:.2f}%".format("Sparsity with L2 penalty:", sparsity_l2_LR))
print("{:<40} {:.2f}%".format("Score with L1 penalty:",
                               clf_l1_LR.score(X, y)))
print("{:<40} {:.2f}%".format("Score with Elastic-Net penalty:",
                               clf_en_LR.score(X, y)))
print("{:<40} {:.2f}%".format("Score with L2 penalty:",
                               clf_l2_LR.score(X, y)))

if i == 0:
    axes_row[0].set_title("L1 penalty")
    axes_row[1].set_title("Elastic-Net\nl1_ratio = %s" % l1_ratio)
    axes_row[2].set_title("L2 penalty")

for ax, coefs in zip(axes_row, [coef_l1_LR, coef_en_LR, coef_l2_LR]):
    ax.imshow(np.abs(coefs.reshape(8, 8)), interpolation='nearest',
              cmap='binary', vmax=1, vmin=0)
    ax.set_xticks(())
    ax.set_yticks(())

axes_row[0].set_ylabel('C = %s' % C)

```



R code example: <https://www.r-bloggers.com/2017/07/machine-learning-explained-regularization/>

Further reading:

1. Discussion for R with logistic regression: <https://www.machinelearning.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.umi.acs.umd.edu/~hcorrada/PML/homeworks/HW04_solutions.pdf

Types of machine learning methods

Learning Problems

- *Supervised Learning: Logistic and linear regression, Backpropagation neural networks, Bayesian neural networks (Week 1 - 5)*
- *Unsupervised Learning : Clustering, PCA, Autoencoders*
- Reinforcement Learning: Used in computer game applications and robotics where data (action) is generated during the learning process.

Learning Techniques

- *Ensemble Learning: Covered in the course where multiple models are used. Random forest, Adaboost and XGboost are examples.*
- Multi-Task Learning: Related tasks that help each other in learning; eg. driving a tractor and driver a car has some overlap in foundational knowledge.
- Transfer Learning
- Active Learning
- Online Learning

Hybrid Learning Problems

- Semi-Supervised Learning: Combination of supervised and unsupervised learning.
- Self-Supervised Learning
- Multi-Instance Learning

Statistical Inference

- Inductive Learning
- Deductive Inference
- Transductive Learning

Source: <https://machinelearningmastery.com/types-of-learning-in-machine-learning/>

Exercise 2.1

Develop a **logistic regression model** for binary classification in **R or Python** and do the following:

1. Create 60/40 train test split and report training and test performance in terms of classification performance.
2. Report AUC and ROC and Precision-Recall curve, F1 Score.
3. Try L1/L2 and elastic net regularisation and compare results.
4. Carry out 10 independent experiment runs for each case and report the mean and std of the results in Part 2.
5. Carry out 10-fold cross-validation on the original dataset and report the results required in Part 2.

You can use Caret with R or any other library. You can use scikit-learn with Python or the code from scratch.

Datasets:

- Pima-Indian diabetes dataset: <https://www.kaggle.com/kumargh/pimaindiansdiabetescsv>
- Survival dataset: <https://archive.ics.uci.edu/ml/datasets/Haberman%27s+Survival>
- Breast Cancer dataset: [https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+\(Diagnostic\)](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic))

Exercise 2.1 Solution

Develop a **logistic regression model** for binary classification in **R or Python** and do the following:

1. Create 60/40 train test split and report training and test performance in terms of classification performance.
2. Report AUC and ROC and Precision-Recall curve, F1 Score.
3. Try L1/L2 and elastic net regularisation and compare results.
4. Carry out 10 independent experiment runs for each case and report the mean and std of the results in Part 2.
5. Carry out 10-fold cross-validation on the original dataset and report the results required in Part 2.

You can use Caret with R or any other library. You can use scikit-learn with Python or the code from scratch.

Datasets:

1. Pima-Indian diabetes dataset: <https://www.kaggle.com/kumargh/pimaindiansdiabetescsv>
2. Survival dataset: <https://archive.ics.uci.edu/ml/datasets/Haberman%27s+Survival>
3. Breast Cancer dataset: [https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+\(Diagnostic\)](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic))

Exercise 2.1 Solutions in R

```
#
#Source: Vasiliki Vamvaka https://github.com/vasilikivamv/MATH5836/blob/master/Week%201/IRIS_week1_

data("iris")
attach(iris)

##### Part 1 #####

## Linear regression between column 1 & 3
fit.13 <- lm(Sepal.Length ~ Petal.Length, data=iris)
summary(fit.13)    # R-squared:    0.76

## Linear regression between column 1 & 2
fit.12 <- lm(Sepal.Length ~ Sepal.Width, data=iris)
summary(fit.12)    # R-squared:    0.01382

## Linear regression between column 1 & 4
fit.14 <- lm(Sepal.Length ~ Petal.Width, data=iris)
summary(fit.14)    # R-squared:    0.669

##### Part 2 #####

#1.  transform to a binary problem with classes 1 and 2 only

binary.iris <- iris[!(Species=="virginica"),]
binary.iris$Species <- ifelse(binary.iris$Species=="setosa",1,2)

#Splitting the data set to training and testing sets

set.seed(123)
training.samples <- sample(1:nrow(binary.iris), nrow(binary.iris)*0.6, replace = F)

binary_iris_train <- binary.iris[training.samples, ]
binary_iris_test <- binary.iris[-training.samples, ]

#### (i) Without normalization ####

# fixing the data as matrices
x <- as.matrix(binary_iris_train[,1:4])
X <- cbind(rep(1,nrow(x)), x)
y <- as.matrix(binary_iris_train$Species)
```

```

m <- length(y)
theta<-rep(0,5)

# cost function

cost<-function(X, y, theta){
  MSE <- sum((X%*%theta- y)^2)/(2*m)
  return(MSE)
}

# gradient decent
graddec<-function(X, y, theta, alpha, iters){
  MSE_store <- rep(0, iters)
  for(i in 1:iters){

    theta <- theta - alpha*(1/m)*(t(X)%*%(X%*%theta - y))

    MSE_store[i] <- cost(X, y, theta)
  }

  results<-list(theta, MSE_store)
  return(results)
}

alpha <- .001
iters <- 1000
results <- graddec(X, y, theta, alpha, iters)
theta <- results[[1]]
cost_hist <- results[[2]]
print(theta)

#           [,1]
#           0.021093121
# Sepal.Length 0.136962483
# Sepal.Width  0.000115254
# Petal.Length 0.225650930
# Petal.Width  0.086051863

# plot of the cost
plot(1:iters, cost_hist, type = 'l')

#### (ii) With normalized data ####

#normalization function
norm.function <- function(feature) {
  (feature - min(feature)) / (max(feature) - min(feature))
}

iris.normal <- as.data.frame (lapply(binary_iris_train[1:4], norm.function ))
iris.normal$Species <- binary_iris_train$Species
head(iris.normal)

```

```

x.norm <- as.matrix(iris.normal[,1:4])
X.norm <- cbind(rep(1,nrow(x.norm)), x.norm)
y.norm <- as.matrix(iris.normal$Species)
m.norm <- length(y.norm)
theta.norm <- rep(0,5)

m <- m.norm
alpha <- .001
iters <- 1000

# cost function

cost.norm<-function(X.norm, y.norm, theta.norm){
  MSE.norm <- sum((X.norm%*%theta.norm- y.norm)^2)/(2*m.norm)
  return(MSE.norm)
}

# gradient decent
graddec.norm<-function(X.norm, y.norm, theta.norm, alpha, iters){
  MSE_store.norm <- rep(0, iters)
  for(i in 1:iters){

    theta.norm <- theta.norm - alpha*(1/m.norm)*(t(X.norm)%*%(X.norm%*%theta.norm - y.norm))

    MSE_store.norm[i] <- cost.norm(X.norm, y.norm, theta.norm)
  }

  results.norm<-list(theta.norm, MSE_store.norm)
  return(results.norm)
}

results.norm <- graddec.norm(X.norm, y.norm, theta.norm, alpha, iters)
theta.norm <- results.norm[[1]]
cost_hist.norm <- results.norm[[2]]
print(theta.norm)
#
#           [,1]
#           0.6744380
# Sepal.Length 0.3435897
# Sepal.Width  0.2346614
# Petal.Length 0.4148836
# Petal.Width  0.3803888

# plot of the cost
plot(1:iters, cost_hist.norm, type = 'l')

```

Extra: Logistic Regression with Maximum likelihood

This lesson is not part of course - for information purpose.

We need to choose weight \mathbf{w} coefficients given input \mathbf{x} and outcomes y for N samples that maximizes likelihood given below

$$\prod_{i=1}^N P(y_i | \mathbf{x}_i, \mathbf{w})$$

Log-likelihood for binary classification

$$l(\mathbf{w}) = \ln \prod_{i=1}^N P(y_i | \mathbf{x}_i, \mathbf{w})$$

Suppose that the sample is made up of independent observations, the logarithm transforms a product of densities into a sum which is very convenient since 1. the asymptotic properties of sums are easier to analyze, 2. products are not numerically stable. Source:

<https://gogul.dev/software/ml/logistic-regression-from-scratch>

Gradient derivation is given here:

<https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/pdfs/40%20LogisticRegression.pdf>

Further info: <https://courses.cs.washington.edu/courses/cse446/13sp/slides/logistic-regression-gradient.pdf>

```
#source: https://gogul.dev/software/ml/logistic-regression-from-scratch
```

```
from sklearn.datasets import load_breast_cancer
import numpy as np
import matplotlib.pyplot as plt
```

```
def sigmoid(score):
    return (1 / (1 + np.exp(-score)))
```

```
def predict_probability(features, weights):
    score = np.dot(features, weights)
    return sigmoid(score)
```

```

def feature_derivative(errors, feature):
    derivative = np.dot(np.transpose(errors), feature)
    return derivative

def compute_log_likelihood(features, label, weights):
    indicator = (label==+1)
    scores = np.dot(features, weights)
    ll = np.sum((np.transpose(np.array([indicator]))-1)*scores - np.log(1. + np.exp(-scores)))
    return ll

def logistic_regression(features, labels, lr, epochs):

    # add bias (intercept) with features matrix
    bias = np.ones((features.shape[0], 1))
    features = np.hstack((bias, features))

    # initialize the weight coefficients
    weights = np.zeros((features.shape[1], 1))

    logs = []

    # loop over epochs times
    for epoch in range(epochs):

        # predict probability for each row in the dataset
        predictions = predict_probability(features, weights)

        # calculate the indicator value
        indicators = (labels==+1)

        # calculate the errors
        errors = np.transpose(np.array([indicators])) - predictions

        # loop over each weight coefficient
        for j in range(len(weights)):

            # calculate the derivative of jth weight coefficient
            derivative = feature_derivative(errors, features[:,j])
            weights[j] += lr * derivative

        # compute the log-likelihood
        ll = compute_log_likelihood(features, labels, weights)
        logs.append(ll)

#MAIN

x = np.linspace(0, len(logs), len(logs))
fig = plt.figure()
plt.plot(x, logs)
fig.suptitle('Training the classifier (without L2)')
plt.xlabel('Epoch')
plt.ylabel('Log-likelihood')

```

```

fig.savefig('train_without_l2.jpg')

return weights

data = load_breast_cancer()

print(data.keys())
print("No.of.data points (rows) : {}".format(len(data.data)))
print("No.of.features (columns) : {}".format(len(data.feature_names)))
print("No.of.classes : {}".format(len(data.target_names)))
print("Class names : {}".format(list(data.target_names)))

import pandas as pd
df = pd.DataFrame(data.data)
print(df.dtypes)

from sklearn.model_selection import train_test_split
# split the dataset into training and testing
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, test_size=0.20, random_

print("X_train : " + str(X_train.shape))
print("y_train : " + str(y_train.shape))
print("X_test : " + str(X_test.shape))
print("y_test : " + str(y_test.shape))

# hyper-parameters
learning_rate = 1e-7
epochs = 500

# perform logistic regression
learned_weights = logistic_regression(X_train, y_train, learning_rate, epochs)

```

Extra: Logistic Regression from scratch

Now that you have learned how to apply a logistic regression (perceptron learning), let's learn how you can use it with gradient descent in Python without using any libraries, but rather, coding from scratch. This is a logistic regression implementation from scratch using basic python arrays (lists).

This is a linear and logistic regression implementation from scratch using basic python arrays (lists).

Code challenge is to convert this by using numpy arrays.



Source: <https://machinelearningmastery.com/implement-logistic-regression-stochastic-gradient-descent-scratch-python/>



Dataset: <https://www.kaggle.com/kumargh/pima-indians-diabetes-csv?select=pima-indians-diabetes.csv>