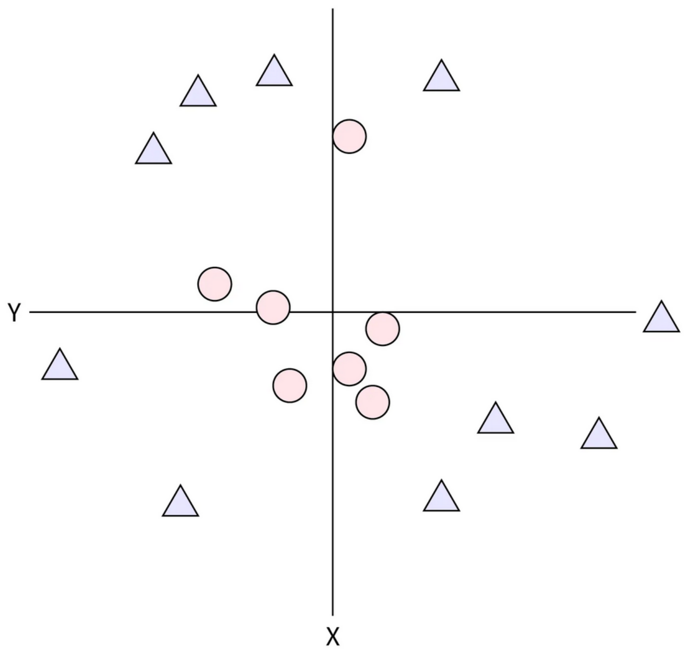
Linear Algebra and its Applications in Machine Learning

Varun Edachali, Adithi Samudrala, Kavin Aravindan

June 2023



Introduction And Objectives

Machine Learning is a technology of developing algorithms that enable computers to learn automatically from past data. It uses various algorithms for building mathematical models and making predictions using historical data.

Tools in Linear Algebra such as matrix operations, eigendecomposition, and more are extensively used in Machine Learning.

This report investigates three key concepts in Machine Learning - Linear Regression, Principal Component Analysis and Support Vector Machines - talks about their use and implementation, and highlights the importance of Linear Algebra in these topics.

The report, as a whole, is essentially divided into three separate sections, each of which delve into the topics mentioned before. They are listed below.

Contents

Linear Regression

Varun Edachali

1 Objectives

To explain Linear Regression from scratch, going over some basic methods to simulate Linear Regression models, and to run it on a real data set.

2 Introduction

Machine Learning models use algorithms to learn from data just like any living being learns from their experiences. They can be divided into two categories based on the learning data and the task to be performed.

1. **Supervised Learning Models**: contain past data with input as well as the required output (also known as labelled data).
    - Regression: the output to be predicted is continuous in nature.
    - Classification: The output variable to be predicted is categorical in nature, such as true or false.
2. **Unsupervised Learning Models**: contain no predefined labels assigned to the past data, only input data is provided and not any output data. It is usually used to find patterns in some data-set.
    - Clustering: Groups or clusters of data are used to train the model, such as customer segmentation in a supermarket.

2.1 But what is Linear Regression?

Linear Regression is a statistical methodology that allows us to predict the relationship between two or more variables. It assumes a linear relationship between the independent variables and the dependent variable (which is to be predicted), and attempts to find the best fitting line that describes this relationship.
As shown above, it utilises past data (including input and corresponding output) to give some predicted output for any continuous input stream.

2.2 Current Research

The research paper cited Maulud, Dastan & Mohsin Abdulazeez, Adnan. (2020). A Review on Linear Regression Comprehensive in Machine Learning. Journal of Applied Science and Technology Trends. 1. 140-147. 10.38094/jastt1457. analyses the use of Linear Regression in data analysis, and their accuracy in various fields. Some notable results are listed below:

Data Set	Technique	Accuracy	Ref. in Paper
Aero-Material	Multiple Linear Regression Model	99.89%	[44], 2018
ANTLR	Simple Linear Regression Model	90.08%	[11], 2019
Pima Indian Diabetes	Multiple Linear Regression Model	82.1%	[14], 2019
Marketing	Multiple Linear Regression Model	84%	[47], 2018
3D Coordinate Medical Data	Multiple Linear Regression Model	X:92%, Y: 90%, Z: 80%	[45], 2018

Thus, Linear Regression is still being used extensively in modern research and can provide accurate predictions of future data on the basis of current data, despite being a very simple and easy to execute model.

3 Simple Linear Regression

In simple linear regression, there is one independent variable and one dependent variable. The model estimates the slope and the intercept of the line of best fit, which helps us to get the relationship between the variables.
In the simplest representation possible, we may have a line with slope β and y-intercept α such that the relationship between any independent variable x and a corresponding dependent variable y can be written as:

$$y = \alpha + \beta \cdot x$$

3.1 The Linear Regression Formula

In most practical cases, such a simple relationship may not always hold exactly for unobserved set of values of dependant and independent values (i.e., the values that we are to predict). These deviations can be factored into the equation to build a more accurate linear regression model.
For n elements in a data set $\{(x_i, y_i) : i \in (1, 2..., n)\}$ we can involve an error term ϵ_i by saying

$$y_i = \alpha + \beta \cdot x_i + \epsilon_i$$

Now, our goal is to find α and β based on the past data, such that we can predict values of our dependant variable y_i for any given independent variable x_i and minimise the error ϵ_i.
Now, we can create a mathematical formulation of our goal. We wish to estimate values of α and β which can provide a "best fit" line for all the past data points. In most practical cases, the best fit is taken on the basis of the mean squared error between predicted values of the dependant variables for our choice of α and β, against the actual values as

stated in the data.
Thus, we want to find:

$$min_{\alpha,\beta} J(\alpha,\beta)$$

$$J(\alpha,\beta) = \frac{1}{n}\sum_{i=1}^{n}\epsilon_i^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \alpha - \beta \cdot x_i)^2$$

Here, $J(\alpha,\beta)$ is commonly called the "cost function" since it calculates the error incurred by our choice of α and β. Note: the value ϵ_i is not a part of our equation of line of best fit, it is purely added to denote the need for an error term in the equation, while calculating β.

4 Generalisation to Multivariate Linear Regression and Formulation of our Methodology

We may have far more than 1 independent variable that determines the value of our dependant variable. This is known as **Multiple Linear Regression** or **Multivariate Linear Regression**. We can simply extend our previously summarised math for multiple variables in this case.
We are given a data set of p independent variables (also known as "fields") as such $\{(y_i, x_{i1}, ...x_{ip}) : i \in (1, 2...n)\}$.
We generalise our equation to account for them through multiple coefficients, which we will label using subscripts ofβ, and include the same error term as usual.

$$y_i = \beta_0 + \beta_1 \cdot x_{i1} + ... + \beta_p \cdot x_{ip} + \epsilon_i$$

One key factor to notice is that we have $p+1$ coefficients for p fields. This will become important in our matrix formulation.
Now, we store this data in matrix form for ease of computation. If β is the column matrix of coefficients, then we can clearly define the sum as:

$$y_i = x_i^T \cdot \beta + \epsilon_i$$

Here, we may assume that the value x_{i0} is 1, as a dummy to keep the coefficient β_0 unaltered. Here, ^{T} denotes transpose.
Now, we can define the matrix y as the column matrix of dependent variables in our data-set, x as the column matrix of values x_i^T, β as the column matrix of coefficients in our equation, and ϵ as the column matrix of error terms to be added.
To summarise:

$$x = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1p} \\ 1 & x_{12} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \dots & x_{np} \end{pmatrix}$$

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

$$\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}$$

$$\epsilon = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

And we can define the equation as:

$$y = x \cdot \beta + \epsilon$$

To summarise the formulation:

- $\underline{y}$: is a vector of observed values / dependent variables. It is represented as an n x 1 matrix.
- $\underline{x}$: can be seen as a matrix of row vectors x_i, which are independent variables. Usually, it is represented as an n x $p+1$ matrix with the element $x_{(i,j)}$ representing the j^{th} coefficient of the i^{th} data-set. Any element $x_{(i,0)}$ is set as 1 to leave the coefficient β_0 independent of all variables in every equation. This is the reason that we have a matrix with $p+1$ columns for p fields. In an arbitrary data set, we may have to manually add this column of 1's to our input data set.
- $\underline{\beta}$ is the coefficient matrix. Our goal is to find the matrix that minimises our error, given by $\sum_{i=1}^{n} \epsilon_i^2$ as before. It is a $p+1$ dimension vector.
- $\underline{\epsilon}$ is our error matrix that accounts for the noise in most practical data sets. We want to minimise the error that is interpreted from this.

To generalise our goal, we can say that our goal is to find β such that $||\epsilon||_2^2$ is minimised. Here, $||a||_p$ represents the p-norm of the vector a.

5 Finding the Best Fit

5.1 The Normal Equation

Finding the best fit using the normal equation involves pure linear algebra.

5.1.1 Derivation of the Normal Equation

Assuming the same terminology as before, let us assume we have n training examples in our data set, and p fields. Thus, we have:

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

$$x = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1p} \\ 1 & x_{12} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \dots & x_{np} \end{pmatrix}$$

We want to find the weights, β which are of the form:

$$\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}$$

Now, we can begin the derivation. The error, as defined before, is simply given as:

$$\epsilon_i = y_i - (\beta_0 + x_{i1} \cdot \beta_1 + \ldots + x_{ip} \cdot \beta_p)$$

Thus, we can define the matrix as follows:

$$\epsilon = \begin{pmatrix} y_1 - (\beta_0 + x_{11} \cdot \beta_1 + \ldots + x_{1p} \cdot \beta_p) \\ y_2 - (\beta_0 + x_{21} \cdot \beta_1 + \ldots + x_{2p} \cdot \beta_p) \\ \vdots \\ y_n - (\beta_0 + x_{n1} \cdot \beta_1 + \ldots + x_{np} \cdot \beta_p) \end{pmatrix} = y - X \cdot \beta$$

The term that we want to decrease is (let):

$$L = \epsilon_1^2 + \epsilon_2^2 + ... + \epsilon_n^2$$

From some basic observation, we see that this is:

$$L = \epsilon^T \cdot \epsilon = (y - X \cdot \beta)^T \cdot (y - X \cdot \beta)$$

This can be written as:

$$L = (y^T - \beta^T \cdot X^T) \cdot (y - X \cdot \beta) = (y^T \cdot y) - (y^T \cdot X \cdot \beta) - (\beta^T \cdot X^T \cdot y) + (\beta^T \cdot x^T \cdot x \cdot \beta)$$

Using the fact that $(A + B)^T = A^T + B^T$ and $(AB)^T = B^T A^T$

To minimise this loss function L we need to take its derivative with respect to the weights β. For this, we use the following facts:

- L is a one dimensional vector. To find its derivative with respect to vector β we find the following, and denote it as a gradient:

$$\nabla_\beta L = \begin{pmatrix} \frac{\partial L}{\partial \beta_1} \\ \frac{\partial L}{\partial \beta_2} \\ \vdots \\ \frac{\partial L}{\partial \beta_p} \end{pmatrix}$$

  We set this as 0, to get the values of β_0 to β_p.

- This gradient is defined as follows:

$$\nabla_\beta(\beta^T \cdot a) = \nabla_\beta(a^T \cdot \beta) = a$$

  for any vector defined as:

$$a = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \end{pmatrix}$$

Clearly, we can now state the following:

$$\nabla_\beta(y^T \cdot y) = 0$$
$$\nabla_\beta(y^T \cdot x \cdot \beta) = \nabla_\beta((x^T \cdot y)^T \cdot \beta) = x^T \cdot y$$
$$\nabla_\beta(\beta^T \cdot x^T \cdot y) = x^T \cdot y$$
$$\nabla_\beta(\beta^T \cdot x^T \cdot x \cdot \beta) = (x^T \cdot x \cdot \beta) + (\beta^T \cdot x^T \cdot x)^T = 2(x^T \cdot x \cdot \beta)$$

The last result was acquired by chain rule. Next, we use the fact that $d(u+v) = d(u)+d(v)$ to finally solve for $\nabla_\beta L = 0$. Clearly, plugging in the acquired results into the the above equation, we get

$$2(x^T \cdot x \cdot \beta) = 2(x^T \cdot y)$$
$$\implies x^T \cdot x \cdot \beta = x^T \cdot y$$
$$\implies \beta = (x^T \cdot x)^{-1} \cdot x^T \cdot y$$

This is the normal equation. We use this to solve simple linear regressions.

Notice that we are essentially solving for β in the equation $y = x \cdot \beta$. Hence, the term $(x^T \cdot x)^{-1} \cdot x^T$ essentially acts like an inverse for x. Thus, we call it a "pseudo-inverse" of x. It is often denoted in the form x^+, and may be considered the closest term we can get to inverting a non-square matrix. It is also known as the **"Moore-Penrose Pseudo-Inverse"**.

To summarise, for a training set of input data x with corresponding output data y we can immediately find the value of vector β by:

$$\beta = x^+ \cdot y = ((x^T \cdot x)^{-1} \cdot x^T) \cdot y$$

5.1.2 Solving from the Normal Equation

We can easily solve directly from the normal equation by finding the pseudo-inverse, as mentioned before, of x to find β as:

$$\beta = x^{+} \cdot y = ((x^T \cdot x)^{-1} \cdot x^T) \cdot y$$

We can also factorise or decompose x to simplify the calculation of the inverse or pseudo-inverse.
We can perform QR factorisation, claiming that $X = Q \cdot R$ where Q is an orthogonal matrix ($Q^T \cdot Q = I$) and R is an upper triangular matrix. On simplification, we obtain the equation:

$$(R^T \cdot R) \cdot \beta = R^T \cdot Q^T \cdot Y$$

If x had linearly independent columns then R will be invertible and we can simply find the inverse, but to be more general, we do not make this assumption, and instead we find the pseudo-inverse of $R^T \cdot R$, once again.

$$\beta = (R^T \cdot R)^{+} \cdot R^T \cdot Q^T \cdot Y$$

We may also perform SVD decomposition, getting $X = U \cdot D \cdot V^T$ where U and V are orthogonal matrices and D is a diagonal matrix. On simplification, we get the equation:

$$\beta = V \cdot D^{+} \cdot U^T \cdot Y$$

where $^{+}$ once again represents the pseudo-inverse. For diagonal matrices, the definition of pseudo-inverse becomes simplified and is easy and fast to calculate. We simply obtain D^{+} by obtaining the reciprocal of all the non-zero diagonal elements of D.

Thus, there are many different ways to solve from the normal equation.

5.2 A brief mention of other methods

5.2.1 Gradient Descent

Let's return to the easy model of Simple Linear Regression. We have the equation with coefficients m and c:

$$y = m \cdot x + c + \epsilon$$

We take either random values for our initial estimates of m and c or some predetermined standard values. Then, we keep taking steps based on our current position and the distance from the actual value. If the loss from our cost function is more, we take larger steps away from our current estimates, and if the loss is small, we take much smaller steps. It is implemented as below:
First, the value of our loss function can be stated as:

$$J_{m,c} = \frac{1}{n} \sum_{i=0}^{n} (y_i - (m \cdot x_i + c))^2$$

We choose an arbitrary learning rate, that determines how much our values change with each step. For large data sets that determine more accuracy, we usually use smaller values

of L, such as 10^{-4}.
Next, we find the partial derivate of the loss function, with respect to m and c, and plug in the current values of x_i and y_i for the same. We obtain:

$$D_m = \frac{1}{n}\sum_{i=0}^{n} 2(y_i - (mx_i + c))(-x_i)$$

$$\implies D_m = \frac{-2}{n}\sum_{i=0}^{n} x_i(y_i - \bar{y_i})$$

where $\bar{y_i} = m \cdot x_i + c$,i.e., the predicted value, and y_i is the actual value.

$$D_c = \frac{-2}{n}\sum_{i=0}^{n}(y_i - \bar{y_i})$$

Then, we update the values of m and c as:

$$m := m - L \cdot D_m$$

$$c := c - L \cdot D_c$$

If the slope is large we need to move away from the current estimate by a larger value as we are far from the expected value, and the converse is also true (if the slope is small, we are close, do not move as much). L or the learning rate is essentially the rate at which we move, depending on the size of the data and the accuracy required, as mentioned before.

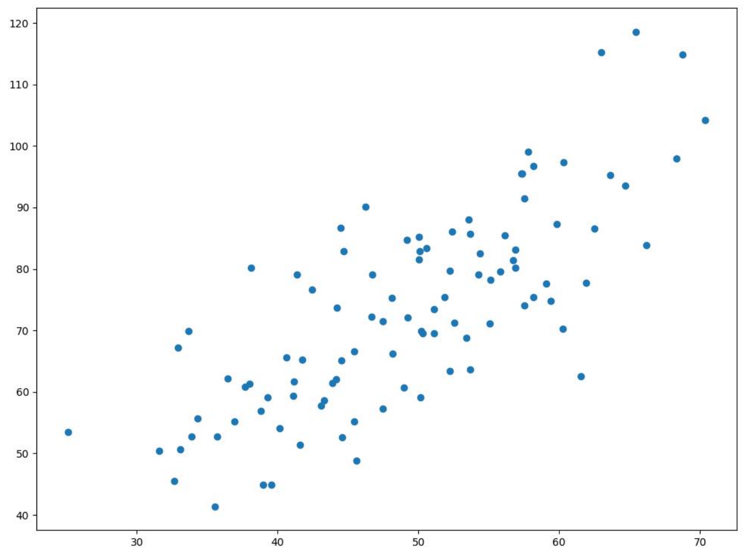
We continue this operation until our loss function $J_{m,c}$ shows that our loss is small (below some determined threshold) or, ideally, zero. We may even just do it for some determined number of iterations, rather than focusing on our loss, if our data must be calculated within some time constraint.
It is easy to see how this can be extended to multiple fields, but the Gradient Descent method extends beyond the linear algebra that is relevant to the material covered in our course, so we will not be going over it.

6 Simulating Linear Regression

We will be using the data set provided in the Towards Data Science article on Linear Regression, for our simulations.

The code for simulation of Gradient Descent will also be largely the same.

6.1 The data to study



This is the visualisation of the data from the article mentioned before, in a scatterplot. We will show our line of best fit in this.

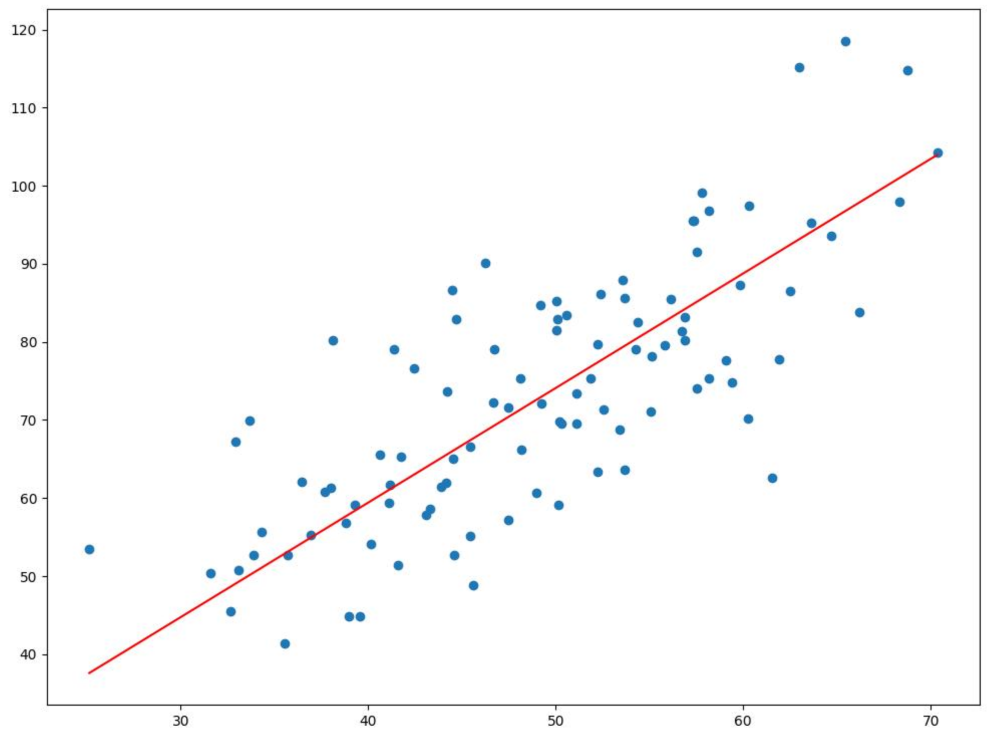
6.2 Using Gradient Descent

The x values are stored in a vector X, and the y values are stored in a matrix Y. Here, X is a vector because we are performing Simple Linear Regression since that is the nature of the input data-set.
The code is as shown below:

```python
# Making the imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (12.0, 9.0)

# Preprocessing Input data
data = pd.read_csv('data.csv')
X = data.iloc[:, 0]
Y = data.iloc[:, 1]

# Building the model
m = 0
c = 0

L = 0.0001  # The learning Rate
epochs = 10000  # The number of iterations to perform gradient descent

n = float(len(X)) # Number of elements in X

# Performing Gradient Descent
```

```
for i in range(epochs):
    Y_pred = m*X + c  # The current predicted value of Y
    D_m = (-2/n) * sum(X * (Y - Y_pred))  # Derivative wrt m
    D_c = (-2/n) * sum(Y - Y_pred)  # Derivative wrt c
    m = m - L * D_m  # Update m
    c = c - L * D_c  # Update c

# values of m and c stored
print (m, c)

# Making predictions
Y_pred = m*X + c

plt.scatter(X, Y)
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='red') #
    predicted
plt.savefig("GradDesc.jpg")
plt.show()
```

The result we observe is:



Things to note:

- To increase accuracy we generally increase the epochs (the number of iterations we choose to run the code) and decrease the learning rate.
- A low learning rate means we must have a higher number of iterations to ensure we actually do get to the final, accurate value.

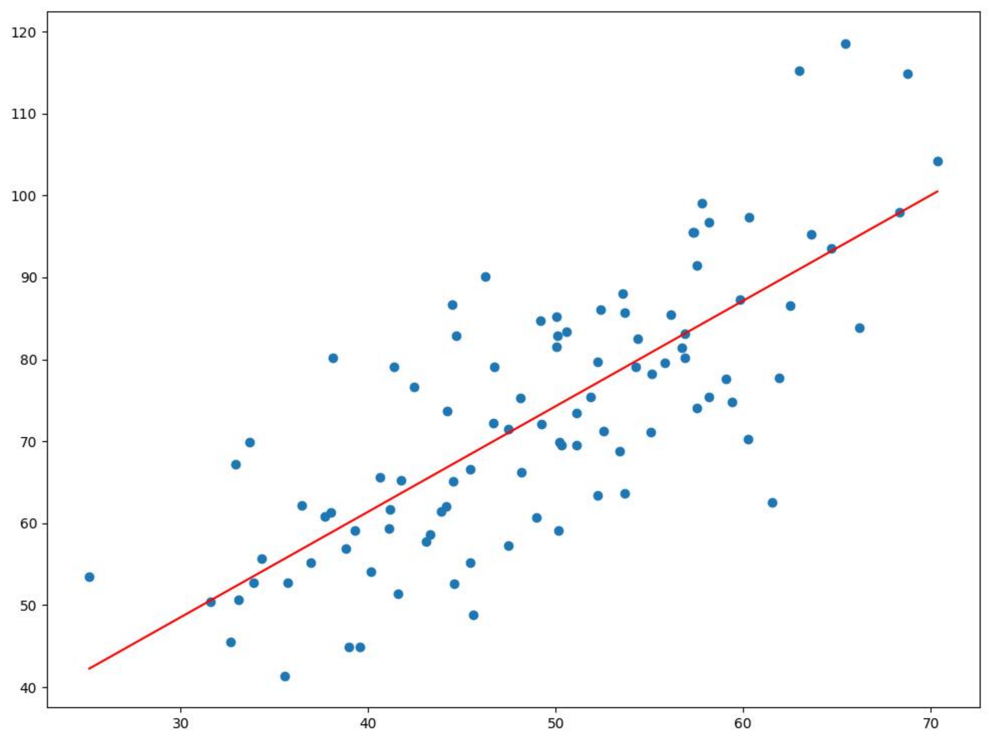
Thus, if we want accuracy, gradient descent is a good algorithm, but it is slow, as we need more iterations of code.

6.3 Using the Normal Equation

The code should be much simpler and will run much faster. There are no large number of iterations and all the time is occupied by matrix operations. The code will look something like this:

```python
# Making the imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (12.0, 9.0)

# Preprocessing Input data
data = pd.read_csv('data.csv')
X = data.iloc[:, 0]
Y = data.iloc[:, 1]

# adding a column of ones to X, to calculate beta
XwithOnes = np.c_[np.ones((X.size, 1)), X]
# calculating the pseudo-inverse of X(with ones)
X_inv = np.linalg.pinv(XwithOnes)

# calculating beta as pinv(X) * Y
beta = np.dot(X_inv, Y)
# print(beta)

# Making predictions for the known X
Y_pred = np.dot(XwithOnes, beta)

plt.scatter(X, Y)
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='red') #
    predicted
plt.savefig("NormalEqn.jpg")
plt.show()
```

The result we get is as follows:



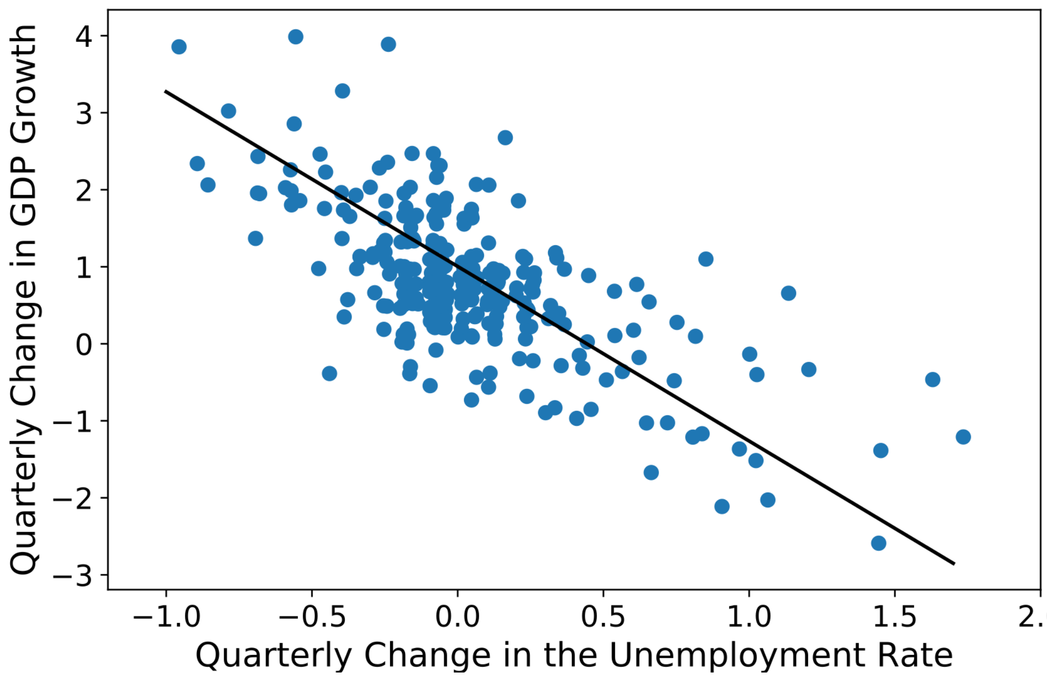
Things to note:

- The calculation is much faster, as stated before, but slightly less intuitive, with the need to add a column of ones to our data-set.
- It is often not as accurate as gradient descent, because the calculation of pseudo-inverse is problematic and often leads to inaccurate results.

Because of the above reasons, calculation using the normal equation is usually not performed by default in most modern statistical libraries and packages, for linear regression.

7 Applications

Linear Regression is commonly used in general machine learning models to predict data on the basis of past data.
A more specific application can be Okun's Law, which studies the relationship between unemployment and losses in a country's production. Often, when showing this relationship, we use a line of best fit which is obtained using linear regression, the same way as we obtained above in our simulations.



Above is a graph of US quarterly data from 1948 through 2016 estimating a form of Okun's law, in which they used a line of best fit, which can be obtained through linear regression.

7.1 Performing Regression on a real data-set

We will be using the Swedish Auto Insurance data set provided on Kaggle. It studies the number of claims made (X) against the total payout received from those claims (Y) in thousands of Kronor.

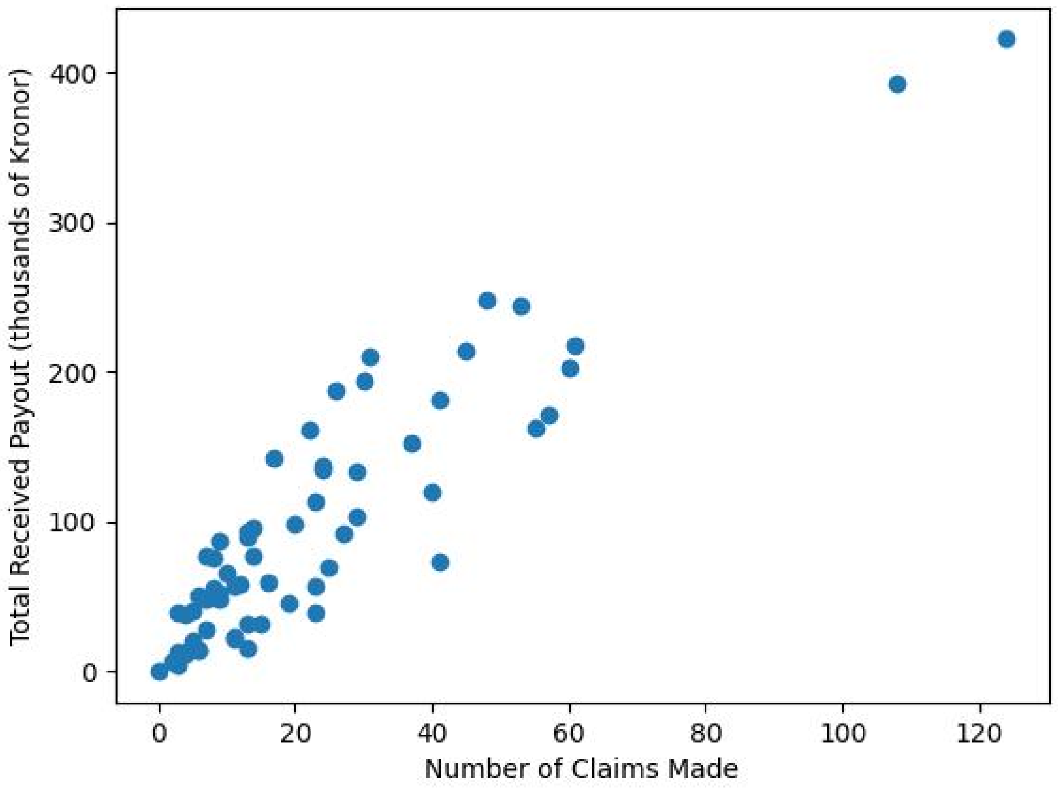
7.1.1 Checking the accuracy of our Linear Regression Model

First, we load the data and find the square root of the mean squared error, to get an idea of how effective Linear Regression is to study this data. We can do so with the following code:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

dataSet = pd.read_csv('swedish_insurance.csv')
dataSet.head()

X = dataSet.drop("PaymentOfClaims", axis = 1)
Y = dataSet.PaymentOfClaims

import sklearn.linear_model as lm

```

```
plt.scatter(X, Y)
plt.xlabel("Number of Claims Made")
plt.ylabel("Total Received Payout (thousands of Kronor)")
plt.savefig('InsuranceData.jpg')
plt.show()

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y)

lm = LinearRegression()
model = lm.fit(X_train, y_train)

from sklearn.metrics import mean_squared_error
y_pred = model.predict(X_test)
print(np.sqrt(mean_squared_error(y_test,y_pred)))
```

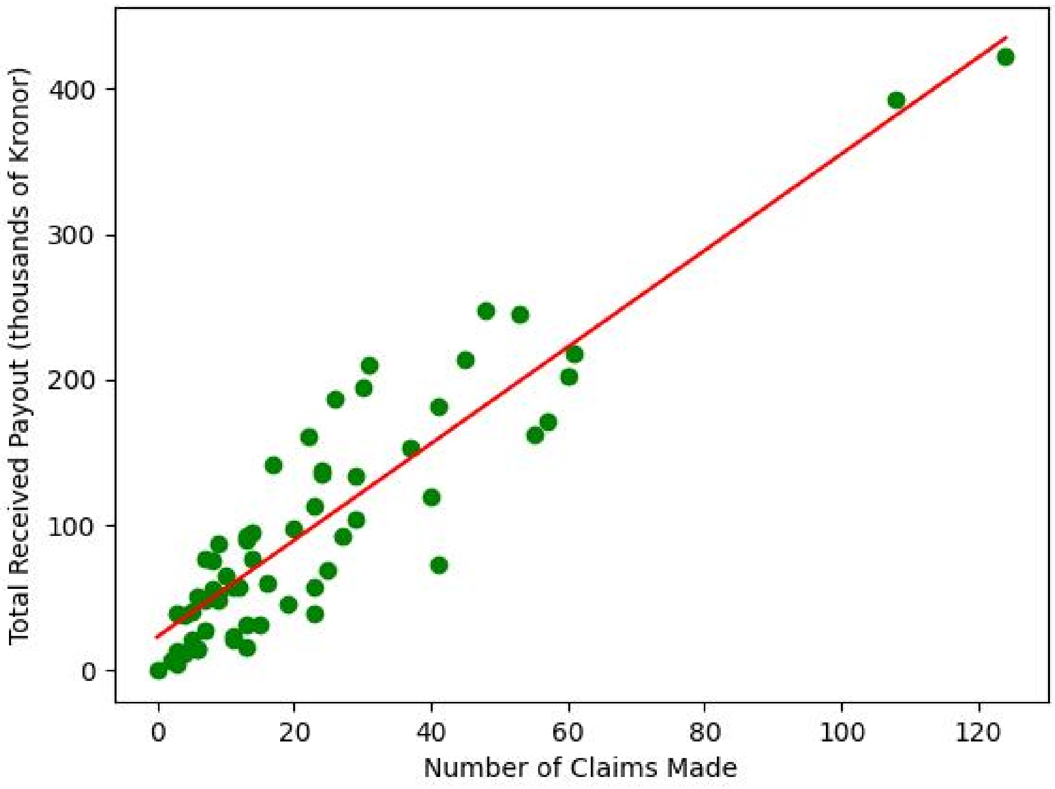
The program first visualises the data, which is shown below:



Then, it prints out the error, as calculated. In our test runs, we noticed errors of 30-40 in the values of 'Y'. Some methods to reduce this error could be to remove the outliers, or to collect more data and use Multivariate Linear Regression with more significant fields.

7.1.2 The Regression Model

Finally, we can draw the line of best fit and use our calculated coefficients to make predictions on the total payout for a given number of claims.



Thus, we performed Simple Linear Regression from scratch on a real data set, and used it to create a training model that can make predictions of the approximate insurance payout for a given number of claims, with an error of about 30,000 kronor.
Considering the simplicity of the model, this is a good simulation!

8 References

8.1 Introduction

- Types of Learning Models
- Difference between supervised and unsupervised

8.1.1 What is Linear Regression

- What is Simple Linear Regression?

8.2 Simple Linear Regression

- What is it?
- Formulation

8.3 Generalisation to Multivariate Linear Regression and Formulation of our Methodology

- Matrix Formulation of LR

8.4 Finding the best fit

8.4.1 The Normal Equation

8.4.2 Derivation of the Normal Equation

- Normal Equation Derivation for Regression

8.4.3 Solving from the Normal Equation

- Slides from Prof Hens
- Pseudo-Inverse

8.4.4 Gradient Descent

- Towards Data Science article on Gradient Descent

8.5 Simulating Linear Regression

- Same as above

8.6 Applications

- Graph of Okun's Law

8.6.1 A real data set

- Kaggle - Auto Insurance in Sweden

Support Vector Machines

Adithi Samudrala

1 Introduction

1.1 Overview

In this report, we are going to look at the Fundamentals of Support Vector Machines, SVM optimization and the techniques used, Kernel Functions and Applications.

1.2 S.O.T.A

SVMs are supervised learning models that analyze data and recognize patterns, used for classification and regression analysis. It can be utilized for characterization or relapse issues cited in the paper Scope of Support Vector Machine in Steganography.
Another such paper is A Method of Identifying Electromagnetic Radiation Sources by Using Support Vector Machines which provided a novel method for identifying EM radiation sources. The SVM was utilized to classify different radiation sources on the basis of the spatial characteristics of radiation sources.
SVMs remain an active area of research, and several advancements have been made in recent years.

2 Fundamentals of SVM

Support Vector Machines (SVMs) are a type of supervised learning algorithm that can be used for classification or regression tasks. The main idea behind SVMs is to find a hyperplane that maximally seperates the different classes in the training data. This is done by finding the hyperplane that has the largest margin, which is defined as the distance between the hyperplane and the closest data points from each class. Once the hyperplane is determined, new data can be classified by determining on which side of the hyperplane it falls.

3 What are Support Vector Machines?

SVM is a relatively simple algo used for classification and/or regression. It is more preferred for classification but is sometimes very useful for regression as well.
The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. The figure 1 clearly illustrates different elements of SVM

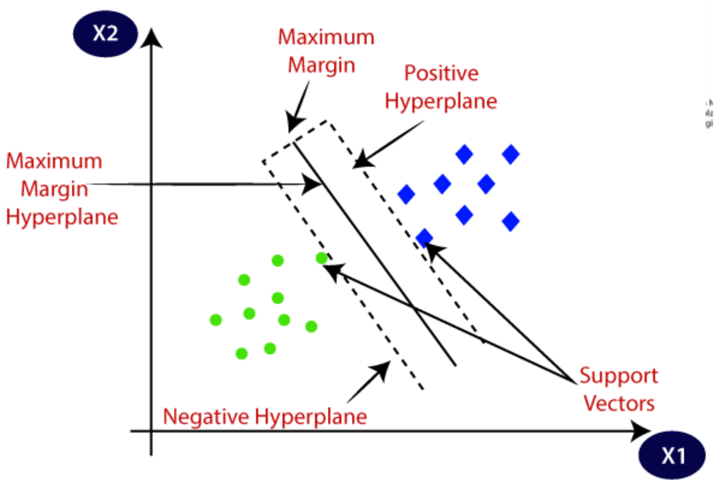


Figure 1:

4 Hyperplane and Support Vectors in the SVM algorithm

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the **hyperplane of SVM**.
The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features, then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.
A hyperplane always has a maximum margin, which means the maximum distance between the data points.

Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a **Support vector**.

5 Types of SVM

SVM can be of two types:

Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

5.1 Linear SVM

Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. We can easily separate these two classes. But there can be multiple

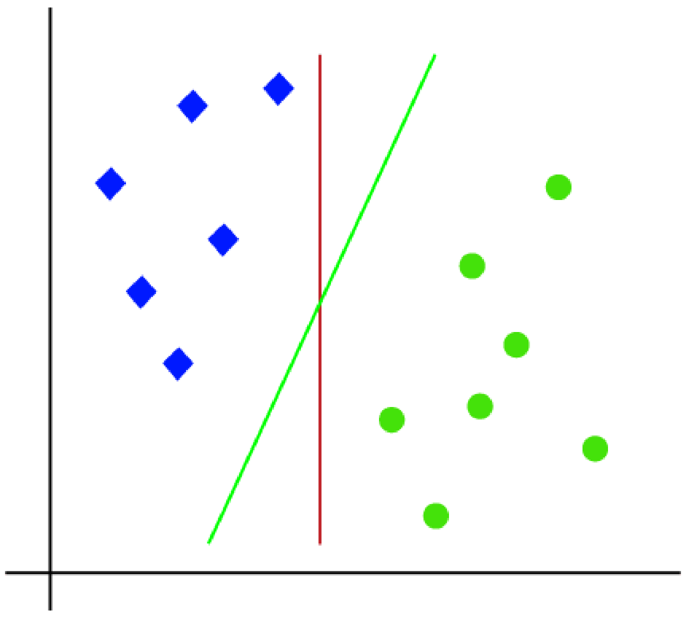


Figure 2:

lines that can separate these classes, as shown in 2 Hence, the SVM algorithm helps to find the best line or decision boundary, namely, the hyperplane. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as margin. And the goal of SVM is to maximize this margin. The hyperplane with maximum margin is called the optimal hyperplane, as shown in 1

5.2 Non-Linear SVM

Since we cannot seperate the data by just a straight line, we add one more dimension. For a dataset as shown in 3, we can find the third dimension by calculating it as follows:
$z = x^2 + y^2$
The sample space now becomes 4

6 Kernel Functions in SVM

SVM algorithms use a set of mathematical functions that are defined as the kernel. The function of kernel is to take data as input and transform it into the required form. Different SVM algorithms use different types of kernel functions. These functions can be different types. For example linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid. The most used type of kernel function is RBF because it has localized and finite response along the entire x-axis. The kernel functions return the inner product between two points in a suitable feature space so that every point is mapped into higher dimensional space via some transformation. Thus by defining a notion of similarity, with little computational cost even in very high-dimensional spaces.

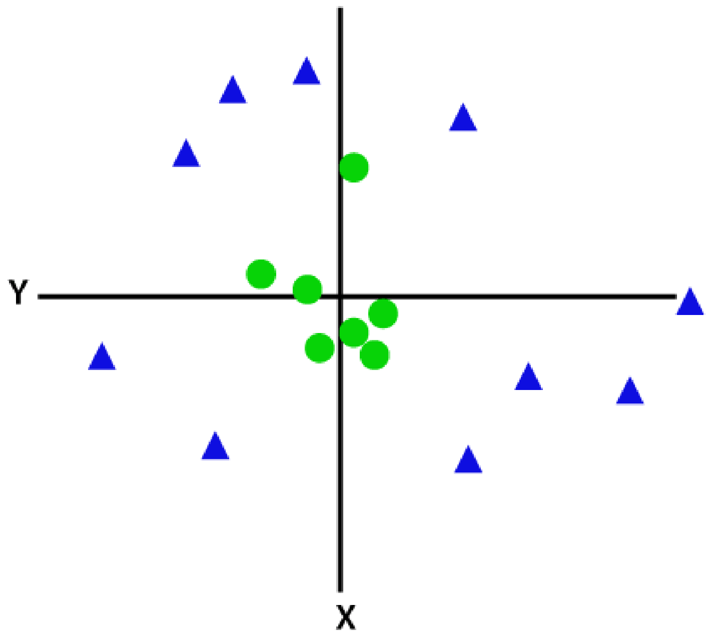


Figure 3:

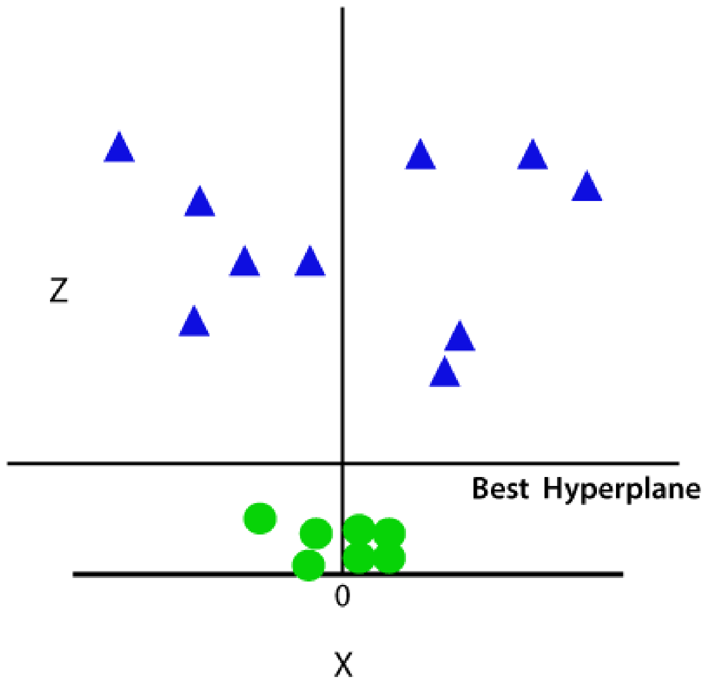


Figure 4:

6.1 Polynomial Kernel

It is used in image processing
Equation is:
$k(x_i, x_j) = (x_i.x_j + 1)^d$ where d is the degree of the polynomial

6.2 Radial basis function

Radial Basis Kernel is a kernel function that is used in machine learning to find a non-linear classifier or regression line.
Kernel Function is used to transform n-dimensional input to m-dimensional input, where m is much higher than n then find the dot product in higher dimension efficiently. The main idea to use kernel is: A linear classifier or regression curve in higher dimensions becomes a Non-linear classifier or regression curve in lower dimensions.

6.2.1 Mathematical Definition of Radial Basis Kernel:

$$k(x_i, x_j) = exp(-\gamma||x_i - x_j||^2)$$

for

$$\gamma > 0$$

sometimes parametrized using:

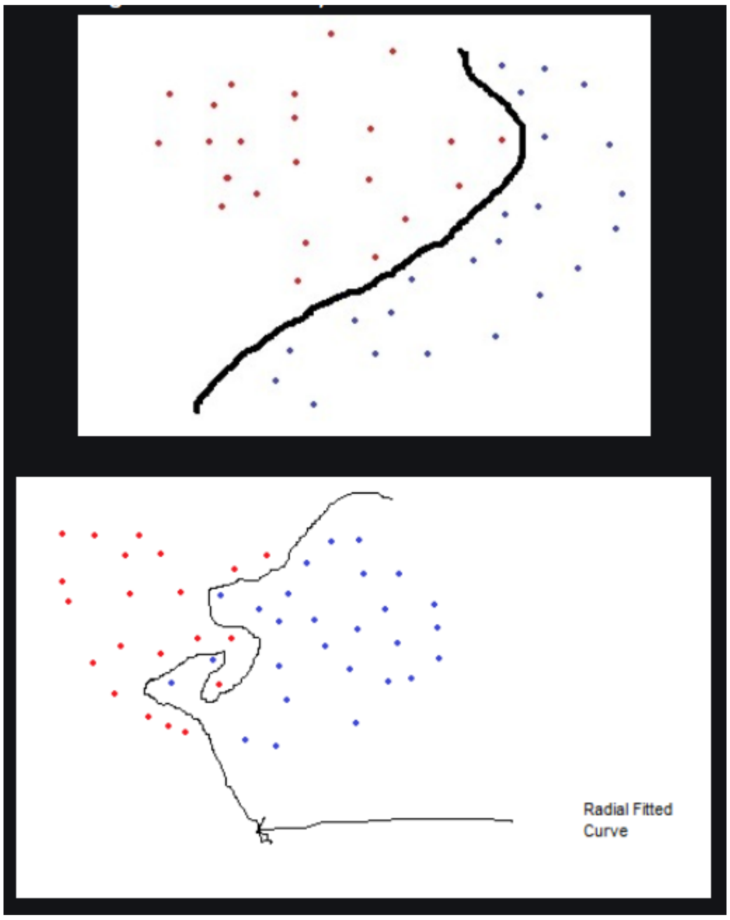
$$\gamma = 1/2\sigma^2$$

γ is a scalar that defines how much influence a single training example (point) has and x_i, x_j are vector point in any fixed dimensional space. But if we expand the above exponential expression, it will go upto infinite power of x_i and x_j, as expansion of e^x contains infinite terms upto infinite power of x hence it involves terms upto infinite powers in infinite dimension.
If we apply any of the algorithms like perceptron Algorithm or linear regression on this kernel, we would be applying our algorithm to new infinite-dimensional datapoint we have created. Hence it will give a hyperplane in infinite dimensions, which will give a very strong non-linear classifier or regression curve after returning to our original dimensions.

6.2.2 Why Radial Basis Kernel Is much powerful?

Although we are applying linear classifier/regression it will give a non-linear classifier or regression line, that will be a polynomial of infinite power. And being a polynomial of infinite power, radial basis kernel is a very powerful kernel, which can give a curve fitting any complex dataset.
The main motive of the kernel is to do calculations in any d-dimensional space where d ¿ 1, so that we can get a quadratic, cubic or any polynomial equation of large degree for our classification/regression line. Since Radial basis kernel uses exponent and as we know the expansion of e^x gives a polynomial equation of infinite power, so using this kernel, we make our regression/classification line infinitely powerful too.

6.2.3 Some Complex Dataset Fitted Using RBF Kernel easily:



6.3 Hyperbolic tangent kernel

Used in neural networks
Equation is:

$$k(x_i, x_j) = tanh(kx_i.x_j + c)$$

6.4 Sigmoid kernel

Used as the proxy for neural networks.
Equation is:

$$k(x, y) = tanh(\alpha x^T y + c)$$

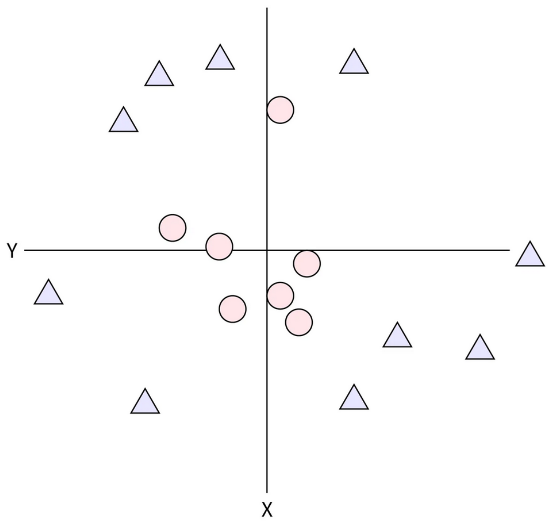
6.5 ANOVA radial basis kernel

Used in regression problems.
Equation is:

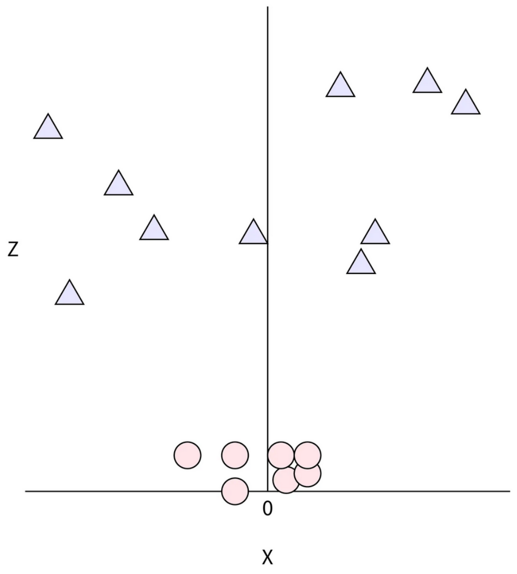
$$k(x, y) = \Sigma_{k=1}^{n} exp(-\sigma(x^k - y^k)^2)^d$$

7 Handling Nonlinear Data with SVM

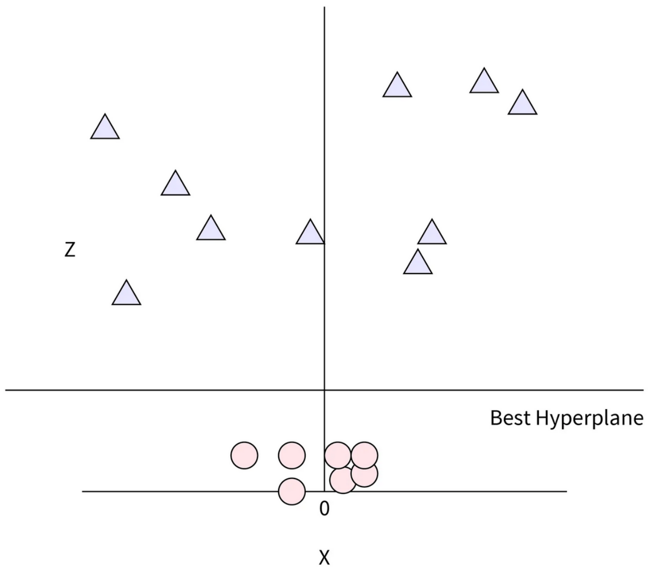
To classify a non-linear dataset, we project the dataset into a higher dimension in which it is linearly separatble. For example:



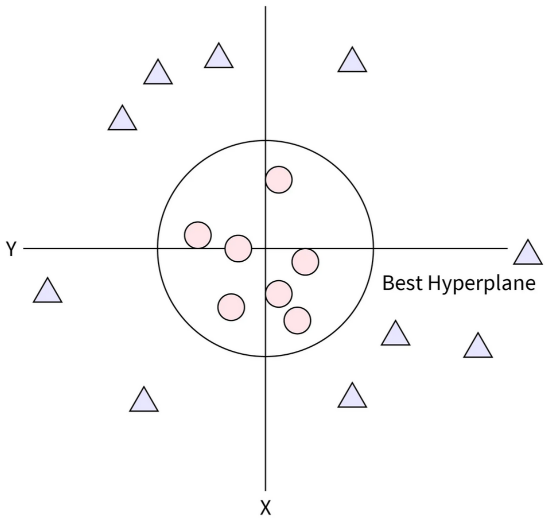
The above data points cannot be separated with a single straight line. But, we can see that that a somewhat circular hyperplane could possibly separate this data. Thus, we introduce a third coordinate Z, with the help of X and Y, using $Z = X^2 + Y^2$. After projecting the dataset into this higher dimension, the new graph we obtain is:



Now, this data is clearly linearly separable by a straight-line hyperplane.
In 3D, the line would look somewhat like this:



If we represented the same hyperplane back in 2-Dimensions, the graph would look as follows:



Thus, a non-linear SVM just takes the data points to a higher dimension to be linearly separable in that higher dimension, and then uses the same linear SVM classification to separate the data points.

7.1 The role of the Kernel Function

The function, as stated before, is:

$$K(x, x') = e^{-\gamma||x-x'||^2}$$

Here, γ is a scalar that defines how much influence a training point has.
The kernel function transforms the n-dimensional input space to an m-dimensional space,

(usually where $n >> m$) so that we can efficiently do the required calculations in a higher dimension, easily.
Thus, the non-linear data is effectively projected onto a higher dimension such that it is easier to classify the data where it could be linearly divided into a plane. We achieve this mathematically using the Kernel function, for efficiency and convenience.

8 Applications of SVM

Some applications of SVM in the real world include:

- **Face Detection**: SVMs can classify parts of the image as a face and other parts as "non-face", and use SVM algorithms to create a square boundary around the face. To elaborate: it contains training data of nxn pixels with two classes, face (say, +1) and non-face (say, -1). It then extracts features from each pixel as face or non-face. It creates a boundary around faces on the basis of pixel brightness and other characteristics, and then uses this data to classify each image by using the same process.
- **BioInformatics**: SVM can be used for protein classification and cancer classification. We can use SVM for identifying the classification of genes, patients on the basis of genes, and other biological problems.
- **Handwriting Recognition**: We use SVMs to recognize handwritten characters used widely.
- **Text Categorization**: SVMs allow text categorisation, by using training data to classify documents into different categories. It categorises on the basis of some generated score compared with some threshold value.

9 Conclusion

Support Vector Machines are a type of supervised learning algorithm, generally used for classification tasks, that operates by finding a hyperplane that maximally separates different classes in the training data. This becomes very useful in both separation and detection algorithms, as we can classify data as either being a part of some set, or not a part of it.
Thus, it can be used for face detection, protein detection, handwriting recognition, etc.

10 References

10.1 Fundamentals of SVM

Fundamentals of SVM

10.2 About SVMs

What are Support Vector Machines?
same as above

10.3 Types

Types
Non-linear

10.4 Kernel functions

Kernel and types
Radial basis kernel

10.5 Handling Non-Linear Data

How do we handle non-linear data in SVMs?
Using SVMs to perform classification

10.6 Applications

Applications of SVMs

Principal Component Analysis

Kavin Aravindan

1 Introduction

In this document we will be talking about Principal Component Analysis also known as PCA. We will discuss about What is PCA, How PCA works,where it can be used effectively and perform PCA on a data set. We will also look at and discuss some recent developments in the field

1.1 What is Dimensionality Reduction and Why do we Need it

Let's start of by defining dimensionality reduction

Definition (Dimensionality Reduction)**.** *Dimensionality reduction, is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data.*

Dimensionality reduction can be divided into two different types

- **Feature Extraction:** It starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps, and in some cases leading to better human interpretations.
- **Feature Selection:** In machine learning and statistics, feature selection is the process of selecting a subset of relevant features (variables, predictors) for use in model construction.

1.2 Curse of Dimensionality

This term was coined by Richard Bellman to describe the challenges in optimising functions with too many input factors/variables. What impact does this have on machine learning and dimensionality reduction? We have two main ones

1. When our initial data set has too many properties, we may run into problems such as not being able to train our model effectively due to the model following the training set too closely
2. Clustering, an important idea in machine learning becomes increasingly difficult with the increase of properties in the data set. This is because all the data points look different with no apparent similarity.

Dimensionality reduction methods are often used to solve the issues that are caused due Curse of Dimensionality. Principal component analysis is a commonly used feature extraction based dimensionality reduction method.

2 Principal Component Analysis

2.1 What is Principal Component Analysis

We define Principal Component Analysis as follows

Definition (Principal Component Analysis)**.** *Principal Component Analysis is a statistical technique for reducing the dimensionality of a data set. This is accomplished by linearly transforming the data into a new coordinate system where (most of) the variation in the data can be described with fewer dimensions than the initial data*

The main goal of principal component analysis, often referred to as PCA, is to throw away features which contribute little to the variation in the data, making it easier to understand the data. It also allows for us to plot the data on a 2D or 3D plane, helping us visually identify clusters of data points.

2.2 Performing PCA

Before starting we want to make sure that our data is in the right format. The data is usually in the form of a table. The table contains one column for a dependant variable and the rest of the columns have independent variables/features. We will be performing all our operations on the independent variable columns only.
We construct a matrix $n * p$ X to store the p features and n data points

1. **Standardizing our data:** We first want to mean center our data. What this means is that we want to take the mean value of each column and subtract it from its respective column.

   After mean centering our data, we have to make a choice based on our requirements. Is the importance of each feature dependent on its variance? If yes, we've finished standardizing our data. Otherwise divide each value in the column by the standard deviation of that column. Let's call our standardized matrix $\hat{X}$

2. **Covariance Matrix:** Now we want to compute the covariance matrix. This can be found easily. Let C denote our covariance matrix then C is given by

$$C = \hat{X}^T \hat{X}$$

3. **Eigenvectors and Eigenvalues of C :** Now we want to calculate the eigenvectors and eigenvalues of the covariance matrix we just found. To do this we take advantage of the fact that C is a diagonalizable matrix. This fact allows us to decompose C into the form of

$$ZDZ^{-1}$$

   where Z is a matrix consisting of the eigenvectors of C and D is a diagonal matrix with its diagonal elements being the eigenvalues of C. (Note: The first element of the diagonal corresponds to the first eigenvector and so on)

4. **Sorting the Eigenvectors and Eigenvalues:** We now take the eigenvalues and eigenvectors we obtained and sort them. By this we mean sort the eigenvalues and change the position of eigenvectors in Z accordingly. Let's call this sorted eigenvector matrix $\bar{Z}$. Also note that these eigenvectors are the principal components we want.

5. **Projecting our Data:** After computing $\bar{Z}$ we need to project our standardized data $\hat{X}$ onto the principal components. This is done with the following computation

$$\bar{X} = \hat{X}\bar{Z}$$

What this does is make every variable a linear combination of the original variables. A thing to note is that the principal components are always orthogonal to each other.

6. **Dropping Components:** There are three main ways in which we do this

   - If you want to be able to visually look at the data using a 2D or 3D plot then we just keep the top 2 or 3 principal components and drop the rest
   - Choose a cumulative variance threshold and pick the required amount of principal components to satisfy this and drop the rest.
   - The last method is to plot a scree graph. A scree graph shows the variance of each principal component. By plotting the scree graph we can visually identify where there is a drop off in the variance values and drop the principal components based on that.

That finishes up the primary steps of PCA. This data can then be used to train machine learning models or any of the other multiple use cases PCA has.

2.3 Why PCA Works

While the above section explains to us the procedure for performing PCA, it doesn't really explain why it works. Let's Discuss some of the intuition behind PCA

- The covariance matrix contains data about how each variable in $\hat{X}$ is related to the other variables present in it. We can see this in an example for two-dimensional data with two variables X and Y

$$\begin{bmatrix} Cov(X,X) & Cov(X,Y) \\ Cov(Y,X) & Cov(Y,Y) \end{bmatrix}$$

  where

$$Cov(X,Y) = \frac{\sum(X-\bar{X})(Y-\bar{Y})}{Number of Data Points}$$

  Here $\bar{X}$ and $\bar{Y}$ denote the mean of the variables X and Y.

- The above fact allows us to pick and mix variables/take a linear combination of the variables to maximize our variance in one direction. Think of it like trying to compress as much important data as possible into one variable.

- The linear combinations were talking about above is what gives us our principal components. An additional factor that helps PCA is that the principal components are always orthogonal to each other meaning the linear combinations are always independent of each other.

- A neat thing that covariance matrix gives us is the importance of each eigenvector through the eigenvalues, which allows us to perform the primary operation of dropping variables/features in PCA.

2.4 Drawbacks of PCA

While PCA might seem to be a perfect solution to the problem of high dimension data sets, it is not without its flaws.

1. PCA assumes that data is linearly related and performs poorly in cases where it is not.
2. The principal components generated by PCA are not easy to interpret. Since the principal components are linear combinations of the features, it becomes difficult to determine which feature is the most important

3 Simulating PCA using Python

We will be performing PCA on the Breast Cancer Wisconsin (Diagnostic) data set

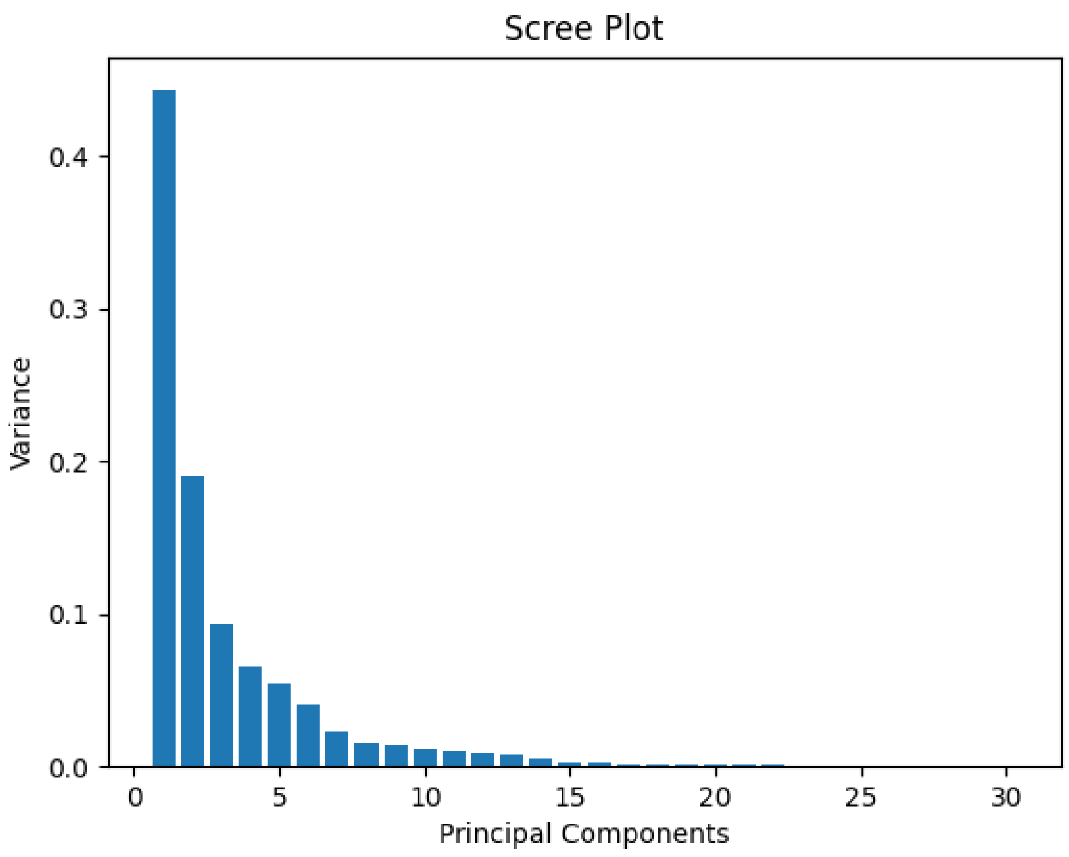
3.1 Code

```
#importing requried libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns

#loading the dataset and trim it to get the matrix X
dataset = pd.read_csv('data.csv')
X = dataset.iloc[:,2:32]

#finding the mean of the columns of the dataset and mean centering the
    data
MeanMatrix = X.mean(axis=0)
X_hat = X - MeanMatrix

#calculating the standard deviation of the dataset and dividing the
    columns by it
StdDeviation = X_hat.std()
X_hat = X_hat / StdDeviation

#computing the covariance matrix C
C = np.matmul(X_hat.T, X_hat)
#Diagonalizing C to obtain the eigenvalues and eigenvectors(Z)
eigenvalues, Z = np.linalg.eig(C)
#sorting the eigenvectors based on the eigenvalues
sort_idx = np.argsort(eigenvalues)
eigenvalues = eigenvalues[np.flip(sort_idx)]
Z = Z.T
Z_bar = Z[:, np.flip(sort_idx)]

#scree plot
sum = 0
for value in eigenvalues:
    sum += value
eigenvalues /= sum
plt.bar(range(1,31), eigenvalues)
```

```python
plt.xlabel('Principal Components')
plt.ylabel('Variance')
plt.title("Scree Plot")
plt.savefig('scree.png')

#projecting our data on to the principal components
X_bar = np.matmul(X_hat, Z_bar.T)
sns.scatterplot(data=X_bar, x=X_bar.iloc[:,0], y=X_bar.iloc[:,1], hue=
    dataset['diagnosis'], palette='Set1')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title("Plot of First Two Principal Components")
plt.savefig("2D.png")
```

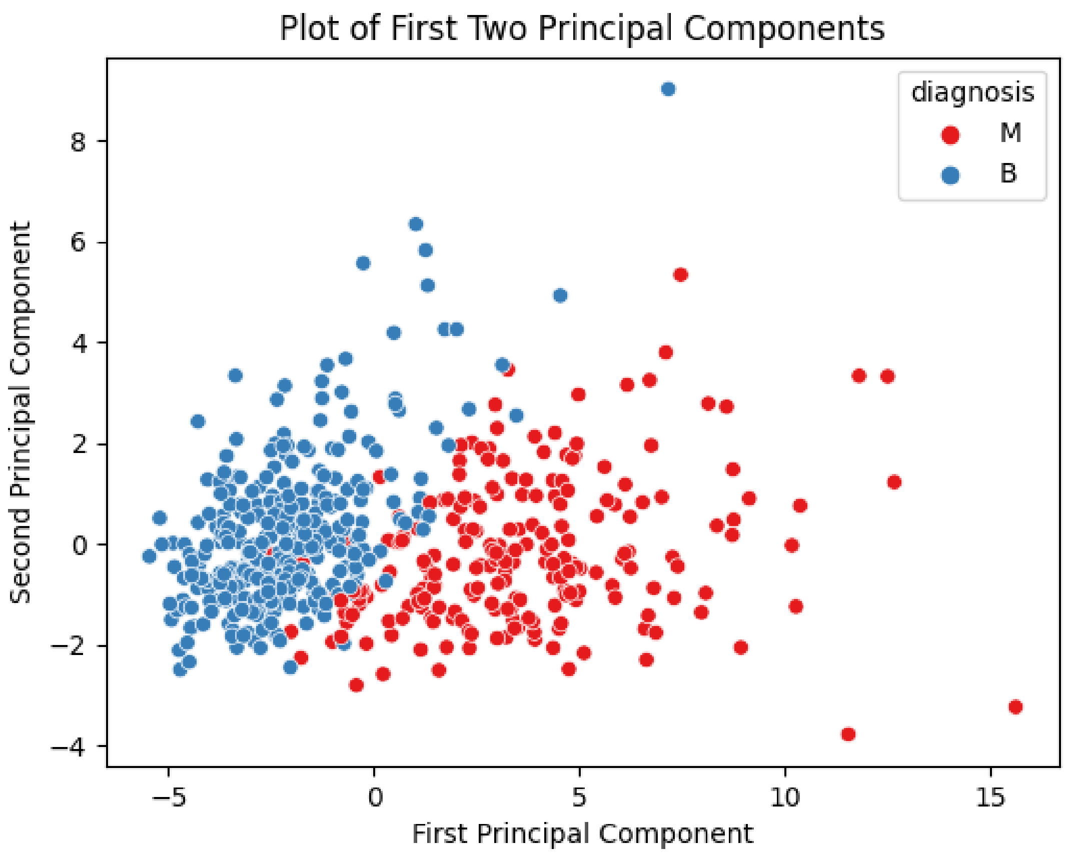
Listing 1: Simulating PCA

3.2 Scree Plot

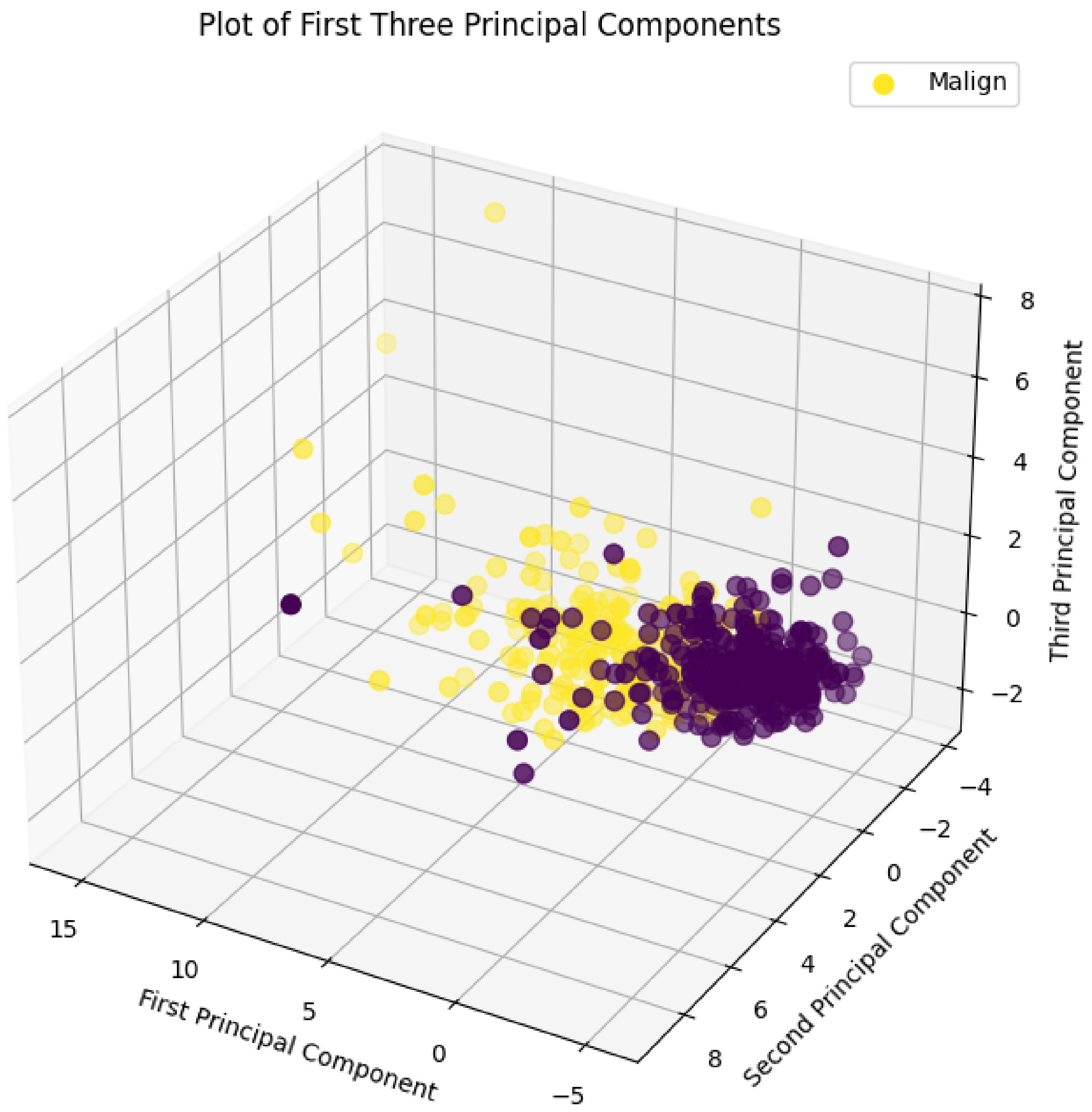


Notice that from the scree plot that there is a significant drop in the variance after the 8*th* principal component. With this info we can say that only the top 8 Principal components contribute the most to the data.

3.3 2D plot


Plot of First Two Principal Components
diagnosis
M
B
Second Principal Component
First Principal Component
8
6
4
2
0
−2
−4
−5
0
5
10
15

3.4 3D plot



The 3D and 2D show to us visible clusters in the data which may lead to us finding useful conclusions.

4 Current Research

Here, we look at some current research relating to principal componenet analysis.

- **Implementation of Principal Component Analysis on Masked and Non-masked Face Recognition**
  This paper uses PCA in order to analyse faces. A collection of faces can be used as the dataset from which we obtain "eigen faces" or eigen vectors that produce the most variation, on which PCA can be done.

- **A Linear Model Based on Principal Component Analysis for Disease**

Prediction
The diagnosis of diseases involve the analysis of a variety of features, all of which constitute a particular "dimension". Applying PCA on these allows us to find the most prominent symptoms, that maximizes clustering of data, which simplifies the process of diagnosis and prediction.

5 Variations of PCA

Let's look at some variations of PCA and the improvements they offer compared to traditional PCA

- Incremental PCA: Incremental PCA allows for efficient handling of larger data sets compared to PCA. It achieves this by processing the data in batches reducing the memory required.
- Sparse PCA: Sparse PCA overcomes PCA's flaw of using a linear combination of all the variables to generate the principal components and instead uses a linear combination of few variables to generate the principal components.
- Probabilistic PCA Probabilistic PCA adds to PCA by providing ways to estimate missing values in the data set. It uses the other data points in the data set to do this.

6 References

6.1 Introduction

1. Dimensionality Reduction, Wikipedia
2. Feature extraction, Wikipedia
3. Feature selection, Wikipedia
4. The Curse of Dimensionality

6.2 Principal Component Analysis

1. A One-Stop Shop for Principal Component Analysis
2. A Step-By-Step Complete Guide to Principal Component Analysis

6.3 Simulating PCA using Python

1. Breast Cancer Wisconsin (Diagnostic)

6.4 Current Research

1. M. S. Ejaz, M. R. Islam, M. Sifatullah and A. Sarker, "Implementation of Principal Component Analysis on Masked and Non-masked Face Recognition," 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT), Dhaka, Bangladesh, 2019, pp. 1-5, doi: 10.1109/ICASERT.2019.8934543.
2. H. Roopa and T. Asha, "A Linear Model Based on Principal Component Analysis for Disease Prediction," in IEEE Access, vol. 7, pp. 105314-105318, 2019, doi: 10.1109/ACCESS.2019.2931956.

6.5 Variations of PCA

1. Sparse PCA, Wikipedia
2. Incremental PCA, scikit-learn
3. Probabilistic PCA

Report Conclusion

Thus, Linear Algebra is used extensively in the most integral of Machine Learning topics. Simple concepts such as matrix multiplication, all the way to more complex ones like eigen-decomposition and factorisation, all play a key part in teaching a machine to make predictions based on historical data, whether it is for regression, classification, clustering, or any given purpose.

THANK YOU