CHAPTER 3
RESEARCH METHODOLOGY

3.1. Dataset

The dataset used in this study was taken from the UCI Machine Learning Repository website. The data consists of 12330 rows and 18 columns with different values according to their respective attributes. Data preprocessing is done by dividing the data into two parts, the train set and the test set. The distribution is done with a ratio of 80% training and 20% testing with the value of "Revenue" as the target class. This division is based on the Pareto Principle which states that in most cases, 80% of the effects come from 20% of the causes. To prevent the number of variants being too high, the distribution will be done evenly from 80%, 60%, 40%, and 20% for each training and testing data. This is done because if the testing data is too little then the performance statistics will have a larger variance, as well as if the training data is too little then the parameter estimates will have a larger variance. Then predictions will be made with the provided algorithm to compare the prediction results which produce the best results for the data. The comparison of algorithms will be carried out using the value of the Precision, Recall, F1-Score, Accuracy. Also some values that are considered to support the results of the comparison.

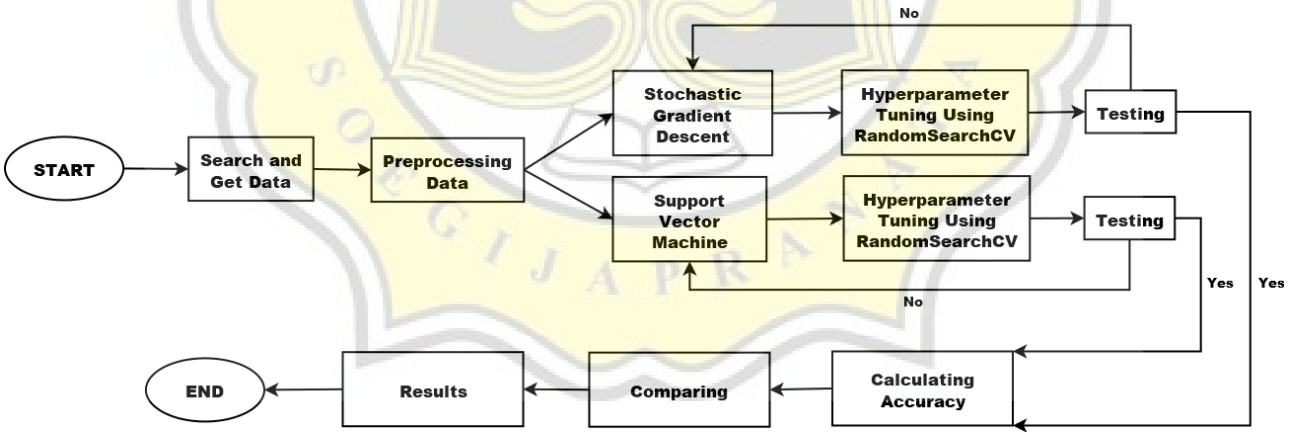

Figure 3.1 Flowchart

3.2. Comparison

The process of comparing the two algorithms is carried out in the same way, as well as tuning the hyperparameters. The first thing to do is process the data first. The data is cleaned of all types of data that are considered unprocessable, such as data in the form of letters, symbols such as semicolons and so on, as well as empty data or commonly known as null data. Then the data will be divided into training data and test data. Training data is data that will later be used to train the algorithm in finding a suitable model, while test data is data that will be used to test and find out the performance of the model obtained at the testing stage. After all the data is divided into predetermined ratios, then the algorithm comparison process is carried out.

The SVM algorithm works by separating the given data set in the best possible way. The distance between any of the closest points is known as the margin. The goal is to select a hyperplane with the maximum possible margin between support vectors in a given dataset. SVM looks for the maximum hyperplane margin in the following steps:

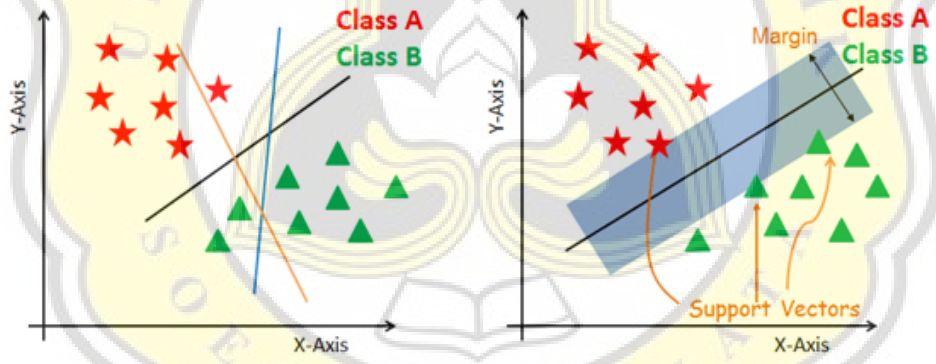

Figure 3.1 SVM Process

Generate a hyperplane that separates classes in the best possible way. The left side image shows three black, blue and orange hyperplanes. Here, blue and orange have a higher misclassification, but black properly separates the two classes. Select the right hyperplane with maximum segregation from one of the closest data points as shown in the image on the right. The following are brief steps in carrying out the prediction calculation process with the SVM algorithm:

1. The data is separated into two categories by curves or lines

2. After the transformation, the boundary between the two categories can be defined by the hyperplane
3. The mathematical functions used for transformations are known as kernel functions, such as linear, polynomial, radial basis function (RBF), and Sigmoid functions

Next algorithm is SGD. The general idea of SGD Algorithm is to start with a random point (in the parabola example we start with a random "x") and find a way to update this point with each iteration so that we go down the slope.

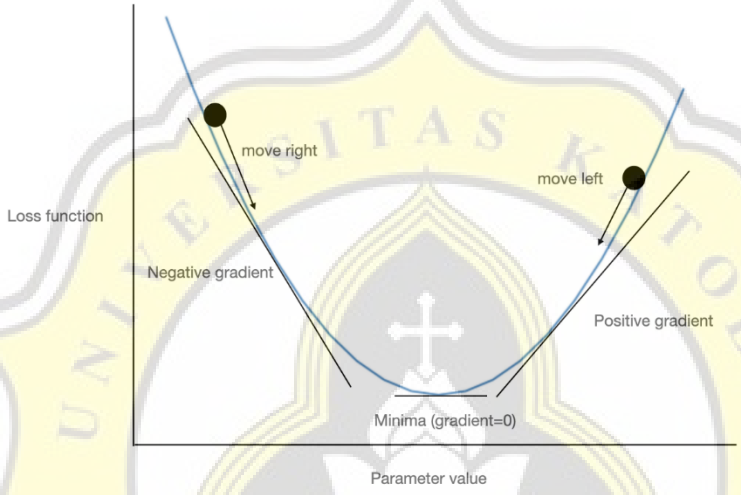

Figure 3.2 SGD Process

The steps of the algorithm are:

1. Find the slope of the objective function with respect to each parameter/feature. In other words, calculate the gradient of the function
2. Choose a random initial value for the parameter. (To clarify, in the parabola example, differentiate "y" with respect to "x". If we have more features like x1, x2 etc., we take a partial derivative of "y" with respect to each feature)
3. Update the gradient function by entering parameter values
4. Calculate the step size for each feature as: step size = gradient * learning rate
5. Calculate new parameters as: new params = old params -step size
6. Repeat steps 3 through 5 until the gradient is almost 0

The "learning rate" mentioned above is a flexible parameter that greatly influences the convergence of the algorithm. The larger learning rate causes the algorithm to take large steps

down the slope and possibly jump over the minimum point thus missing it. So, it's always good to stick to a low learning rate like 0.01. It can also be shown mathematically that the gradient descent algorithm takes larger steps down the slope if the starting point is high above and takes smaller steps as it reaches closer to the destination to be careful not to miss and also be fast enough.