CHAPTER 5
IMPLEMENTATION AND RESULTS

5.1 Implementation

```
1. num_words = 2500
2. tokenizer = Tokenizer(num_words)
3. tokenizer.fit_on_texts(x_train)
4. list_tokenized_train = tokenizer.texts_to_sequences(x_train)
5. max_review_length = 100
6. x_train = pad_sequences(list_tokenized_train, maxlen=max_review_length)
7. model = Sequential()
8. model.add(Embedding(num_words+1, 600, input_length=max_review_length))
9. model.add(LSTM(600))
10. model.add(Dense(1, activation='sigmoid'))
11. model.compile(loss='binary_crossentropy',optimizer=opt,metrics=['accur
    acy'])
12. model.summary()
13. model.fit(x_train,y_train,epochs=10,batch_size=32,validation_split=0.2
    )
14. list_tokenized_test = tokenizer.texts_to_sequences(x_test)
15. x_test = pad_sequences(list_tokenized_test, maxlen=max_review_length)
16. prediction = model.predict(x_test)
17. y_pred = (prediction > 0.5)
18. print(classification_report(y_test, y_pred))
```

The program code lines 1 - 18 above are the core program code for Long Short Term Memory method. Line 1 of the program code contains a command to limit words to a maximum of 2500 words so that the words to be carried out by the tokenizer process are not too long. Lines 2 and 3 contain the Tokenizer command to split the text into a token or word. The 4th line contains the command to create a list of tokenizers that have been created for training data. The 5th line contains the command to limit words that have been tokenized. Line 6 serves to change words that are not the same size to be the same for training data. The 7th line contains the command to create a Sequential LSTM model, which is to change the token to word embedding. The 8th line is the program code that functions to convert words that have become tokenized into vector form, the 8th line also functions as the input layer of the LSTM. The 9th row is the LSTM layer and the neurons that you want to test. The 10th line is the output layer with an activation function that functions to combine all layers, which is usually called a full layer connection. Lines 11 and 12 are codes for optimizing the already created model. The 13th line is the command to test the training data from the model that has been created. The 14th line contains the command to create a list of tokenizers that have been created for data testing. The 15th line

contains a command to change words that are not the same size to be equal for testing data. Lines 16 to 18 contain commands to create predictive models from testing data based on models that have been tested using previous training data.

```
1.  class SVM
2.  def __init__(self, iters= 10, learning_rate= 1, parameter= 1):
3.  self.iterations = iters
4.  self.lr = learning_rate
5.  self.parameter = parameter
6.  self.weight = None
7.  self.bias = None
8.  def fit(self, x, y):
9.  y_ = np.where(y<=0, -1, 1)
10. sample, features = x.shape
11. self.weight = np.random.random_sample(features)
12. self.bias = np.random.random_sample()
13. for _ in range(self.iterations):
14. for index, x_value in enumerate(x):
15. condition = y_[index] * ((x_value @ self.weight) - self.bias) >= 1
16. if condition:
17. classification_weight = 2 * self.parameter * self.weight
18. classifitation_bias = 0
19. self.weight -= self.lr * classification_weight
20. self.bias -= self.lr * classifitation_bias
21. else:
22. y_[index] = y_[index].reshape((1, 1))
23. classification_weight = (2 * self.parameter * self.weight) -
    (x_value.T * y_[index])
24. classifitation_bias = y_[index]
25. self.weight -= self.lr * classification_weight
26. self.bias -= self.lr * classifitation_bias
27. def predict(self, x):
28. linear = x @ self.weight + self.bias
29. sign_matrix = np.sign(linear)
30. result = np.where(sign_matrix<=0, 0, 1).astype(np.float32)
31. return result
32. def accuracy(self, y_pred, y_test):
33. return np.sum(y_pred == y_test)/ len(y_test)
34. x_train, x_test, y_train, y_test = train_test_split(x, y,
    test_size=0.2)
35. y_train = np.array(y_train)
36. y_train = np.expand_dims(y_train, axis=-1)
37. svm = SVM()
38. svm.fit(x_train, y_train)
39. predictions = svm.predict(x_test)
40. score = svm.accuracy(predictions, y_test)
41. score
```

The program code lines 1 - 41 above are the core program code for Support Vector Machine method. Lines 1-7 are program code that contains commands about parameters to be determined based on the model you want to create. Lines 8-10 are code to create a target or parameter y_ in conditions of only numbers 0, -1, and 1. Lines 11 and 12 are used to determine the weight and bias of the parameters that have been created. Lines 13-26 are the command code for building a classification using the SVM model. Lines 27-31 are program codes to generate output or predictions from the SVM model that has been created. Lines 32-33 are command codes to determine the accuracy of testing data from models that have been created and tested using training data. Line 34 is used to split the data into training data and testing data. Lines 35-36 are the program code to add 1 dimension to the y parameter. Lines 37-38 are program codes to test the training data based on the model that has been created. Lines 39-41 are used to make predictions of the testing data based on the training data that has been tested previously.

5.2 Results

5.2.1. Long Short Term Memory Training Results

The first test is to test the number of neurons in the layer amounting to 100 to 900 neurons by using the Sigmoid activation function in the output layer. In this test, the training data is divided and tested 3 times, the training data of 20%, 50% and 80%. It is also repeated for testing each neuron and test data 5 times. This test was conducted to determine the optimal number of neurons in the Sigmoid activation function. These parameters are trained to determine the value of accuracy and loss in the training and validation data. The training process uses 10 epochs and 32 batch sizes. Based on the data obtained from training and validation testing, the highest accuracy value is 0.9994, accuracy loss is 0.0050, validation accuracy is 0.7876, and validation loss is 0.1459 on 600 neuron with 80% Training Data and 20% Testing Data.

Table 5.1. LSTM Train Results

Train Acc	Train Loss	Validation Acc	Validation Loss
0.7951	0.5218	0.8005	0.4818
0.8619	0.3312	0.8057	0.5212
0.9540	0.1233	0.7824	0.6628
0.9812	0.0598	0.7668	0.8519
0.9942	0.0245	0.8083	0.8697
0.9968	0.0107	0.8187	0.0346
0.9961	0.0148	0.7876	0.9093
0.9981	0.0080	0.7746	0.0413
0.9987	0.057	0.7953	0.0966
0.9994	**0.050**	**0.7876**	**0.1459**

5.2.2. Long Short Term Memory Test results

This stage is the stage of testing the model with data testing. The data to be tested amounted to 2411 data with a total of 1940 positive data and 471 negative data. Furthermore, the model will predict reviews in each class to know how accurately the model can predict that class. After the model is predicted, the results obtained from testing the Long short term memory model on 600 neurons and the distribution of test data of 20%. The best results of testing the predicted data neurons 100 to 900, obtained the highest accuracy results of 83.8% accuracy on neurons 600 and obtained results with 88.2% precision, 92.3% recall, and 90.2% F1-Score. It can be seen in **Table 5.2**, that a total of 483 data were tested. With positive original data 391, and negative original data 92. Then the model predicts positive data for 409 data and predicts negative data for 74 data. This means that the Long Short Term Memory method incorrectly predicts 18 positive data and incorrectly predicts 18 negative data.

Table 5.2. LSTM Confusion Matrix Evaluation Results

Actual Label	**Predicted Label**	
	Negative	***Positive***
Negative	44	48
Positive	30	361

Table 5.3. LSTM Test Results

Accuracy	83.8%
Precision	88.2%
Recall	92.3%
F1-Score	90.2%

Table 5.4. LSTM All Test Results

Neuron	Datatest 20%	Datatest 50%	Datatest 80%
100	81.1 %	82.1 %	80.6 %
	80.9 %	79.1 %	80.0 %
	80.7 %	78.3 %	80.1 %
	79.9 %	77.1 %	80.9 %
	82.1 %	81.8 %	80.1 %
200	81.5 %	78.9 %	80.7 %
	81.5 %	81.1 %	79.9 %
	83.4 %	80.8 %	80.3 %
	81.3 %	79.9 %	80.0 %
	80.1 %	79.6 %	79.5 %
300	78.8 %	80.8 %	80.7 %
	81.9 %	81.4 %	80.7 %
	82.1 %	77.2 %	80.0 %
	82.6 %	80.5 %	80.0 %
	83.6 %	80.3 %	80.5 %
400	80.5 %	77.1 %	77.8 %
	80.7 %	78.1 %	80.7 %
	81.5 %	81.3 %	78.6 %
	80.9 %	80.4 %	79.6 %
	83.2 %	81.5 %	78.0 %
	81.5 %	80.5 %	77.3 %
	81.9 %	76.4 %	79.4 %

500	82.4 %	81.0 %	78.3 %
	80.9 %	78.8 %	79.4 %
	81.3 %	82.2 %	78.5 %
600	**83.8 %**	80.3 %	77.7 %
	81.1 %	81.0 %	78.6 %
	80.3 %	82.3 %	79.7 %
	79.5 %	80.5 %	80.4 %
	82.4 %	79.9 %	79.3 %
700	81.9 %	81.0 %	75.8 %
	83.0 %	81.2 %	80.1 %
	80.5 %	81.0 %	79.4 %
	77.8 %	79.5 %	75.0 %
	81.5 %	80.2 %	72.6 %
800	80.9 %	83.4 %	79.5 %
	81.1 %	78.1 %	76.8 %
	81.1 %	81.2 %	76.0 %
	80.5 %	79.1 %	79.7 %
	79.9 %	78.6 %	77.5 %
900	79.7 %	80.5 %	80.3 %
	81.3 %	81.0 %	77.6 %
	79.7 %	79.5 %	72.3 %
	76.8 %	81.7 %	72.9 %
	81.5 %	82.0 %	80.6 %

5.3 Support Vector Machine Results

In the tests carried out for the Support Vector Machine. The first test is to change numpy which has been converted from a data frame into a vectorizer. After converting numpy to vectorizer, a Support Vector Machine model with Scratch is created and run. Then the test data was divided and tested 3 times, test data of 20%, 50%, and 80%. After split the test data, the Support Vector Machine algorithm is tested to get the results of the test. The test is repeated 10 times for each test data using 1 lambda parameters ($\lambda = 1$). Based on the test by dividing 3 data test, the best prediction results were obtained, namely the distribution of 20% test data with an accuracy of 71.0%, the precision of 83.9%, recall of 79.6% and F1-Score 81.7%. It can be seen in **Table 5.5**, that a total of 483 data were tested. With positive original data 393, and negative original data 90. Then the model predicts positive data of 373 data and predicts negative data of 110 data. This means that the Support Vector Machine method incorrectly predicts positive data by 20 data, and incorrectly predicts negative data by 20 data.

Table 5.5. SVM Confusion Matrix Evaluation Results

Actual Label	**Predicted Label**	
	Negative	***Positive***
Negative	30	60
Positive	80	313

Table 5.6. SVM Test Results

Accuracy	71.0%
Precision	83.9%
Recall	79.6%
F1-Score	81.7%

Table 5.7. SVM All Test Results

Parameter	Datatest 20%	Datatest 50%	Datatest 80%
λ = 1	70 %	68.3 %	67.3 %
	67.4 %	65.9 %	66.8 %
	65.4 %	65.4 %	66.9 %
	66.2 %	66.9 %	66.9 %
	71 %	66.6 %	67.3 %
	64.3 %	66.5 %	66.3 %
	69.1 %	65.3 %	66.9 %
	68.7 %	68.9 %	67.4 %
	67.9 %	67.9 %	67.2 %
	66.2 %	65 %	66.6 %

5.4 Support Vector Machine Result With Undersampling Method

In this study, the Support Vector Machine model was tested with balanced data. Data balancing is done by the undersampling method. An undersampling method is a method to reduce the number of datasets from the majority class so that the amount of data in the majority class is close to the minority class. This research uses a data sharing of 20% of test data and uses a linear kernel. The test was repeated 10 times and the highest accuracy was 94%, 91% precision, 99% recall, and 95% F1-Score. It can be concluded here that the Support Vector Machine model is very good for processing balanced data.

Table 5.8. SVM Undersampling Method Confusion Matrix Evaluation Results

Actual Label	**Predicted Label**	
	Negative	***Positive***
Negative	423	48
Positive	6	465

Table 5.9. SVM Undersampling Method Test Results

Accuracy	94 %
Precision	91 %
Recall	99 %
F1-Score	95 %

Table 5.10. SVM Undersampling Method All Test Results

Kernel	Datatest 20%
Linear	**94 %**
	92 %
	93 %
	93 %
	93 %
	93 %
	91 %
	92 %
	93 %
	91 %

5.5 Support Vector Machine Using Kernel

This time the test was carried out using a different kernel in Support Vector Machine method. In addition to testing using the Linear kernel, other kernels tested are the Polynomial, Radial Basis Function (RBF) and Sigmoid kernels. Testing each kernel uses a 20% test data and the test is repeated 5 times.

In the Polynomial kernel, the highest accuracy is 84%, precision is 84%, recall is 100% and F1-Score is 91%. Meanwhile, in the Radial Basis Function (RBF) kernel, the highest accuracy is 83%, precision is 83%, recall is 100% and F1-Score is 91%. The last test is using the Sigmoid kernel, the highest accuracy is 84%, precision is 85%, recall is 98% and F1-Score is 91%

Table 5.11. SVM Kernel All Test Result

Polynomial	**RBF**	**Sigmoid**
84 %	80 %	**84 %**
81 %	81 %	83 %
83 %	**83 %**	82 %
80 %	82 %	82 %
82 %	82 %	83 %

Table 5.12. SVM Polynomial Kernel Confussion Matrix Evaluation Results

Actual Label	**Predicted Label**	
	Negative	***Positive***
Negative	5	78
Positive	0	400

Table 5.13. SVM Polynomial Kernel Test Results

Accuracy	84 %
Precision	84 %
Recall	100 %
F1-Score	91 %

Table 5.14. SVM RBF Kernel Confussion Matrix Evaluation Results

Actual Label	**Predicted Label**	
	Negative	***Positive***
Negative	7	80
Positive	0	396

Table 5.15. SVM RBF Kernel Test Results

Accuracy	83 %
Precision	83 %
Recall	100 %
F1-Score	91 %

Table 5.16. SVM Sigmoid Kernel Confussion Matrix Evaluation Results

Actual Label	**Predicted Label**	
	Negative	***Positive***
Negative	16	71
Positive	6	390

Table 5.17. SVM Sigmoid Kernel Test Results

Accuracy	84 %
Precision	85 %
Recall	98 %
F1-Score	91 %