

CHAPTER 5

IMPLEMENTATION AND RESULTS

5.1. Overview

In the previous chapter, we discussed the analysis and design used in this research. Now, the research breaks into two parts of this chapter. The first part is implementation, how the code in this algorithm from dataset preprocessing until the model or algorithm code. In the next part is the result from the Adaboost (Adaptive Boosting) algorithm, the score and performance we get with the semi-supervised learning. It is compared with the Adaboost (Adaptive Boosting) algorithm in supervised learning. For the addition, we used comparison between the learning with SVD (Singular Value Decomposition) and without SVD (Singular Value Decomposition).

5.2. Implementation

In this implementation explained about how the code in this algorithm from the dataset and the model or algorithm code. The first is about dataset, dataset will split into preprocessing data and data processing. The second is about algorithm code, how Adaboost (Adaptive Boosting) algorithm scratches from feature extraction.

Now, the research talks about dataset preprocessing data. In this preprocessing data, we start from checking the dataset especially in the “Content” column and “Annotation” column. It needs to be checked to match or not for each other.

```
1. data.info()
```

From the line 1, give how many row which used in the dataset. It is same or not, if not we need to checked again. And the output shows as below.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 364 entries, 0 to 363
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Url         364 non-null   object
1   Content     364 non-null   object
2   Title       364 non-null   object
3   Annotation  364 non-null   object
dtypes: object(4)
memory usage: 11.5+ KB
```

Figure 5.1 Dataset Info

Figure 5.2.1 shows how many rows in each column are non-null. Non-null in this case is not empty and not NaN. The data shows that there are the same number of each column which is 364 rows not null. The type of each column is an object. The object is a form of a collection of data. It has alternatives code for the specific column shown in the code below

```
1. data['Annotation'].count()
```

The code above shows how many rows are in the “Annotation” column. It can be replaced with other columns like “Content” or “Url”. If there are many rows which are null means that NaN or maybe empty it can be dropped or filled with the majority class.

In the “Content” column, it’s very dirty because of the stopwords, news headlines like “Jakarta – Sindonews” that affected the data training. So, in this process it cleaned by this code below

```
1. def clean_text(news):
2.     news = re.sub(r'^\w+[\^~]*[ -] ?\|?', ' ', news)
3.     news = re.sub(r'https?:\/\/\S+|www\.\S+', ' ', news)
4.     news = re.sub(r'<.*?>', ' ', news)
5.     news = re.sub(r'\d+', ' ', news)
6.     news = re.sub(r'@\w+', ' ', news)
7.     news = re.sub(r'[\^~\s\d]', ' ', news)
8.     news = re.sub(r'\s+', ' ', news).strip()
9.     news = " ".join([word for word in str(news).split() if word not
10.                     in stops])
11.     return news.lower()
```

In the code above, it is the core of the preprocessing data because it can very affect the prediction. Line 1 means we defined a function named clean_text with parameters news. In line 2, we used a library called Regex (Regular Expression) to remove sources of the news like “Jakarta –”. Re.sub means to replace a substring with another substring. Regex will filter that beginning with a word that shows in the symbol ^w and + means which token is matched at least once and more. And then [\^~] means that match in any character that is not in the set. * means that matches 0 or more tokens with that pattern. [-] means that it matches any character on the set. In this case, space and character “-”. The next symbol ?\|? means that after this pattern is optional for spaces and escaped characters after that character. The goal of Line 3 is to remove links in news content. Line 4 is to remove html sentences. Line 5 to remove any number in the news content. Line 6 to remove any sentences that begin with the symbol “@”. Line 7 is to remove punctuation like “!”, “.”, “?”, “!”. Line 8 to remove whitespace. Line 9 to filter stopwords if there are no

stopwords the words will skip and rearrange in a sentence. Line 10 is to return sentences into lowercase. Now, we will discuss stemming. The code for stemming is shown below

```
1. factory = StemmerFactory()
2. stemmer = factory.create_stemmer()
3. data['stemText'] = data['CleanText'].apply(lambda x:
    stemmer.stem(x))
```

In the code above, it is the stemming process. In this process, we used the stemmer library from Sastrawi library. It is very good for the stemming process. Line 1 is used to define a factory for stemmers. Line 2 is used to create stemmers. Line 3 converts each row into stem sentences form and input into a dataframe column named “stemText”. It takes at least 20 minutes for 365 data. After preprocessing, we can analyze data by using word count in each class and detect outliers using boxplot.

```
1. def word_freq(clean_text_list, top_n):
2.     flat = [item for sublist in clean_text_list for item in
    sublist]
3.     with_counts = Counter(flat)
4.     top = with_counts.most_common(top_n)
5.     word = [each[0] for each in top]
6.     num = [each[1] for each in top]
```

In the code above, it is explained one by one. The goal of that code is to rank the dominant word in the dataset.

```
1. cl_text_list = data['CleanSplitText'].tolist()
2. wf = word_freq(cl_text_list, 20)
3. wf.head(20)
```

For line 1 is to make a dataset in column “CleanSplitText” to list. It was done for the word_freq function. In line 2, it called a function named word_freq and sent a parameter list, and the number of top words. In line 3, it shows the top 20 of the word. The output of this code above is the picture below.

| | 0 | 1 | | | |
|--|---|-----------|-----|----|------------|
| | 0 | tahun | 719 | 10 | akhir 280 |
| | 1 | jadi | 594 | 11 | laku 257 |
| | 2 | main | 541 | 12 | baik 254 |
| | 3 | kata | 489 | 13 | menang 252 |
| | 4 | sebut | 458 | 14 | tak 250 |
| | 5 | baru | 368 | 15 | dua 246 |
| | 6 | lebih | 335 | 16 | partai 238 |
| | 7 | besar | 316 | 17 | satu 233 |
| | 8 | indonesia | 307 | 18 | rp 228 |
| | 9 | persen | 290 | 19 | hari 227 |

Figure 5.2 The Top 20 Word Count of Whole Data

Figure 5.2 we can know the top 20 words. We can also make a word count from each class. It is not too much change from a whole document. Besides a word count, it used boxplot to detect outliers. Outliers are detected by boxplot for the code shown below.

1. `dfconcat = pd.concat([dfconcat], axis=0, ignore_index=True)`
2. `dfconcat.boxplot(by = 'Category', column = ['count'], grid = False)`

The first line means we concat data on each class in one other dataframe. In Line 2, we used a boxplot with a group by “Category” column to get from “Count” column, `grid = False` means show boxplot with grid background or not. It shows boxplot with the picture below

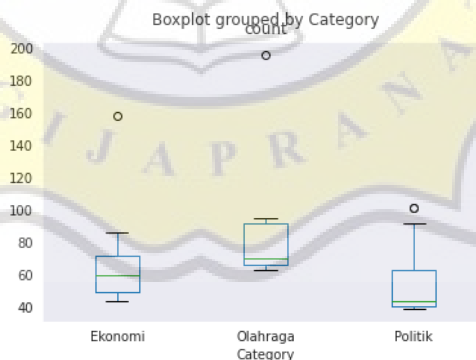


Figure 5.3 Boxplot to Detect Outliers

Figure 5.2.3, the data have an outlier. It shows a circle that is far away from the Quartile 1 and far away from the max value on each class. It is not a big problem because in the learning

algorithm we used ensemble learning in boosting approach. It can solve these outliers, especially small outliers.

For the data processing, we used a library for feature extraction. In this feature extraction, we used TF-IDF(Term of Frequency Inverse Document Frequency) and SVD (Single Value Decomposition) to reduce dimension in the matrix. The code below shows how to get the TF-IDF.

```
1. texts = df['CleanSplitText'].astype('str')
2. tfidf_vectorizer = TfidfVectorizer(ngram_range=(1, 2),
3.                                   min_df = 2,
4.                                   max_df = .95)
```

For line 1 is to change the data type from series to string. Line 2 defines TF-IDF from tensorflow with the parameter. The parameter used n-gram, min_df, and max_df. N-gram means how many words will be used in the research. Min_df means a minimum document frequency will be used for the processing data. Max_df means a maximum document frequency will be used for the processing data. The SVD (Singular Decomposition Value) is shown by the code below.

```
1. lsa = TruncatedSVD(n_components=100,
2.                    n_iter=10)
```

For line 1 we defined the SVD (Singular Decomposition Value) algorithm to reduce dimensionality in a sparse matrix. N_components is Desired dimensionality of output data. It is less than the number of features. N_iter is the number of iterations for a randomized SVD solver.

After preprocessing and processing data, now we hold on modelling. In modelling we used the Adaboost (Adaptive Boosting) algorithm. In this algorithm, we split into two learning. The first is supervised learning and the second is semi-supervised learning. Supervised learning starts with splitting the data into train and test data. The splitting process is shown by the code below

```
1. X_train, X_test, y_train, y_test = train_test_split(xdata, ydata,
test_size=0.1)
```

For line 1 means divide the data into two parts: train and test in each divide into X and Y. X split into X_train and x_test. Y split into y_train and y_test. Train_test_split is a function from ScikitLearn that can split into data train and data test. It works with parameter data x and y, and the test_size is how many percent is split between train and test. Now, let us see how supervised learning is with Adaboost (Adaptive Boosting) code below.

```
1. clf = AdaBoostClassifier(n_estimators=5)
```

For line 1 it is called an Adaboost class with n_estimator 5. N_estimators means how maximum stumps will be created in that model. Stumps is a tree with only one branch. The function is explained in the code below.

```
1. class AdaBoostClassifier:
2.
3.     def __init__(self,base_estimator=None,n_estimators=50):
4.         self.n_estimators = n_estimators
5.         self.models = [None]*n_estimators
6.         if base_estimator == None:
7.             base_estimator = DecisionTreeClassifier(max_depth=1)
8.         self.base_estimator = base_estimator
9.         self.estimator_errors_ = []
```

For line 1 means define a class named AdaBoostClassifier. Line 3 defines a constructor. So, if the object is formed, the default constructor will be applied. Line 4 defines n_estimators. Line 5 defines a model that has N times n_estimators. Line 6 detects whether there is a base estimator or not. Base estimator is where the boosted ensemble is built. Line 7 shows the default used decision tree classifiers but the maximum depth of tree is only 1. Line 8 defines base_estimator for the object. Line 9 defines an empty array to collect classification error for each estimator in the boosted ensemble.

```
1. def fit(self,X,y):
2.     X = np.float64(X)
3.     N = len(y)
4.     w = np.array([1/N for i in range(N)])
5.     self.createLabelDict(np.unique(y))
6.     k = len(self.classes)
```

For line 1 means define a class named fit. The goal of this function is to train data and start boosting methods. Line 2 is collected from X train to variable X. line 3 is to count the number of rows which will train. Line 4 is calculated weight for each data. Line 5 is to create a unique dictionary that converts from label to number. Line 6 counts how many classes in the dataset. The code below is to iterate and make the stumps.

```
1. for m in range(self.n_estimators):
2.     Gm = base_estimator.clone(self.base_estimator).\
3.         fit(X,y,sample_weight=w).predict
4.     incorrect = Gm(X) != y
5.     errM = np.average(incorrect,weights=w,axis=0)
6.     self.estimator_errors_.append(errM)
7.     BetaM = (np.log((1-errM)/errM)+np.log(k-1))
8.     w *= np.exp(BetaM*incorrect*(w > 0))
```

For line 1 means define how many iterations will be created. Line 2 is to construct a new unfitted estimator with the same parameters and fit with the function fit and make predictions that

are all collected in the Gm variable. Line 4 is collected from Gm where it is incorrectly predicted. Line 5 is the calculated error rate. Line 6 is appended to estimator_errors from error rate. Line 7 is calculated betta value or in the theory called alpha value with that formula. Line 8 is calculated as a new weight. The other code of this implementation will be show as below

```
1. def createLabelDict(self, classes):
2.     self.labelDict = {}
3.     self.classes = classes
4.     for i, cl in enumerate(classes):
5.         self.labelDict[cl] = i
```

For line 1 means define a function to create a label dictionary with parameter classes. Line 2 is a defined set for label dict. Line 3 is to get classes from numpy unique from data y. line 4 is to iterate the index and name of classes. Line 5 is define Label dictionary as get index insert into labelDict.

```
1. def predict(self, X):
2.     k = len(self.classes)
3.     y_pred = sum(Bm*indexToVector(Gm(X), k, self.labelDict) \
4.                 for Bm, Gm in self.models)
5.     iTL = np.vectorize(indexToLabel)
6.     return iTL(np.argmax(y_pred, axis=1), self)
```

For line 1 means define a function named predict. Line 2 is the number of classes. Line 3 is get y_pred from the result of sum from Bm times to index to vector from X. Bm gets from iteration in the model per iteration in the fit function. Line 5 is a converter from list to vector from the result of the function indexToLabel. Line 6 is the return function into the specific class.

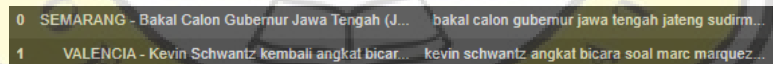


Figure 5.4 Dataset for Real scenario

The **figure 5.4** shows dataset will be used for real scenario and the output will be in the **figure 5.5** below.

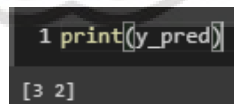


Figure 5.5 The output from prediction

5.3. Results

The result of this evaluation method is divided into four variations. The first is an imbalanced class with supervised learning. The second is an imbalanced class with semi-

supervised learning. It does after supervised learning. The third is a balanced class with supervised learning. The fourth is a balanced class with supervised learning. They are using dataset proportions 90:10, 90% for training set and validation set in supervised learning and 10% for making predictions in semi-supervised learning. So, the total data that is used for this model is 328 rows. It split into 25:75, 50:50, and 75:25. It can be shown in the table below.



Table 5.1 Imbalanced Class Supervised Learning Table

| | | Train : Test | | |
|------------------|------------------|---------------------|--------------|--------------|
| Parameter | | 25:75 | 50:50 | 75:25 |
| | Accuracy | 82.02 | 92.22 | 92.72 |
| Macro | Precision | 84.87 | 91.57 | 93.06 |
| | Recall | 81.83 | 90.93 | 91.15 |
| | F1-Score | 79.88 | 90.13 | 91.19 |
| Micro | Precision | 84.52 | 92.22 | 92.73 |
| | Recall | 82.08 | 92.22 | 92.73 |
| | F1-Score | 83.00 | 92.22 | 92.73 |

From the **Table 5.1**, we know that the method of that learning is supervised learning. The performance is shown by parameters from this method with datasets from different ratios. The parameters also divide into macro and micro averaged. The first is 25:75, the second is 50:50, and 75:25. With these ratios 25:75, the model gets 82.02 % accuracy. For the macro-averaged, precision gets 84.87%, recall gets 81.83%, and F1-Score gets 79.88%. For the micro-averaged, precision gets 84.52%, recall gets 82.08%, and F1-Score gets 83%. In macro-averaged precision gets 84.87% higher than micro-averaged precision which has only 84.52%. In micro averaged, precision of each class calculated and added, then divided by the number of classes. Precision intuitively the performance only predicts the data really positively. From the formula it is inversely proportional with the false negative that the actual data is right and the prediction is wrong. It can be said that less false positives give higher precision. So, in this case that is not too many false negatives.

If we can see, in micro-averaged there are many with the same value especially precision, recall, and F1-Score. It happens because precision and recall are the same and F1-Score gets the mean between them. For example, it is explained in the table below.

Table 5.2 Micro-averaged with Same Value

| | | | | | | | | | | |
|------------|---|---|---|---|---|---|---|---|---|---|
| Label | 0 | 2 | 1 | 1 | 2 | 0 | 1 | 0 | 1 | 1 |
| Prediction | 1 | 2 | 1 | 0 | 0 | 1 | 1 | 0 | 2 | 2 |

From the **Table 5.2**, we can see that there are 2 columns labeled and predicted. Label represents the class or categorize for the news and prediction represents the model that predicts the data. From this table, we can also calculate the result from this formula below.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Figure 5.5 Precision Formula

$$\text{Recall} = \frac{TP}{TP + FN}$$

Figure 5.6 Recall Formula

In the **Figure 5.5** is the precision formula. In the **Figure 5.6** is the recall formula. TP is the amount of samples that were predicted with the correct label. In this case, the number of TP is 4. It is shown by all the green cells. While FP is the number of actual is false and predicted is true. In this case, the number of FP is 6. It is shown with the red cells. For the first red cells the 0 should predict 0, but 1 was predicted. It can be said that is false positive for 1 class. Otherwise, if the second column is 2 and the predicted is 2. It is true positive and there is no FP count. FN is the number of while actual is true and the predicted is false. In this case, the number of FN is 6. It is shown with the red cells. For the first red cells the 0 should predict 0, but 1 was predicted. It can be said that there are false negatives for class 0. Otherwise, if the second column is 2 and the predicted is 2. It is true negatives and there is no FN count. From the example above, there are many possibilities with the same value, especially precision, recall, and F1-Score in micro-averaged because of that.

With this ratio of 50:50, the model gets 92.22 % accuracy. For the macro-averaged, precision gets 91.57%, recall gets 90.93%, and F1-Score gets 90.13 %. For the micro-averaged, precision, recall, and F1-Score get 92.22%.

With this ratio of 75:25, the model gets 92.72 % accuracy. For the macro-averaged, precision gets 93.06%, recall gets 91.15%, and F1-Score gets 91.19 %. For the micro-averaged, precision, recall, and F1-Score get 92.73%.

It can be seen that if the training set is added, the performance of each parameter will increase too. It also concluded that the model gets a lot of training data. The learning performance will be increased as add training data in the model.

Table 5.3 Imbalanced Class Semi-Supervised Learning Table

| Parameter | | Train : Test | | |
|-----------|-----------|--------------|-------|-------|
| | | 25:75 | 50:50 | 75:25 |
| Macro | Accuracy | 94.56 | 94.89 | 92.33 |
| | Precision | 94.78 | 95.67 | 92.80 |
| | Recall | 93.23 | 94.1 | 91.82 |
| | F1-Score | 93.18 | 94.43 | 91.77 |
| Micro | Precision | 94.56 | 94.89 | 92.33 |
| | Recall | 94.56 | 94.89 | 92.33 |
| | F1-Score | 94.56 | 94.89 | 92.33 |

From the **Table 5.3**, we know that the method of that learning is semi-supervised learning. The performance is shown by parameters from this method with datasets from different ratios. The parameters also divide into macro and micro averaged. The first is 25:75, the second is 50:50, and 75:25.

With these ratios 25:75, the model gets 94.56 % accuracy. For the macro-averaged, precision gets 94.78%, recall gets 93.23%, and F1-Score gets 93.18%. For the micro-averaged, precision gets 94.56%, recall gets 94.56%, and F1-Score gets 94.56%. In macro-averaged precision it is 94.78% smaller than micro-averaged precision which has only 94.56%.

With this ratio of 50:50, the model gets 94.89 % accuracy. For the macro-averaged, precision gets 95.67%, recall gets 94.10%, and F1-Score gets 94.43 %. For the micro-averaged, precision, recall, and F1-Score get 94.89%.

With this ratio of 75:25, the model gets 92.33 % accuracy. For the macro-averaged, precision gets 92.80%, recall gets 91.82%, and F1-Score gets 91.77 %. For the micro-averaged, precision, recall, and F1-Score get 92.33%.

It can be seen that if the training set is added in supervised learning, the performance of each parameter in semi-supervised learning will increase too. It also concluded that the model gets a lot of training data. The learning performance will be increased as add training data in the model. Adding a semi-supervised learning model with pseudo labelling, it can further improve the accuracy and precision recall.

Table 5.4 Balanced Class Supervised Learning Table

| | | Train : Test | | |
|------------------|------------------|---------------------|--------------|--------------|
| Parameter | | 25:75 | 50:50 | 75:25 |
| | Accuracy | 92.22 | 88.61 | 91.11 |
| Macro | Precision | 91.57 | 80.66 | 82.48 |
| | Recall | 91.80 | 83.12 | 84.17 |
| | F1-Score | 90.24 | 80.72 | 82.55 |
| Micro | Precision | 92.78 | 88.86 | 91.99 |
| | Recall | 92.22 | 88.61 | 91.11 |
| | F1-Score | 92.47 | 88.73 | 91.53 |

From the table 5.3.4 , we know that the method of that learning is supervised learning. But, in this case, we used a balanced dataset in each class. The performance is shown by parameters from this method with datasets from different ratios. The parameters also divide into macro and micro averaged. The first is 25:75, the second is 50:50, and 75:25.

With these ratios 25:75, the model gets 92.22 % accuracy. For the macro-averaged, precision gets 91.57%, recall gets 91.80%, and F1-Score gets 90.24%. For the micro-averaged, precision gets 92.78%, recall gets 92.22%, and F1-Score gets 92.47%. In macro-averaged precision it is 91.57% close to micro-averaged precision which is 92.78%.

With this ratio of 50:50, the model gets 88.61 % accuracy. For the macro-averaged, precision gets 80.66%, recall gets 83.12%, and F1-Score gets 80.72 %. For the micro-averaged, precision gets 88.86% , recall gets 88.61%, and F1-Score gets 88.73%.

With this ratio of 75:25, the model gets 91.11 % accuracy. For the macro-averaged, precision gets 82.48%, recall gets 84.17%, and F1-Score gets 82.55 %. For the micro-averaged, precision gets 91.99%, recall gets 91.11%, and F1-Score gets 91.53%.

```
Ekonomi      80
Politik      80
Olahraga     80
Name: Annotation, dtype: int64
```

Figure 5.2.3 Balanced Class with 80 Rows of Each Class

From the description above, we know that using a balanced dataset, supervised learning with Adaboost can get the result closer to each other from macro and micro. We know that they have a closer score for each other. It happens because the data for each class is balanced, which amounts to 80 rows. It can be smaller because the data have manually undersampling that reduces the majority of the class adjusted to the minority class. The picture below shows the balanced class with 80 rows of each class.

Table 5.5 Balanced Class Semi Supervised Learning Table

| | | Train : Test | | |
|-----------|-----------|--------------|-------|-------|
| Parameter | | 25:75 | 50:50 | 75:25 |
| Macro | Accuracy | 95.69 | 94.58 | 91.25 |
| | Precision | 87.38 | 83.69 | 74.84 |
| | Recall | 88.34 | 85.87 | 77.58 |
| | F1-Score | 87.55 | 84.3 | 75.65 |
| Micro | Precision | 95.69 | 94.58 | 91.38 |
| | Recall | 95.69 | 94.58 | 91.25 |
| | F1-Score | 95.69 | 94.58 | 91.32 |

From the **Table 5.5**, we know that the method of that learning is semi-supervised learning. But, in this case, we used a balanced dataset in each class. The performance is shown by parameters from this method with datasets from different ratios. The parameters also divide into macro and micro averaged. The first is 25:75, the second is 50:50, and 75:25.

With these ratios 25:75, the model gets 95.69 % accuracy. For the macro-averaged, precision gets 87.38 %, recall gets 88.34 %, and F1-Score gets 87.55%. For the micro-averaged, precision, recall, F1-Scores get the equal number 95.69 %.

With this ratio of 50:50, the model gets 94.58 % accuracy. For the macro-averaged, precision gets 83.69 %, recall gets 85.87 %, and F1-Score gets 84.3 %. For the micro-averaged, precision, recall, and F1-Score get the equal number 94.58 %.

With this ratio of 75:25, the model gets 91.25 % accuracy. For the macro-averaged, precision gets 74.84 %, recall gets 77.58 %, and F1-Score gets 75.65 %. For the micro-averaged, precision gets 91.38 %, recall gets 91.25 %, and F1-Score gets 91.32 %.

As we can also see, the model for each macro-averaged not performed well. It happens because the data is very small. It is just 80 rows compared with imbalanced data which is bigger than a balanced dataset. It affects the learning data process that is not enough, if it is just only 240 rows. So, the learning model is not very good. Macro-averaged and micro-averaged have quite a difference. It happens because balanced data only use micro-averaged opposition with imbalanced data we only focus on macro-averaged. If we look on micro-averaged, the model is very good because it can improve the performance of learning.

Table 5.6 Imbalanced Class Supervised Learning 780 rows without SVD Algorithm

| | | Train : Test | | |
|-----------|-----------|--------------|-------|-------|
| Parameter | | 25:75 | 50:50 | 75:25 |
| | Accuracy | 80.42 | 86.22 | 91.82 |
| Macro | Precision | 78.24 | 88.06 | 91.41 |
| | Recall | 77.69 | 85.21 | 91.75 |
| | F1-Score | 74.40 | 84.38 | 90.75 |

| | | | | |
|-------|-----------|-------|-------|-------|
| Micro | Precision | 80.77 | 86.22 | 91.82 |
| | Recall | 80.42 | 86.22 | 91.82 |
| | F1-Score | 80.58 | 86.22 | 91.82 |

In the **Table 5.6** , we see that without the SVD algorithm, the performance of that machine learning decreased. It can be seen from each ratio of the dataset that shown decreased performance compared with SVD Algorithm. It can be explained that SVD algorithms affect the performance of machine learning, because the dimension of sparse matrices can be reduced. If there is no SVD algorithm, the dimension matrices can cause bias for the classification. It happens because SVD works with R and S values. If the R value is lower the SVD can be optimized, but it also depends on S value. If the S value is good enough, and the R value will lower, the data will be noisy. But, if the S value is good enough and the R value is also good enough, the data will be optimized.

Table 5.7 Imbalanced Class with 780 Rows Data Supervised Learning Table with SVD Algorithm

| | | Train : Test | | |
|-----------|-----------|--------------|-------|-------|
| Parameter | | 25:75 | 50:50 | 75:25 |
| | Accuracy | 97.33 | 95.64 | 96.84 |
| Macro | Precision | 96.67 | 94.55 | 96.03 |
| | Recall | 96.55 | 94.79 | 97.35 |

| | | | | |
|--------------|------------------|-------|-------|-------|
| | F1-Score | 95.92 | 94.35 | 96.51 |
| Micro | Precision | 97.33 | 95.64 | 96.84 |
| | Recall | 97.33 | 95.64 | 96.84 |
| | F1-Score | 97.33 | 95.64 | 96.84 |

From the **Table 5.7** , we know that the additional data can affect the result. As shown by the table above, with a ratio 25:75, the performance of machine learning improves well compared to 365 rows of data. The precision and recall is just a little difference, so that it can not be said that is overfitted or underfit. The macro averaged and micro averaged also have a little difference. The multi classification from this case overall is good, but if the training data increases, it has a stagnan result. From this result, with the same number of data with 365 rows of data it also represented the result from the division of the dataset with 780 rows of data.

Table 5.8 Imbalanced Class with 780 Rows Data Semi-Supervised Learning Table with SVD Algorithm

| | | Train : Test | | |
|-----------|-----------|--------------|-------|-------|
| Parameter | | 25:75 | 50:50 | 75:25 |
| | Accuracy | 99.40 | 97.31 | 96.88 |
| Macro | Precision | 99.31 | 96.45 | 96.30 |
| | Recall | 99.25 | 97.35 | 97.01 |

| | | | | |
|-------|-----------|-------|-------|-------|
| | F1-Score | 99.26 | 96.79 | 96.53 |
| Micro | Precision | 99.40 | 97.31 | 96.88 |
| | Recall | 99.40 | 97.31 | 96.88 |
| | F1-Score | 99.40 | 97.31 | 96.88 |

Supervised learning affects semi-supervised learning from the number of dataset. From the **Table 5.8**, the result of semi-supervised learning can increase the performance of supervised learning. Overall, with the same ratio, it can increase the performance.

Semi-supervised learning also can improve the accuracy. By the data from supervised learning, we can see the precision, recall, and F1-Score with the same ratio in semi-supervised learning and the performance improved well. If the data varied with the same learning (supervised or semi-supervised learning) the performance of learning also increased. Although, there is the result that is not increased, but it is just 1-5% only a small difference between them.

With the addition of data, it also performs with a little difference in performance of measure. Natural Language Processing (NLP) has many features in each row of the dataset. So, with the additional data, it may add the feature but not all the feature is tested with the test data. In this case there is a possibility that all of the features can not be tested in dataset.

From the **Table 5.1** and **Table 5.4** we know that imbalanced dataset with the same number of data has difference result. As we know that, ensemble learning can solve imbalanced dataset. From the algorithm we used adaboost algorithm which is one of the ensemble learning method (boosting). We can see that, the result of imbalanced dataset is better than balanced dataset. It can be said that adaboost algorithm work well in imbalanced dataset that is boost the machine learning performance.