CHAPTER 5
IMPLEMENTATION AND RESULTS

5.1. Implementation

5.1.1. Dataset preprocessing

1. Removing symbols

```
1. removesymbols = re.sub(r"[^a-zA-Z ]", '', str(word))
```

All of each reviews from main and prediction dataset in word variable are removing symbols, numbers, emoticons and other unusual symbols led to leaving only uppercase, lowercase, and the space of it on removesymbols variable.

2. Lowercasing

```
2. lowercase = removesymbols.lower()
```

All of each reviews after removing symbols are lowered through of using ".lower()" on removesymbols variable to lowercase variable into all of it are lowercase on lowercase variable

3. Tokenization

```
3. tokenized = lowercase.split()
```

After lowercasing, each of words on reviews are changed into array through of using ".split()" on lowercase variable to tokenized variable for ease the next process. But it retokenize again after stopwords removal and lemmatization process like in the code below.

```
4. stwords = stwords.split()
```

Retokenized after stopwords removal using ".split()" on stwords variable to stwords variable.

```
5. lemmatized = lemmatized.split()
```

Retokenized after lemmatization using ".split()" on lemmatized variable to lemmatized variable.

4. Stopwords

```
6. exception                                                    =
   ['no','not','t','can',"can't","cant",'don',"don't",'ain','arent',"aren'
   t",'couldn',"couldn't",'didn',"didn't",'doesn',"doesn't",'hadn',"hadn't
   ",'hasn',"hasn't",'haven',"haven't",'isn',"isn't",'mightn',"mightn't",'
   mustn',"mustn't",'needn',"needn't",'shouldn',"shouldn't",'wasn',"wasn't
   ",'weren',"weren't",'won',"won't",'wouldn',"wouldn't"]
7. stopwords = set(stopwords.words('english')).difference(exception)
```

Stopwords are set into English language with of exception variable such as words in the code above to not removing certain words that should be removed by stopwords. The stopword are set into stopwords variable along with difference on exception variable.

```
8. n_stop = 0
9. for w in tokenized:
10.
11.     if n_stop == 0 :
12.         if w in stopwords :
13.             stw = ''
14.             stwords = (stw)
15.
16.         else :
17.             stw = w
18.             stwords = (stw + ' ')
19.
20.    else:
21.         if w in stopwords :
22.             stw = ''
23.             stwords = (stwords + stw)
24.
25.         else :
26.             stw = w
27.             stwords = (stwords + stw + ' ')
28.
29.    n_stop = n_stop+1
```

There is a n_stop variable that used to count all words (w variable) from each reviews that begin from 0. The words (w variable) of tokenized review (stopwords variable) are removing stopwords that are in stopwords to inputted on stwords variable that contains unremoved stopwords. If n_stop variable is equal to 0, then if a word (w variable) is removed, stwords only get empty value (from stw variable) and if a word (w variable) is

not removed, stword get a word (from stw variable) and a space for next word. If n_stop variable is more than 0, then if a word (w variable) is removed, stwords only get empty value (from stw variable) to merge with old stwords variable and if a word (w variable) is not removed, stword get a word (from stw variable) to merge with old stwords variable and a space for next word.

5. Lemmatizing

```
30. from nltk.stem import WordNetLemmatizer
31. lemmatizer = WordNetLemmatizer()
```

The lemmatizer variable used to abbreviate the using WordNetLemmatizer language to lemmatize the words in each review after import WordNetLemmatizer from nltk library.

```
32. n_lem = 0
33. for w in stwords:
34.
35.     if n_lem == 0 :
36.         lem= lemmatizer.lemmatize(w)
37.         lemmatized = (lem + ' ')
38.     else:
39.         lem= lemmatizer.lemmatize(w)
40.         lemmatized = (lemmatized + lem + ' ')
41.
42.     n_lem = n_lem+1
```

There is a n_lem variable that used to count all words (w variable) from each reviews that begin from 0. The words (w variable) of each tokenized stopwords review (stwords variable) are changed the certain words, especially plural words into root words (lem variable) transfered to lemmatized variable. If n_lem variable is equal to 0, then either changed and unchanged words (w variable) in lemmatize process using ".lemmatize()" are resulted into lem variable and then transfered to lemmatized variable with a space for next word. If n_lem variable is more than 0, then either changed and unchanged words (w variable) in lemmatize process using ".lemmatize()" are resulted into lem variable and then transfered to lemmatized variable with a space for next word to merge with old lemmatize variable.

6. Stemming

```
43. from nltk.stem.snowball import SnowballStemmer
44. ss = SnowballStemmer(language='english')
```

The ss variable used to abbreviate the using Snowball Stemmer using English language to stem the words in each review after import Snowball Stemmer from nltk library.

```
n_st = 0
for w in lemmatized:

    if n_st == 0 :
        st= ss.stem(w)
        stemmed = (st + ' ')
    else:
        st= ss.stem(w)
        stemmed = (stemmed + st + ' ')

    n_st = n_st+1
```

There is a n_st variable that used to count all words (w variable) from each reviews that begin from 0. The words (w variable) of each tokenized lemmatization review (lemmatized variable) are changed the all of inflected words into root words (st variable) transfered to stemmed variable.

If n_st variable is equal to 0, then either changed and unchanged words (w variable) in stemmed process using ".stem()" are resulted into st variable and then transfered to stemmed variable with a space for next word.

If n_st variable is equal to 0, then either changed and unchanged words (w variable) in stemmed process using ".stem()" are resulted into st variable and then transfered to stemmed variable with a space for next word to merge with old stemmed variable.

5.1.2. Sentiment determiner

```
reviewText = cleaning(dataset["reviewText"][x])
overall = dataset["overall"][x]

if dataset["overall"][x] == 1.0 :
    ov = -1
elif dataset["overall"][x] == 2.0 :
    ov = -0.5
elif dataset["overall"][x] == 3.0 :
    ov = 0
elif dataset["overall"][x] == 4.0 :
    ov = 0.5
else :
    ov = 1.0

blob = TextBlob(reviewText).sentiment.polarity

if dataset["helpfulVotes"].isna()[x] :
    sentiment_mean = float(format((blob+ov)/2, ".3f"))
else :
```

```
75.     sentiment_mean = float(format((blob+(2*ov))/3, ".3f"))
76.
77. if sentiment_mean >0 :
78.     sentiment = "Positive"
79. else :
80.     sentiment = "Negative"
```

After a review text is preprocessed through cleaning function on reviewText column on reviewText variable, then it processed for processing sentiment polarity from TextBlob library which transferred into blob variable. The overall from each review on overall variable is processed into mapped overall by using if else on certain overall which transferred into ov variable. Then it computed into calculating the average of sentiment using blob and ov variable with a if else function to determine whether there are helpful votes in a review which the result formatted into float number and 3 digit number decimal separator to sentiment_mean variable. As mentioned before, if there are no helpful votes on this review, only calculate the average of blob and ov variable but if there are any helpful votes on this review, calculate the average of doubled blob variable and ov variable. If the sentiment_mean variable are resulted into >0, then "Positive" string inputted into sentiment variable, if the sentiment_mean variable are resulted into <=0, then "Negative" string inputted into sentiment variable.

5.1.3. Balancing the dataset

```
81. count_negative = dataset["sentiment"].value_counts()['Negative']
```

All of "Negative" reviews are counted and transferred into count_negative variable.

```
82. dataset_class_negative = dataset[dataset['sentiment'] == "Negative"]
83. dataset_class_positive = dataset[dataset['sentiment'] == "Positive"]
84. undersampling_pos = dataset_class_positive.sample(count_negative)
```

All of reviews with "Positive" sentiment in the sentiment column are transferred into dataset_class_positive and then sampled using ".sample()" with the number of count_negative variable. Then, the "Positive" sentiment number are decreased into the same number of the "Negative" sentiment transferred into undersampling_pos variable. Meanwhile, all of reviews with "Negative" sentiment in the sentiment column are transferred only into dataset_class_negative and there no change in the number of it.

```
85. dataset = pd.concat([undersampling_pos, dataset_class_negative])
```

The combined of undersampling_pos and dataset_class_negative variable using concat function are transferred into dataset.

5.1.4. Train test split

```
86. from sklearn.model_selection import train_test_split
87. X_train,          X_test,          y_train,          y_test          =
    train_test_split(dataset['reviewText'],           dataset['sentiment'],
    test_size=0.1, random_state=42)
```

After import train_test_split from sklearn library, the balanced dataset are splitted into X_train, X_test, y_train, and y_test which use "reviewText" column and "sentiment" column and using the different parameters of test_size for project comparison (could be 0.1, 0.2, 0.3, 0.4, 0.5 for comparison study).

5.1.5. Term Frequency–Inverse Document Frequency (TF-IDF)

```
88. from sklearn.feature_extraction.text import TfidfVectorizer
89. vectorizer = TfidfVectorizer()
90. X_train = vectorizer.fit_transform(X_train)
91. X_test = vectorizer.transform(X_test)
92. print(X_train.shape)
93. print(X_test.shape)
```

Each of splitted datasets (X_train and X_test) are fitted (.fit_transform()) and transformed (.transform()) using vectorizer variable that abbreviated from TfidfVectorizer() after importing TfidfVectorizer from sklearn library. And if using print(X_train.shape) and print(X_test.shape), it compiles same number of shapes from each splitted datasets.

5.1.6. Support Vector Machine (SVM)

1. Cross Validation Score

```
94. from sklearn.svm import SVC
95. from sklearn.model_selection import cross_val_score
96. model = SVC(kernel='linear')
97. cvlinear_5_fold = cross_val_score(model, X_train, y_train, cv=5)
98. cvlinear_5fold_mean = cvlinear_5_fold.mean()
```

After importing Support Vector Classification (SVC) and cross_val_score, the model variable contains SVC function that contained with kernel parameter that using different kernels (could be linear, rbf, poly, sigmoid). Along with model, X_train, y_train variable, and cv parameters that uses 5 and 10 in the cross_val_score function are transferred into cvlinear_5_fold variable (the word

linear can be replaced by linear, rbf, poly, and sigmoid and 5 can be replaced by 10 to make the project comparison) to display the 5 iterations for accuracy fold. And for the average of 5-fold accuracy, use the ".mean()" function next to cvlinear_5_fold variable.

2. Accuracy Evaluation

```
99. from sklearn.svm import SVC
100.
101.model = SVC(kernel='linear')
102.model.fit(X_train, y_train)
103.predict = model.predict(X_test)
104.
105.from sklearn.metrics import f1_score, recall_score, precision_score,
   confusion_matrix, accuracy_score
106.
107.f1_rbf = f1_score(y_test, predict)
108.accuracy_rbf = accuracy_score(y_test, predict)
109.precision_rbf = precision_score(y_test, predict)
110.recall_rbf = recall_score(y_test, predict)
```

After importing SVC and cross_val_score, the model variable contains SVC function that contained with kernel parameter that using different kernels (could be linear, rbf, poly, sigmoid). Then, the model variable are fitted with the splitted dataset (X_train and y_train) and then using predict function (predict()) next to model variable transferred into predict variable to predict all of testing dataset from splitted dataset (X_test variable). Finally, it evaluated using predict and y_test variable using f1_score, recall_score, precision_score, and accuracy_score function after importing it from sklearn library.

5.1.7. Dataset Prediction

```
111.predicted_words = [predicted_words]
112.predicted_words = vectorizer.transform(predicted_words)
113.result = model.predict(predicted_words)
```

Each of reviews of prediction dataset are transferred into predicted_words dataset after preprocessing it like on doing preprocessing the main dataset mentioned before. Next, it changed into giving "[]" between the predicted_words variable. For example, "phone good" changed into "[phone good]". Then it transformed (.transform()) using vectorizer variable that also abbreviated from TfidfVectorizer() in predicted_words variable. Afterwards, using predict function (predict())

next to model variable transferred into result variable to predict sentiment result from predicted_words variable which can be result "[0]" as Negative or "[1]" as Positive.

5.2. Results

5.2.1. Train and Test Accuracy on Main Dataset

Table 5.1. Accuracy evaluation on main dataset

SVM Kernels	**Test size accuracy**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	89,85 %	89,50 %	89,05 %	89,20 %	88,88 %
RBF	**90,40 %**	**90,16%**	**89,85 %**	**90,07 %**	**89,76 %**
Polynomial	89,65 %	89,20 %	88,62 %	88,43 %	87,80 %
Sigmoid	87,32 %	87,11 %	86,73 %	87,26 %	87,39 %

After I compiled all of each SVM kernels and each test sizes using main dataset, the highest accuracy on this result based on SVM kernel are all in Radial Basis Function (RBF) kernel in all test size. With the highest percentage is 90,40% in 0,1 test size.

Table 5.2. Accuracy evaluation on main dataset (5-fold)

SVM Kernels	**Test size accuracy (5-fold)**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	89,19 %	89,23 %	89,01 %	88,77 %	88,82 %
RBF	**90,11 %**	**90,13 %**	**89,87 %**	**89,66 %**	**89,59 %**
Polynomial	88,38 %	88,26 %	87,58 %	87,39 %	86,87 %
Sigmoid	87,23 %	87,48 %	87,26 %	87,58 %	87,51 %

Table 5.3. Accuracy evaluation on main dataset (10-fold)

SVM Kernels	**Test size accuracy (10-fold)**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	89,38 %	89,35 %	89,14 %	88,90 %	89,02 %
RBF	**90,26 %**	**90,24 %**	**89,95 %**	**89,78 %**	**89,74 %**
Polynomial	88,71 %	88,51 %	87,92 %	87,72 %	87,27 %
Sigmoid	87,11 %	87,54 %	87,33 %	87,37 %	87,37 %

Meanwhile, for the 5 and 10-fold cross validation after I compile using main dataset, the highest accuracy on this result based on SVM kernel are all in RBF kernel in all test size and in 5 and 10-fold cross validation. With the highest percentage is 90,13% in 0,2 test size on 5-fold cross validation and 90,26% in 0,1 test size on 10-fold cross validation.

5.2.2. Train and Test Accuracy on Main Dataset without Stemming

Table 5.4. Accuracy evaluation on main dataset without stemming

SVM Kernels	**Test size accuracy**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	90,26 %	89,04 %	90,22 %	88,98 %	88,73 %
RBF	**90,97 %**	**90,12 %**	91,28 %	**89,63 %**	**89,66 %**
Polynomial	89,36 %	88,81 %	**92,85 %**	88,17 %	88,15 %
Sigmoid	87,71 %	86,66 %	86,26 %	86,95 %	87,22 %

When I tried to compile all the project without using stemming, all of SVM kernels I tried based on the test size, RBF is the highest accuracy in all test size except of 0,3 test size where Polynomial kernel is the highest accuracy with the percentage is 92,85 which is the highest accuracy.

Table 5.5. Accuracy evaluation on main dataset without stemming (5-fold)

SVM Kernels	**Test size accuracy (5-fold)**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	89,10 %	89,22 %	89,11 %	89,40 %	88,74 %
RBF	**89,81 %**	**90,08 %**	**89,73 %**	**90,11 %**	**89,55 %**
Polynomial	88,22 %	88,08 %	87,73 %	87,88 %	86,92 %
Sigmoid	86,90 %	87,04 %	87,40 %	88,18 %	87,71 %

Table 5.6. Accuracy evaluation on main dataset without stemming (10-fold)

SVM Kernels	**Test size accuracy (10-fold)**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	89,15 %	89,40 %	89,17 %	89,44 %	88,97 %
RBF	**89,94 %**	**90,11 %**	**90,00 %**	**90,21 %**	**89,63 %**
Polynomial	88,49 %	88,40 %	88,09 %	87,95 %	87,32 %
Sigmoid	86,82 %	87,00 %	87,35 %	87,65 %	87,58 %

Meanwhile, when it comes to 5-fold accuracy, RBF is the highest accuracy in all test size with the highest percentage is 90,11 % with 0,4 test size. And for 10-fold accuracy, RBF is the highest accuracy in all test size with the highest percentage is 90,21 % with 0,4 test size.

5.2.3. Train and Test Accuracy on Main Dataset without Lemmatization

Table 5.7. Accuracy evaluation on main dataset without lemmatization

SVM Kernels	**Test size accuracy**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	89,78 %	89,68 %	89,23 %	89,28 %	89,20 %
RBF	**90,38 %**	**90,57 %**	**90,05 %**	**90,02 %**	**90,17 %**
Polynomial	89,01 %	88,33 %	88,30 %	87,75 %	87,15 %
Sigmoid	86,55 %	86,35 %	87,50 %	87,17 %	87,34 %

When I tried to compile all the project without using lemmatization, all of SVM kernels I tried based on the test size, RBF is the highest accuracy in all test size the highest accuracy is in 0,2 test size with the percentage is 90,57 %.

Table 5.8. Accuracy evaluation on main dataset without lemmatization (5-fold)

SVM Kernels	**Test size accuracy (5-fold)**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	89,74 %	89,44 %	89,58 %	89,20 %	89,09 %
RBF	**90,50 %**	**90,29 %**	**90,20 %**	**90,05 %**	**89,83 %**
Polynomial	87,78 %	87,36 %	87,18 %	86,68 %	85,77 %
Sigmoid	87,45 %	87,57 %	87,60 %	87,63 %	88,05 %

Table 5.9. Accuracy evaluation on main dataset without lemmatization (10-fold)

SVM Kernels	**Test size accuracy (10-fold)**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	89,67 %	89,41 %	89,69 %	89,25 %	89,09 %
RBF	**90,46 %**	**90,34 %**	**90,41 %**	**90,01 %**	**89,90 %**
Polynomial	88,14 %	87,80 %	87,50 %	87,03 %	86,23 %
Sigmoid	87,43 %	87,00 %	87,62 %	87,67 %	87,84 %

Meanwhile, when it comes to 5-fold accuracy, RBF is the highest accuracy in all test size with the highest percentage is 90,50 % with 0,1 test size. And for 10-fold accuracy, RBF is the highest accuracy in all test size with the highest percentage is 90,46 % with 0,1 test size.

5.2.4. Train and Test Accuracy on Main Dataset without Helpful Votes Calculation

Table 5.10. Accuracy evaluation on main dataset without helpful votes calculation

SVM Kernels	**Test size accuracy**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	89,45 %	89,32 %	89,21 %	88,92 %	88,85 %
RBF	**90,56 %**	**90,14 %**	**89,91 %**	**89,78 %**	**89,66 %**
Polynomial	88,90 %	88,88 %	88,27 %	88,19 %	87,61 %
Sigmoid	87,47 %	87,61 %	87,16 %	87,63 %	87,18 %

When I tried to compile all the project without using helpful votes calculation, all of SVM kernels I tried based on the test size, RBF is the highest accuracy in all test size the highest accuracy is in 0,1 test size with the percentage is 90,56 %.

Table 5.11. Accuracy evaluation on main dataset without helpful votes calculation (5-fold)

SVM Kernels	**Test size accuracy (5-fold)**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	89,50 %	89,40 %	89,41 %	88,92 %	88,93 %
RBF	**90,29 %**	**90,10 %**	**90,12 %**	**89,79 %**	**89,61 %**
Polynomial	88,36 %	88,20 %	87,75 %	87,46 %	86,98 %
Sigmoid	87,44 %	87,18 %	87,39 %	87,49 %	87,73 %

Table 5.12. Accuracy evaluation on main dataset without helpful votes calculation (10-fold)

SVM Kernels	**Test size accuracy (10-fold)**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	89,54 %	89,38 %	89,47 %	89,12 %	88,96 %
RBF	**90,35 %**	**90,19 %**	**90,26 %**	**89,96 %**	**89,76 %**
Polynomial	88,76 %	88,56 %	88,04 %	87,87 %	87,24 %
Sigmoid	87,27 %	87,32 %	87,21 %	87,21 %	87,59 %

Meanwhile, when it comes to 5-fold accuracy, RBF is the highest accuracy in all test size with the highest percentage is 90,29 % with 0,1 test size. And for 10-fold accuracy, RBF is the highest accuracy in all test size with the highest percentage is 90,35 % with 0,1 test size.

5.2.5. Train and Test Accuracy on Main Dataset without Undersampling

Table 5.13. Accuracy evaluation on main dataset without undersampling

SVM Kernels	**Test size accuracy**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	91,16 %	90,98 %	90,92 %	90,83 %	90,73 %
RBF	**92,03 %**	**91,82 %**	**91,77 %**	**91,60 %**	**91,37 %**
Polynomial	89,03 %	88,29 %	88,09 %	87,61 %	86,97 %
Sigmoid	89,22 %	89,50 %	89,66 %	89,42 %	89,65 %

When I tried to compile all the project without using undersampling that it takes a long time, all of SVM kernels I tried based on the test size, RBF is the highest accuracy in all test size the highest accuracy is in 0,1 test size with the percentage is 92,03 %

Table 5.14. Accuracy evaluation on main dataset without undersampling (5-fold)

SVM Kernels	**Test size accuracy (5-fold)**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	90,69 %	90,61 %	90,35 %	90,25 %	90,14 %
RBF	**91,50 %**	**91,37 %**	**91,01 %**	**90,90 %**	**90,71 %**
Polynomial	87,52 %	87,12 %	86,38 %	85,82 %	85,19 %
Sigmoid	89,24 %	89,30 %	89,19 %	89,17 %	89,47 %

Table 5.15. Accuracy evaluation on main dataset without undersampling (10-fold)

SVM Kernels	**Test size accuracy (10-fold)**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	90,77 %	90,62 %	90,46 %	90,35 %	90,20 %
RBF	**91,63 %**	**91,48 %**	**91,26 %**	**91,01 %**	**90,82 %**
Polynomial	87,94 %	87,55 %	86,93 %	86,36 %	85,73 %
Sigmoid	89,22 %	89,22 %	89,28 %	89,24 %	89,21 %

Meanwhile, when it comes to 5-fold accuracy, RBF is the highest accuracy in all test size with the highest percentage is 91,50 % with 0,1 test size. And for 10-fold accuracy, RBF is the highest accuracy in all test size with the highest percentage is 91,63 % with 0,1 test size.

5.2.6. Accuracy on Prediction Dataset

Table 5.16. Predicted accuracy on 720 rewiews

SVM Kernels	**Test size accuracy**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	643	642	637	642	640
RBF	**646**	**644**	**649**	**646**	**651**
Polynomial	628	636	628	631	624
Sigmoid	636	631	634	634	644

Table 5.17. Predicted accuracy on 720 reviews in percentage

SVM Kernels	**Test size accuracy**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	89,31 %	89,17 %	88,47 %	89,17 %	88,89 %
RBF	**89,72 %**	**89,44 %**	**90,14 %**	**89,72 %**	**90,42 %**
Polynomial	87,22 %	88,33 %	87,22 %	87,64 %	86,67 %
Sigmoid	88,33 %	87,64 %	88,06 %	88,06 %	89,44 %

In this project using prediction dataset, I compare the tested prediction dataset with the labelled sentiment on prediction dataset using sentiment determiner which same thing used on main dataset but without detecting the helpful votes because of no helpful votes column are here on prediction dataset.

$$Accuracy = \frac{Correct\ predictions}{All\ of\ predictions} \times 100$$

In this experiment, I use the percentage formulation which is written in above to calculate the all of correct prediction divided with all of prediction multiplied with 100 to make a percentage of them based of SVM kernels and test sizes.

From the result on this, in the Table 5.16 and 5.17 the highest accuracy based of SVM kernels on prediction dataset is the RBF kernel in all test sizes with the highest percentage is 90,42% in 0,5 size (Table 5.16) which 651 of 720 (Table 5.17) reviews on prediction dataset are correctly predicted.

5.2.7. Accuracy on Prediction Dataset without Stemming

Table 5.18. Predicted accuracy on 720 reviews without stemming

SVM Kernels	**Test size accuracy**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	647	643	638	637	637
RBF	**652**	**650**	**643**	**643**	**647**
Polynomial	629	627	618	622	616
Sigmoid	633	630	631	634	633

Table 5.19. Predicted accuracy on 720 reviews without stemming in percentage

SVM Kernels	**Test size accuracy**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	89,86 %	89,31 %	88,61 %	88,47 %	88,47 %
RBF	**90,56 %**	**90,28 %**	**89,31 %**	**89,31 %**	**89,86 %**
Polynomial	87,36 %	87,08 %	85,83 %	86,39 %	85,56 %
Sigmoid	87,92 %	87,50 %	87,64 %	88,06 %	87,92 %

From the result if stemming is not used, in the Table 5.18 and 5.19 the highest accuracy based of SVM kernels on prediction dataset is the RBF kernel in all test sizes with the highest percentage is 90,56% in 0,1 size (Table 5.18) which 652 of 720 (Table 5.19) reviews on prediction dataset are correctly predicted.

5.2.8. Accuracy on Prediction Dataset without Lemmatization

Table 5.20. Predicted accuracy on 720 reviews without lemmatization

SVM Kernels	**Test size accuracy**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	635	636	635	632	629
RBF	**641**	**644**	**642**	**637**	**637**
Polynomial	624	620	610	605	607
Sigmoid	630	627	627	631	626

Table 5.21. Predicted accuracy on 720 reviews without lemmatization in percentage

SVM Kernels	**Test size accuracy**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	88,19 %	88,33 %	88,19 %	87,78 %	87,36 %
RBF	**89,03 %**	**89,44 %**	**89,17 %**	**88,47 %**	**88,47 %**
Polynomial	86,67 %	86,11 %	84,72 %	84,03 %	84,31 %
Sigmoid	87,50 %	87,08 %	87,08 %	87,64 %	86,94 %

From the result if lemmatization is not used, in the Table 5.20 and 5.21 the highest accuracy based of SVM kernels on prediction dataset is the RBF kernel in all test sizes with the highest percentage is 89,44% in 0,2 size (Table 5.20) which 644 of 720 (Table 5.21) reviews on prediction dataset are correctly predicted.

5.2.9. Accuracy on Prediction Dataset without Helpful Votes Calculation

Table 5.22. Predicted accuracy on 720 reviews without helpful votes calculation

SVM Kernels	**Test size accuracy**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	646	645	639	632	632
RBF	**653**	**647**	**646**	**634**	**646**
Polynomial	630	628	628	619	619
Sigmoid	637	632	628	631	632

Table 5.23. Predicted accuracy on 720 reviews without helpful votes calculation in percentage

SVM Kernels	**Test size accuracy**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	89,72 %	89,58 %	88,75 %	87,78 %	87,78 %
RBF	**90,69 %**	**89,86 %**	**89,72 %**	**88,06 %**	**89,72 %**
Polynomial	87,50 %	87,22 %	87,22 %	85,97 %	85,97 %
Sigmoid	88,47 %	87,78 %	87,22 %	87,64 %	87,78 %

From the result if helpful votes calculation is not used, in the Table 5.22 and 5.23 the highest accuracy based of SVM kernels on prediction dataset is the RBF kernel in all test sizes with the highest percentage is 90,69% in 0,1 size (Table 5.22) which 653 of 720 (Table 5.23) reviews on prediction dataset are correctly predicted.

5.2.10.Accuracy on Prediction Dataset without Undersampling

Table 5.24. Predicted accuracy on 720 reviews without undersampling

SVM Kernels	**Test size accuracy**				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	659	663	658	659	653
RBF	**667**	**666**	**666**	**662**	**659**
Polynomial	632	629	626	625	626
Sigmoid	651	653	655	647	652

Table 5.25. Predicted accuracy on 720 reviews without undersampling in percentage

SVM Kernels	Test size accuracy				
	0,1	***0,2***	***0,3***	***0,4***	***0,5***
Linear	91,53 %	92,08 %	91,39 %	91,53 %	90,69 %
RBF	**92,64 %**	**92,50 %**	**92,50 %**	**91,94 %**	**91,53 %**
Polynomial	87,78 %	87,36 %	86,94 %	86,81 %	86,94 %
Sigmoid	90,42 %	90,69 %	90,97 %	89,86 %	90,56 %

From the result if undersampling is not used, in the Table 5.24 and 5.25 the highest accuracy based of SVM kernels on prediction dataset is the RBF kernel in all test sizes with the highest percentage is 92,64% in 0,1 size (Table 5.24) which 667 of 720 (Table 5.25) reviews on prediction dataset are correctly predicted.

5.2.11.Conlusion from whole Accuracy Results

Table 5.26. Highest accuracy Comparison between 5 experiments

	Normal	**Without Stemming**	**Without Lemmatization**	**Without Helpful Votes Calculation**	**Without Undersampling**
Accuracy Evaluation	90,40 %, RBF kernel, 0,1 test size	**92,85 %, Polynomial kernel, 0,3 test size**	90,57 %, RBF kernel, 0,2 test size	90,56 %, RBF kernel, 0,1 test size	92,03 %, RBF kernel, 0,1 test size
5 fold	90,13 %, RBF kernel, 0,2 test size	90,11 %, RBF kernel, 0,4 test size	90,50 %, RBF kernel, 0,1 test size	90,29 %, RBF kernel, 0,1 test size	**91,50 %, RBF kernel, 0,1 test size**
10 fold	90,26 %, RBF kernel, 0,1 test size	90,21 %, RBF kernel, 0,4 test size	90,46 %, RBF kernel, 0,1 test size	90,35 %, RBF kernel, 0,1 test size	**91,63 %, RBF kernel, 0,1 test size**
Prediction	90,42 %, RBF kernel, 0,5 test size	90,56 %, RBF kernel, 0,1 test size	89,44 %, RBF kernel, 0,2 test size	90,69 %, RBF kernel, 0,1 test size	**92,64 %, RBF kernel, 0,1 test size**

When I compare the highest percentage of accuracy, 5 fold, 10 fold, and prediction in this project experiment, it got result that the highest percentage and SVM kernel of accuracy evaluation is without stemming experiment with 92,85% and Polynomial kernel with 0,3 test size compared to normal experiment that only got 90,40% and RBF kernel with 0,1 test size. Meanwhile, the

highest percentage and SVM kernel in 5-fold, 10-fold, and prediction experiment is without undersampling experiment with each percentage are 91,50%, 91,63%, and 92,64% and all of them are in RBF kernel with 0,1 test size compared to normal experiment with each percentage are 90,13%, 90,26%, and 90,42% with 0,2, 0,1, and 0,5 test size and all of them are in RBF kernel. So, the accuracy experiment of without stemming and Polynomial kernel is higher than accuracy experiment of normal experiment. Then, prediction, 5-fold, and 10-fold of without undersampling percentage are higher than normal experiment. It concluded that without stemming experiment has better accuracy test and without undersampling has better 5-fold, 10-fold, and prediction test.

Table 5.27. Tendency of 4 other experiments against normal experiments

	Without Stemming	**Without Lemmatization**	**Without Helpful Votes Calculation**	**Without Undersampling**
Accuracy	Decreased	Decreased	Decreased	Increased
5-fold	Decreased	Increased	Increased	Increased
10-fold	Decreased	Increased	Increased	Increased
Prediction	Decreased	Decreased	Decreased	Increased

When I compare the tendency of increasing or decreasing on 4 other experiments (without stemming, without lemmatization, without helpul votes calculation, and without undersampling) on whole test size and SVM kernels in each experiments, resulted that if using without stemming feature, all of the percentage accuracy on it tended to be decreased compared into normal experiment, and in contrast with using without undersampling feature that all of the percentage accuracy on it tended to be increased compared into normal experiment.

In the without lemmatization and without helpful votes calculation experiments, the accuracy and prediction percentage results are decreased compared to 5-fold and 10-fold that percentage results are increased.

Interestingly, although of without stemming experiment is the better accuracy than normal experiment, many of whole test size and SVM kernels accuracy percentage on it tended to be decreasing in some test size and SVM kernels (many of it in 0,2, 0,4, and 0,5 test size). Although of increased percentage accuracy on without undersampling experiment compared into normal experiment, the accuracy percentage on without stemming experiment is the higher than on without undersampling.