VINCENT MICHAEL SUTANTO - 2111434
vincent.michael_sutanto.vg6@is.naist.jp

Q. Before an era of deep learning, NLP researchers had to select features for their model. In this tutorial, we will manually select these features and apply them to our models. Your missions are:

- Create your own model to predict the correct class of dataset (DO NOT USE DEEP LEARNING method)
- It can be: perceptron, Naive Bayes, etc. It depends on you.
- Select features for your model.
- Some simple features can be n-gram, capital word, etc. Again, they depend on you.
- Write your report
- Write a short explaining about your program and show us your code
- Show your accuracy on the test set

Training file: https://github.com/neubig/nlptutorial/blob/master/data/titles-en-train.labeled

Test file: https://github.com/neubig/nlptutorial/blob/master/data/titles-en-test.word

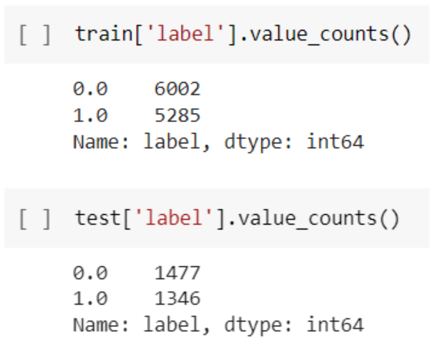
Abstract

In this task, I tested the dataset with ***logistic regression*** *that I coded from scratch. By combining the features of* ***1-gram, 2-gram, sentence length, and consecutive uppercase letters****, the logistic regression model is able to produce an accuracy of up to* ***91.64%****.*

**please see the notebook for the code, because I think it would be too long to explain all the code in this report*

Data Analysis

For class prediction problems, we first need to know what the characteristics of the data used are. The first thing I did was look at the comparison between the amount of data from the PERSON class and the NON-PERSON class. This is important considering that the classification model tends to be better if it is trained on data that has a balance from each class. It can be seen from the below picture that the data used is quite balanced, so there is no need to undersampling / oversampling the train data and test data.

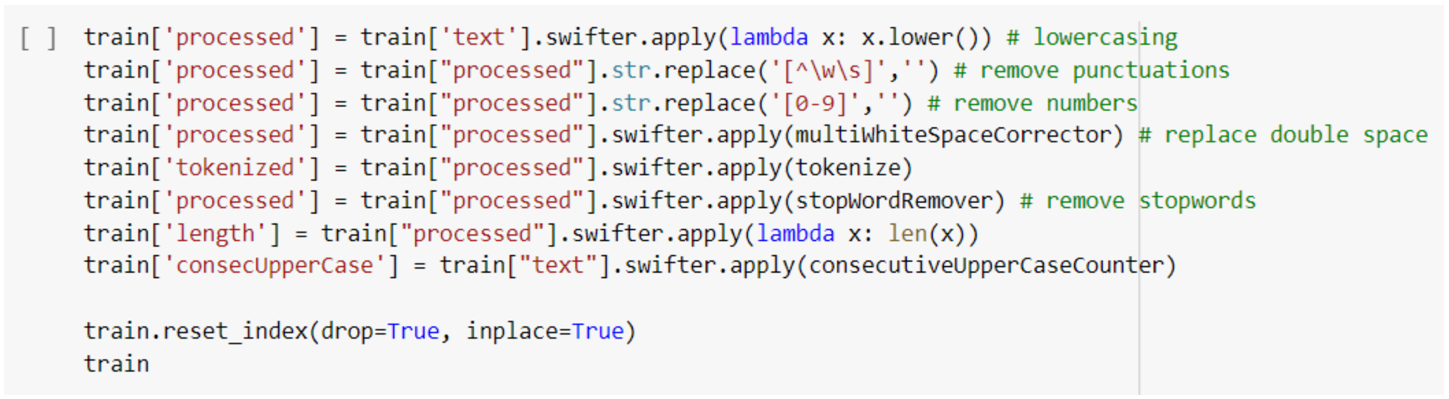


```
[ ] train['label'].value_counts()

    0.0    6002
    1.0    5285
    Name: label, dtype: int64

[ ] test['label'].value_counts()

    0.0    1477
    1.0    1346
    Name: label, dtype: int64
```

Data distribution

Preprocessing and Feature Generation

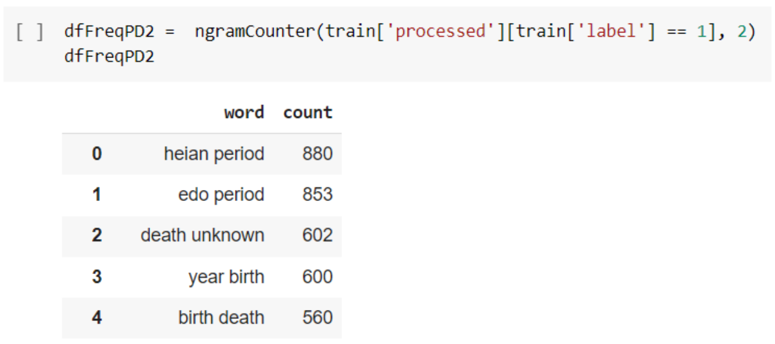
Next, preprocessing is carried out on the dataset. Sequentially, the processes carried out are lowercasing, removing punctuation, removing numbers, correcting whitespace, tokenization, and removing stopwords. The previously mentioned process is continued by generating 2 new features, namely the length of the sentence and the number of consecutive uppercases that emerge from the sentence. Consecutive uppercase is a strong feature because the name of the PERSON class is almost always written in all uppercase letters (Sodo YAMAGUCHI, Kakugoro INOUE, etc).



```
[ ] train['processed'] = train['text'].swifter.apply(lambda x: x.lower()) # lowercasing
    train['processed'] = train["processed"].str.replace('[^\w\s]','') # remove punctuations
    train['processed'] = train["processed"].str.replace('[0-9]','') # remove numbers
    train['processed'] = train["processed"].swifter.apply(multiWhiteSpaceCorrector) # replace double space
    train['tokenized'] = train["processed"].swifter.apply(tokenize)
    train['processed'] = train["processed"].swifter.apply(stopWordRemover) # remove stopwords
    train['length'] = train["processed"].swifter.apply(lambda x: len(x))
    train['consecUpperCase'] = train["text"].swifter.apply(consecutiveUpperCaseCounter)

    train.reset_index(drop=True, inplace=True)
    train
```

Steps of preprocessing

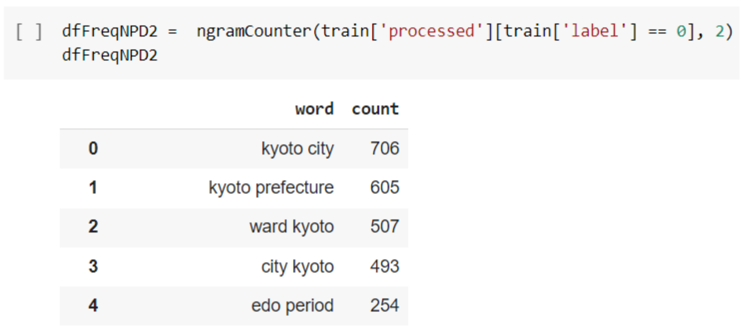
In addition to the additional sentence length feature and the number of consecutive uppercase, I added a new feature in the form of n-gram words. This feature is generated by calling the ngramCounter function, where the function will return a dataframe containing the frequency of occurrence of n-grams. In addition, I perform an outer join function also to find the probability of occurrence of an n-gram in the PERSON class.



```
[ ] dfFreqPD2 =  ngramCounter(train['processed'][train['label'] == 1], 2)
    dfFreqPD2
```

	word	count
0	heian period	880
1	edo period	853
2	death unknown	602
3	year birth	600
4	birth death	560

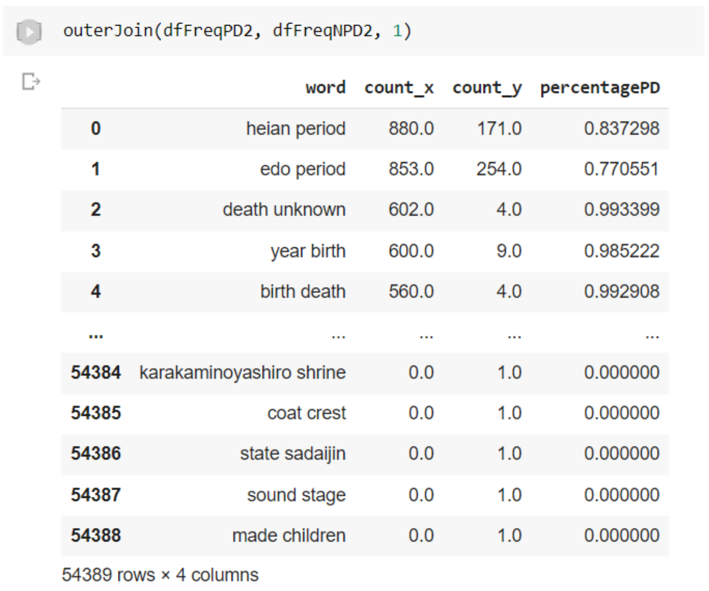
2-gram frequencies on class PERSON



```
[ ] dfFreqNPD2 =  ngramCounter(train['processed'][train['label'] == 0], 2)
    dfFreqNPD2
```

	word	count
0	kyoto city	706
1	kyoto prefecture	605
2	ward kyoto	507
3	city kyoto	493
4	edo period	254

2-gram frequencies on class NON-PERSON



```
outerJoin(dfFreqPD2, dfFreqNPD2, 1)
```

	word	count_x	count_y	percentagePD
0	heian period	880.0	171.0	0.837298
1	edo period	853.0	254.0	0.770551
2	death unknown	602.0	4.0	0.993399
3	year birth	600.0	9.0	0.985222
4	birth death	560.0	4.0	0.992908
...	...	...	...	...
54384	karakaminoyashiro shrine	0.0	1.0	0.000000
54385	coat crest	0.0	1.0	0.000000
54386	state sadaijin	0.0	1.0	0.000000
54387	sound stage	0.0	1.0	0.000000
54388	made children	0.0	1.0	0.000000

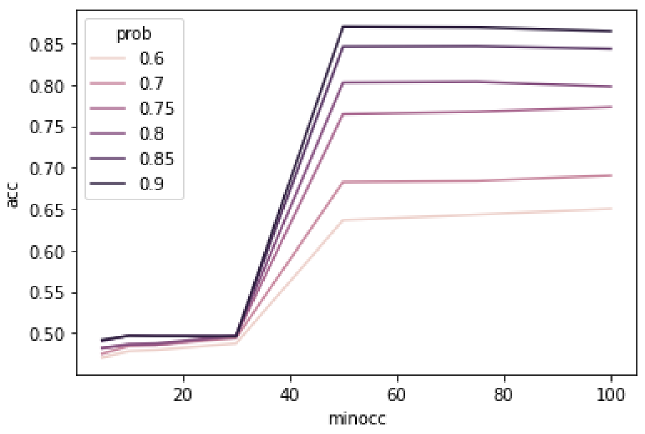
54389 rows × 4 columns

Outer-join of 2-gram frequencies on both class PERSON and NON-PERSON

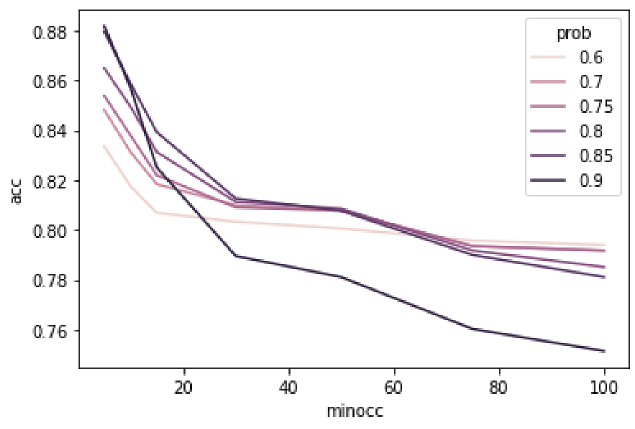
Notice that the outerJoin function has a third input parameter. The third input parameter sets the minimum frequency of sentences that appear from the n-gram class PERSON. For example, when the third parameter is set to 50, then n-grams with a frequency less than 50 from the PERSON class will not be outerJoined. I do this so that later the n-grams used do not overfit the data train. Furthermore, it can also be seen that the outerJoin function returns the percentagePD column. The percentagePD column is the percentage of occurrences of an n-gram, whether it is skewed to the PERSON class or to the NON-PERSON class. I use this feature as a threshold to apply the dominant n-gram to the PERSON class only.

However, fine-tuning needs to be done to find out the best minimum frequency and probability combination for each n-gram. Therefore, I did 5-Fold Cross Validation on the training data to find out the value of the hyperparameter of the n-gram feature. At this stage the Logistic Regression model is used. The results of the best combination of parameters are presented in the following table.

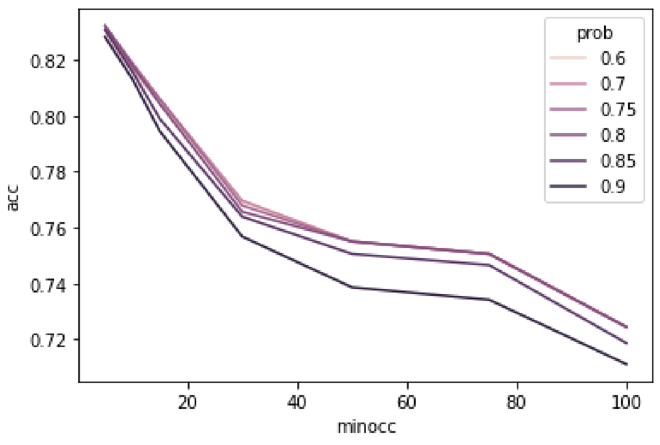
N-GRAM	PROBABILITY	MINIMUM OCCURRENCE	ACCURACY
1	0.9	50	0.8697385910500665
2	0.9	5	0.8817013735046522
3	0.75	5	0.8320779796189632
4	0.8	5	0.765618077093487



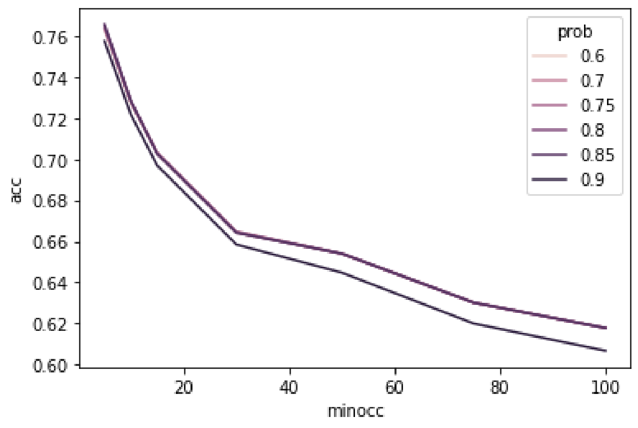
1-gram



2-gram



3-gram



4-gram

Evaluation on Test Data

I trained the logistic regression model that I created from scratch on several training schemes as tabulated in the following table.

Feature(s)	Accuracy
consecUpperCase	0.8104852993269571
1,2,3,4-gram	0.8675168260715551

1,2,3,4-gram + length	0.8685795253276656
1,2,3,4-gram + consecUpperCase	0.8855827134254339
1,2,3,4-gram + consecUpperCase + length	0.8813319164009918
1,2,3-gram + consecUpperCase + length	0.9096705632306057
1,2-gram + consecUpperCase + length	**0.916400991852639**
1-gram + consecUpperCase + length	0.8537017357421183
2-gram + consecUpperCase + length	0.7913567127169677

From the table above, it can be seen how using only one feature (consecutive uppercase) the logistic regression model can achieve up to 81% accuracy. I also investigated that there were miss-labeled data in the train and test data, where PERSON names that met the consecutive uppercase criteria were labeled as NON-PERSON. If the data is improved, I believe the accuracy of using 1 feature will increase.

Next, I tried the combinations of the features that were built before. From the table above, it can be seen that by not using the 3-gram and 4-gram features, and combining all the remaining features, the logistic regression model is able to achieve an accuracy rate of up to 91.64%. In other words, the 3-gram and 4-gram features do not increase the accuracy, while the combination of the 1-gram and 2-gram features increases the accuracy of the model.