Assignment Lesson 8 - 9

# - Clustering & Text Mining -

Clustering is an unsupervised method in machine learning–trying to get the hidden group inside a dataset. Clustering is also proven to be useful in various Text Mining procedures.

In this assignment, you will try to perform one Clustering and one Classification using the results of text mining. The specification of the dataset is given in each part.

**Part A**

**[ ! ]** *Two students who submit their assignments by the deadline with the best accuracy and text-mining steps will receive an* **Amazon Gift Card worth 3000 yen**. *The result will be announced around 2 weeks after the deadline of the assignment.*

**Part B**

**[ ! ]** *Two students who submit their assignments by the deadline with the most interesting topic and text-mining steps will receive an* **Amazon Gift Card worth 3000 yen**. *The result will be announced around 2 weeks after the deadline of the assignment.*

# Part A: Classification of Spam Messages

## Prior Knowledge

### Data Science Objective

**Objective**

The SMS Spam Collection is a set of SMS-tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged according to *ham* (legitimate) or *spam*.

The files contain one message per line. Each line is composed of two columns: v1 contains the label (ham or spam), and v2 contains the raw text. We are wondering if we can use this dataset to build a prediction model that will accurately classify which texts are spam.

**Link**

**Rapidminer**

**Dataset 1: SMS Spam Detection**

To keep all the rows, when reading the csv, uncheck the 'Use Quotes' option. 


**Python**

You can also use the following code to retrieve the dataset:

```
!curl https://dl.dropboxusercontent.com/s/9zvdqsfu75cz0oo/spam_or_ham.csv?dl=0 -so spam_or_ham.csv
```

To prevent error when reading the csv, use encoding_errors='ignore'

# Data Preparation & Exploration

## Dataset

This is the screenshot of the **top 20 rows** of the dataset.

| v1 | v2 |
|---|---|
| ham | Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat... |
| ham | Ok lar... Joking wif u oni... |
| spam | Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's |
| ham | U dun say so early hor... U c already then say... |
| ham | Nah I don't think he goes to usf, he lives around here though |
| spam | FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, €1.50 to rcv |
| ham | Even my brother is not like to speak with me. They treat me like aids patent. |
| ham | As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 to copy your friends Callertune |
| spam | WINNER!! As a valued network customer you have been selected to receivea €900 prize reward! To claim call 09061701461. Claim code KL341. Valid 12 hours only. |
| spam | Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030 |
| ham | I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today. |
| spam | SIX chances to win CASH! From 100 to 20,000 pounds txt> CSH11 and send to 87575. Cost 150p/day, 6days, 16+ TsandCs apply Reply HL 4 info |
| spam | URGENT! You have won a 1 week FREE membership in our €100,000 Prize Jackpot! Txt the word: CLAIM to No: 81010 T&C www.dbuk.net LCCLTD POBOX 4403LDNW1A7RW18 |
| ham | I've been searching for the right words to thank you for this breather. I promise i wont take your help for granted and will fulfil my promise. You have been wonderful and a blessing at all times. |
| ham | I HAVE A DATE ON SUNDAY WITH WILL!! |
| spam | XXXMobileMovieClub: To use your credit, click the WAP link in the next txt message or click here>> http://wap. xxxmobilemovieclub.com?n=QJKGIGHJJGCBL |
| ham | Oh k...i'm watching here:) |
| ham | Eh u remember how 2 spell his name... Yes i did. He v naughty make until i v wet. |
| ham | Fine if that€€s the way u feel. That€€s the way its gota b |
| spam | England v Macedonia - dont miss the goals/team news. Txt ur national team to 87077 eg ENGLAND to 87077 Try:WALES, SCOTLAND 4txt/1.20 POBOXox36504W45WQ 16+ |

## Mining the Text 5/5

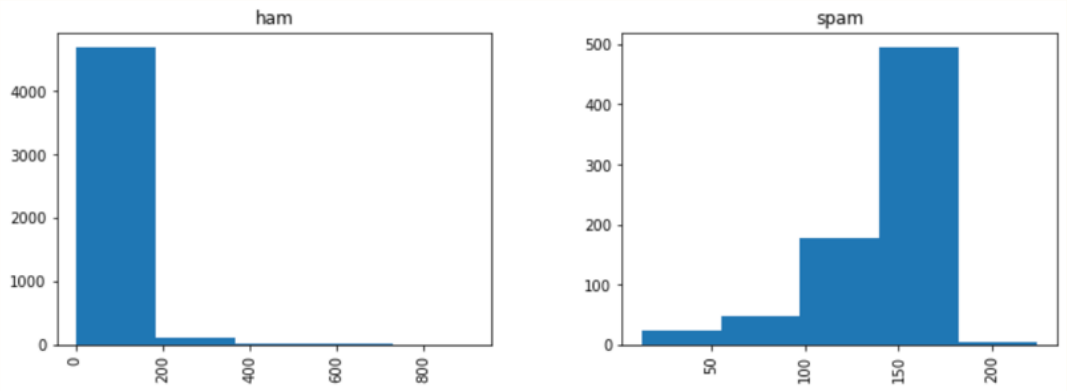
Perform text mining on the dataset. Keep in mind that we are **trying to see the type of word that usually appear in spam messages**. Use at least **5 text-mining steps** in the whole process. Give the screenshot of the whole process here. (In Rapidminer, the text-mining architecture. In Python, the code (no need to include the output)).

```
df = pd.read_csv(f"{_PATH}{_FILE_NAME}")
df.columns = [_LABEL, _MSG]
df.describe()
```

| | label | messages |
|---|---|---|
| count | 5572 | 5572 |
| unique | 2 | 5169 |
| top | ham | Sorry, I'll call later |
| freq | 4825 | 30 |

```
df[_LENGTH] = df[_MSG].apply(len)
df.sort_values(by=_LENGTH, ascending=False).head(10)
df.hist(column=_LENGTH, by=_LABEL,figsize=(12,4), bins = 5)
plt.show()
```

✓ 0.2s




```
spam_messages = df[df[_LABEL] == "spam"][_MSG]
ham_messages = df[df[_LABEL] == "ham"][_MSG]
print(f"Number of spam messages: {len(spam_messages)}")
print(f"Number of ham messages: {len(ham_messages)}")
```

```
Number of spam messages: 747
Number of ham messages: 4825
```

```
_STEMMER = PorterStemmer()

def text_preprocess(message: str, stemmer: object = _STEMMER) -> list:
    tokens = word_tokenize(message)
    final_list = []
    for token in tokens:
        out_punc = re.sub(r'\W+', "", token)
        if len(out_punc) > 2 and out_punc.lower() not in en_stop_words:
            stem = stemmer.stem(out_punc)
            final_list.append(stem)
    return final_list
```

```
df[_TOKEN] = df[_MSG].apply(text_preprocess)
df[_TOKEN] = df[_TOKEN].agg(lambda x: " ".join(map(str, x)))
df.head()
```

Give an analysis/reasoning for why you do those steps for this particular objective. The first step is given as an example. (Feel free to modify Step 1 if you are planning to start with something else.)

**Step 1: Tokenization**

**Reasoning: (20 - 50 words)**

We first decided to tokenize our corpus, i.e., to discretize each message into words that we will refer to as tokens.

**Step 2: Remove all special characters and punctuation 5/5**

**Reasoning: (20 - 50 words)**

After tokenization, we realized that some tokens contain punctuation or special characters, like for example, “Sorry,”. However, these punctuation characters are not useful and may be a problem as “sorry” and “sorry,” will be considered as two different tokens, even though their meaning is the same.

**Step 3: Remove stop words 5/5**

**Reasoning: (20 - 50 words)**

To have good performances, we need to remove all words that do not carry any useful information and hence should be ignored during spam detection. So, we decided to remove all stop words. Using this filter reduces the amount of data in our corpus, which will improve our results.

**Step 4: Remove tokens of length inferior or equal to 2 5/5**

**Reasoning: (20 - 50 words)**

However, when looking at the messages, we identified that almost all tokens whose length is lesser than or equal to two contain no useful information regarding the class of the document. For example, “nt” doesn’t inform if the message is spam or not. We fixed the length to two, because although there are tokens that have a length of three and are useless like “the”, “for”, “was”, etc. we might lose important information such as “win”, “sex”, “see” by removing all the tokens of length 3. Like before, this step reduces the size of the corpus for better performance.

**Step 5: Use Porter’s Stemming 5/5**

**Reasoning: (20 - 50 words)**

By looking at the corpus, we realized that words from the same family (like “see” and “saw”) tends to not be in the same category (for example “see” is more represented in spam messages). However, to simplify our corpus, the next step we decided to do, is to stem words (reduce terms to their basic stems). Stemming is a process of reducing words to their basic form by stripping the plural from nouns (e.g., “texts” to “text”), the suffixes from verbs (e.g., “replied” to “reply”), or other affixes.

**Step 6: (optional)**

**Reasoning: (20 - 50 words)**

**Step 7: (optional)**

**Reasoning: (20 - 50 words)**

### Making TF-IDF matrix 5/5

Give a screenshot of the **proof of making** the TF-IDF matrix (no need to show the matrix). Also, give the **size of the final TF-IDF matrix.**

```
# Bag of Words
vectorizer = CountVectorizer()
bow_transformer = vectorizer.fit(df[_TOKEN])
messages_bow = bow_transformer.transform(df[_TOKEN])
```

```
    # Transform entire BoW into tf-idf corpus
    tfidf_transformer = TfidfTransformer().fit(messages_bow)
    messages_tfidf = tfidf_transformer.transform(messages_bow)
    print(messages_tfidf.shape)
```

✓ 0.2s

```
(5572, 7545)
```

## Classification Modeling

### Classification 10/10 10/10

Split the data into training and test set (30% with stratified sampling). Make any classification model and apply it to the test set. Give a screenshot of the model here.

```
    def _print_shape(X: object, y: object, set: str) -> None:
        print(f"{set} dataset features size: {X.shape}")
        print(f"{set} dataset label size: {y.shape}\n")

    X_train, X_test, y_train, y_test = train_test_split(
        messages_tfidf, df[_LABEL], test_size=0.3, stratify=df[_LABEL].tolist()
    )

    _print_shape(X_train, y_train, "Train")
    _print_shape(X_test, y_test, "Test")
```

✓ 0.3s

```
Train dataset features size: (3900, 7545)
Train dataset label size: (3900,)

Test dataset features size: (1672, 7545)
Test dataset label size: (1672,)
```

```
clf = XGBClassifier()
clf.fit(X_train, y_train)
predict_train = clf.predict(X_train)
predict_test = clf.predict(X_test)
```

✓ 0.4s

```
from sklearn.metrics import classification_report

def _get_accuracy(predict: object, y: object, dataset: str):
    print(f"Accuracy for the {dataset} dataset: {accuracy_score(y, predict)*100:.3}%")
    print(f"Confusion Matrix:\n{confusion_matrix(predict, y)}")
    print(classification_report(y, predict))
    print()

#_get_accuracy(predict_train, y_train, "train")
_get_accuracy(predict_test, y_test, "test")
```

✓ 0.5s

```
Accuracy for the test dataset: 98.0%
Confusion Matrix:
[[1440   25]
 [   8  199]]
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1448
           1       0.96      0.89      0.92       224

    accuracy                           0.98      1672
   macro avg       0.97      0.94      0.96      1672
weighted avg       0.98      0.98      0.98      1672
```

**Performance: (for this problem, which metric is best used? Accuracy? Recall?)**

**The final accuracy is 98.0%, and the final F1-Score is 92%.**

We decided to use F1-Score as a metric because we want to avoid false negatives and false positives. Indeed, we don't have to have spam in our main email box, but we also don't want to categorize an important message as spam. We want to reduce both the risk of spam and losing important information.

**Analysis**（20 - 80 words）

(Hint: What do you think the performance evaluation results mean? Is it good enough? If you have more computational power, would you do things differently?)

The result is quite good. However, it could be improved if there were more spam messages. Indeed, the data is quite imbalanced (which reflects reality; there are more ham messages than spam messages). We tried to solve this problem by downsampling the data. However, after that, the corpus was too small to use it. Having more spam messages could help the performance of our model. If we have more computational power, we can try to use a higher computational cost model, such as an LSTM or Transformer.

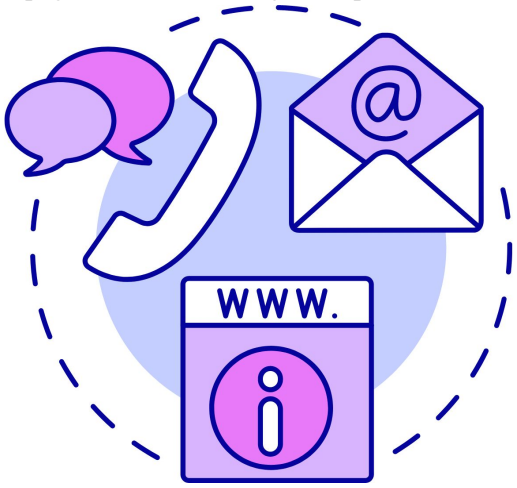
# Part B: Clustering on Some Wikipedia Pages

## Prior Knowledge

### Data Science Objective

**Objective**

Wikipedia contains a lot of webpage with a broad range of topics.




We are wondering if we can use the information on Wikipedia to find a natural grouping of some topics.

State the data science goal concerning the clustering that you will perform on Wikipedia pages on **any topic of your interest** (such as celebrities, movies, games, history, etc.). Ex: In what ways can different movies by Stephen Spielberg be grouped together?

The focus of this study is Wagashi, a traditional Japanese confectionery that is often served with tea. In what way different can different Wagashi be grouped together?

# Data Preparation & Exploration

## Dataset

Give the list of the Wikipedia link that you are planning to perform text-mining on (at least 10).

```
['Akumaki', 'https://en.wikipedia.org/wiki/Akumaki'],
['Amanatto', 'https://en.wikipedia.org/wiki/Amanatt%C5%8D'],
['Arare', 'https://en.wikipedia.org/wiki/Arare_(food)'],
['Beika', 'https://en.wikipedia.org/wiki/Beika'],
['Botamochi', 'https://en.wikipedia.org/wiki/Botamochi'],
['Daifuku', 'https://en.wikipedia.org/wiki/Daifuku'],
['Dango', 'https://en.wikipedia.org/wiki/Dango'],
['Dorayaki', 'https://en.wikipedia.org/wiki/Dorayaki'],
['Gionbo', 'https://en.wikipedia.org/wiki/Gionb%C5%8D'],
['Gyuhi', 'https://en.wikipedia.org/wiki/Gy%C5%ABhi'],
['Hanabiramochi', 'https://en.wikipedia.org/wiki/Hanabiramochi'],
['Higashi', 'https://en.wikipedia.org/wiki/Higashi_(food)'],
['Hishi', 'https://en.wikipedia.org/wiki/Hishi_mochi'],
['Imagawayaki', 'https://en.wikipedia.org/wiki/Imagawayaki'],
['Karinto', 'https://en.wikipedia.org/wiki/Karint%C5%8D'],
['Konpeito', 'https://en.wikipedia.org/wiki/Konpeit%C5%8D'],
['Kuzumochi', 'https://en.wikipedia.org/wiki/Kuzumochi'],
['Manju', 'https://en.wikipedia.org/wiki/Manj%C5%AB'],
['Mizuame', 'https://en.wikipedia.org/wiki/Mizuame'],
['Monaka', 'https://en.wikipedia.org/wiki/Monaka'],
['Namagashi', 'https://en.wikipedia.org/wiki/Namagashi'],
['Sakuramochi', 'https://en.wikipedia.org/wiki/Sakuramochi'],
['Senbei', 'https://en.wikipedia.org/wiki/Senbei'],
['Taiyaki', 'https://en.wikipedia.org/wiki/Taiyaki'],
['Tokoroten', 'https://en.wikipedia.org/wiki/Tokoroten'],
['Uiro', 'https://en.wikipedia.org/wiki/Uir%C5%8D'],
['Warabimochi', 'https://en.wikipedia.org/wiki/Warabimochi'],
['Wasanbon', 'https://en.wikipedia.org/wiki/Wasanbon'],
['Yatsuhashi', 'https://en.wikipedia.org/wiki/Yatsuhashi'],
['Yokan', 'https://en.wikipedia.org/wiki/Y%C5%8Dkan'],
['Yubeshi', 'https://en.wikipedia.org/wiki/Yubeshi']
```

## Mining the Text 5/5

Perform text mining on the dataset. Keep in mind that we are **trying to see the type of word that can distinguish different clusters**. Use at least **5 text-mining steps** in the whole process. Give the screenshot of the whole process here. (In Rapidminer, the text-mining architecture. In Python, the code (no need to include the output)).

```
## Retrieving raw texts
def _remove_wikimarkers(row: str) -> str:
    _BEGIN_MARK = "Jump to search"
    _END_MARKS = ["See also[edit]", "External links[edit]", "References[edit]", "Retrieved from "]
    row = row.partition(_BEGIN_MARK)[2]
    for end_marker in _END_MARKS:
        row = row.partition(end_marker)[0]
    row = re.sub(r'\n(.*?)(Learn how and when to remove this template message)', '', row)
    row = re.sub(r'\n(.*?)Click [show] for important translation instructions.', '', row)
    row = re.sub(r'Consider adding a topic to this template(.*?)\n', '', row)
    row = re.sub(r'You should also add the template (.*?)\n', '', row)
    row = re.sub(r'This article may be expanded with text translated(.*?)Translation.)\n', '', row)
    return row

df[_TEXT] = df[_TEXT].apply(_remove_wikimarkers)
df.to_csv(f"{_PATH}wagashi_minimal.csv")

## Tokenize
regexp = RegexpTokenizer("\w+")
df[_TEXT] = df[_TEXT].apply(regexp.tokenize)

## Lower case
df[_TEXT] = df[_TEXT].apply(lambda x:[item.lower() for item in x])

## Removing Stopwords
df[_TEXT] = df[_TEXT].apply(lambda x: [item for item in x if item not in en_stopwords])

## Stemming using Porter Stemmer
stemmer = PorterStemmer()
df[_TEXT] = df[_TEXT].apply(lambda x: [stemmer.stem(item) for item in x])

## Dropping words that appear in all documents
common_words = list(reduce(set.intersection, [set(item) for item in df[_TEXT] ]))
df[_TEXT] = df[_TEXT].apply(lambda x:[item for item in x if item not in common_words])

## Dropping Short Words
df[_TEXT] = df[_TEXT].apply(lambda x: [item for item in x if len(item)>2])
```

Give an analysis/reasoning for why you do those steps for this particular objective.

### Step 1: Removing all Wikipedia markers 5/5

**Reasoning: (20 - 50 words)**

Wikipedia pages don't always have the same designs. They can have different structures. For example, some parts such as "External references", "See also" or "Translation" can be written or not on a page.

However, those different structure elements won't help us to categorize the Wagashi themselves. Indeed, if we keep them, Manju, Dango, and Yatsuhashi will always be categorized together because there are a lot of Wikipedia markers indicating that the original page in Japanese contains more information than the English one.

Therefore we considered that the raw data is kept between the mentions "Jump to search" and the references. We also removed markers inside the text referencing translation or a lack of references.

### Step 2: Tokenize 5/5

**Reasoning: (20 - 50 words)**

We first decided to tokenize our corpus, i.e., to discretize each message into words that we will refer to as tokens. Indeed, we want to see which Japanese confectioneries have similar properties, so we need to evaluate our corpus at a word level. We kept only word characters.

### Step 3: Lowercase 5/5

**Reasoning: (20 - 50 words)**

As we want to see meaningful patterns between the different pages, we should not distinguish two words that have different letter cases but the same meaning. Therefore the whole corpus is changed to lowercase.

### Step 4: Removing stopwords 5/5

**Reasoning: (20 - 50 words)**

Words that carry information about a sentence's structure and not about Wagashi itself should be ignored during classification. Hence, we decided to remove all stop words. Using this filter guarantees that we will classify the Wagashi based on what they are (substance) and not how they are described (form).

### Step 5: Stemming using Porter Stemmer 5/5

**Reasoning: (20 - 50 words)**

The next step we decided to do, is to stem words (reduce terms to their basic stems). As our corpus is rather short, we think that having all worlds to their basic stems will make the classification better. It will be easier to gather words that have the same meaning together.

### Step 6: Dropping Short words

**Reasoning: (20 - 50 words)**

We realized that there are a lot of short words that don't carry a lot of information.

For example:

- Numbers such as "1", "2", etc.
- Abbreviations such as "en"
- …

We decided to remove words that have a length smaller than 2. If we chose 3, we might lose important information such as "soy" or "hot". If we removed all numbers, we might lose information about quantity or date like "17th", "1772" or "1603".

**Step 7: optional**

**Reasoning: (20 - 50 words)**

### Making TF-IDF matrix 5/5

Give a screenshot of the **proof of making** the TF-IDF matrix (no need to show the matrix). Also, give the **size of the final TF-IDF matrix.**

```
## Make Tfidf Matrix
vectorizer = TfidfVectorizer(ngram_range = (1,2))
tfidf_encodings = vectorizer.fit_transform(np.array(df[_TEXT].astype(str)))
df_tfidf = tfidf_encodings.toarray()
vectors_for_training = np.array(df_tfidf.tolist())
print(vectors_for_training.shape)
```

✓ 0.7s

```
(31, 6646)
```

## Unsupervised Modeling

### Clustering 5/5 2/2 8/8

Perform clustering on the TF-IDF matrix. You can use any clustering method. Give a screenshot of the model here.

```
_NB_CLUSTERS = 4

#cluster_model = AgglomerativeClustering(n_clusters=_NB_CLUSTERS, linkage='average').fit(vectors_for_training)
#cluster_model = KMeans(n_clusters=_NB_CLUSTERS).fit(vectors_for_training)
cluster_model = Birch(n_clusters=_NB_CLUSTERS).fit(vectors_for_training)

results = pd.DataFrame(
          np.hstack((cluster_model.labels_.reshape(-1,1),
          (df[_LABEL].values.reshape(-1,1))))
        )
results.columns = [_CLUSTER, _LABEL]
results.sort_values(by=_CLUSTER)

for id in range(0, _NB_CLUSTERS):
    cluster_text = _get_cluster_items(id, results)
    print(_count_tokens(cluster_text))
    print()
```

Give a screenshot of the result of the clustering here. （The ID name and the cluster it belongs to should be clearly shown.)

## Clustering of 31 Wikipedia pages about Wagashi with the BIRCH algorithm

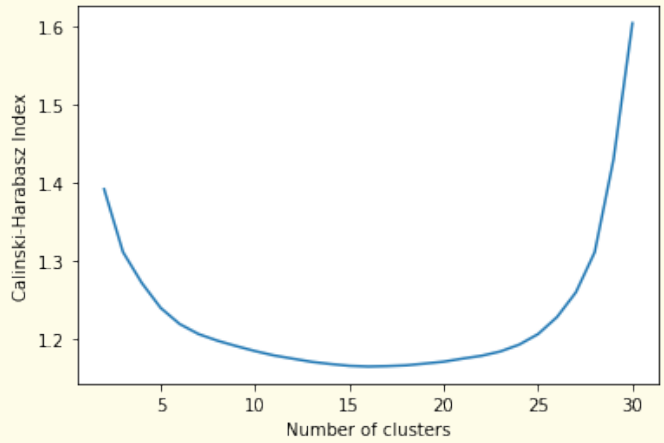
| Cluster 0 | Cluster 1 | Cluster 2 | Cluster 3 |
|---|---|---|---|
| Arare | Amanatto | Botamochi* | Akumaki |
| Beika | Gionbo | Daifuku | Dango |
| Senbei | Higashi* | Dorayaki | Gyuhi |
| | Konpeito | Imagawaki | Hanabiramochi |
| | Namagashi | Manju | Hishi |
| | Wasanbon | Monaka | Karinto* |
| | | Sakuramochi | Kuzumochi |
| | | Taiyaki | Mizuame |
| | | | Tokoroten |
| | | | Uiro |
| | | | Warabimochi |
| | | | Yatsuhashi |
| | | | Yokan |
| | | | Yubeshi |
| **CLUSTER'S LABELISATION AFTER ANALYSIS** | | | |
| **Rice Crackers** | **Candy/ Sugar** | **Filled with red bean paste** | **Other** |

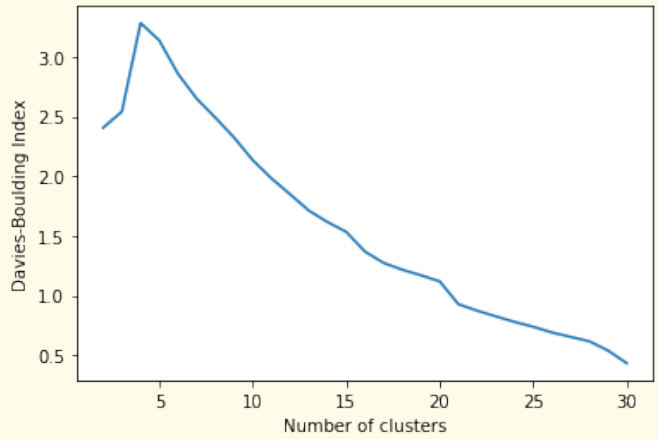
The labels have been chosen after looking at the most frequent common words of the different cluster members. Members marked with an asterisk (*) are considered to be misclassified:

| Wagashi | Classified as | Cluster which may be more appropriate | Reasons |
|---|---|---|---|
| Higashi | Candy/ Sugar | Rice Crackers | Higashi is a broad term for dry confectioneries. It includes rice crackers but also biscuits made with Wasanbon. This term encompasses the clusters Rice Crackers and Candy/ Sugar |
| Botamochi | Filled with red bean paste | Other | Technically, a botamochi is not filled with red bean paste but surrounded by it. |
| Karinto | Other | Candy/ Sugar | Karinto is a sweet and deep-fried snack mainly made with brown sugar. |

**Performance: (Any Clustering performance metrics such as Davies-Bouldin index)**

```
David-Bouldin index: 3.28
Calinski-Harabasz index: 1.27
```




**Analysis**（30 - 100 words）

(Hint: Why do you choose that clustering technique? Do you think the result makes sense? Can you figure out what type of grouping happening there?)

We tried different techniques:

- **Agglomerative Cluster**: For N clusters, we always had results with 1 element in N-1 clusters and the rest in the last cluster. The corpus is too short and diverse to have good performances.
- **KMeans**: One cluster that regroups all dry confectioneries tends to be stable. However, the rest is unstable.

Therefore we chose the algorithm **BIRCH**. This method performs better on bigger corpus and has an easy implementation. We think BIRCH is a better option than DBSCAN as we are not sure that our data is dense. Moreover, it is also easier to fine-tune compared to DBSCAN, which has a lot of parameters (epsilon and minimal sample size).

By looking at the members for each cluster and the main tokens, and also based on our own knowledge, we labeled the clusters (cf above). We think our result is quite good. However, the cluster "Other" could be sectioned into smaller clusters. However, when we increase the number of clusters, it becomes impossible to analyze them. Indeed a lot of cluster members don't share any common information. The same problem appears when we decrease the number of clusters.

One way to ameliorate our model would be to use:

- Bigrams: to capture the context of a word (Indeed, the term "wasanbon" in "*Higashi* made with *wasanbon*, " and in "***Wasanbon*** (和三盆) is a fine-grained Japanese sugar" should be different considered )
- Embedding: to capture words that have similar/close meaning (For example "anko" is similar than "sweet red bean paste")