

hw6-1-tf-idf

February 13, 2025

1 HOMEWORK 6: TEXT CLASSIFICATION

In this homework, you will create models to classify texts from TRUE call-center. There are two classification tasks: 1. Action Classification: Identify which action the customer would like to take (e.g. enquire, report, cancel) 2. Object Classification: Identify which object the customer is referring to (e.g. payment, true money, internet, roaming)

We will focus only on the Object Classification task for this homework.

In this homework, you are asked to compare different text classification models in terms of accuracy and inference time.

You will need to build 3 different models.

1. A model based on tf-idf
2. A model based on MUSE
3. A model based on wangchanBERTa

You will be asked to submit 3 different files (.pdf from .ipynb) that do the 3 different models. Finally, answer the accuracy and runtime numbers in MCV.

This homework is quite free form, and your answer may vary. We hope that the processing during the course of this assignment will make you think more about the design choices in text classification.

```
[1]: # !wget --no-check-certificate https://www.dropbox.com/s/37u83g55p19kvrl/  
      ↪ clean-phone-data-for-students.csv
```

```
[2]: !pip install pythainlp
```

```
Requirement already satisfied: pythainlp in  
/Users/pupipatsingkhorn/miniconda3/envs/datascience/lib/python3.11/site-packages  
(5.0.5)  
Requirement already satisfied: requests>=2.22.0 in  
/Users/pupipatsingkhorn/miniconda3/envs/datascience/lib/python3.11/site-packages  
(from pythainlp) (2.32.3)  
Requirement already satisfied: charset-normalizer<4,>=2 in  
/Users/pupipatsingkhorn/miniconda3/envs/datascience/lib/python3.11/site-packages  
(from requests>=2.22.0->pythainlp) (3.3.2)  
Requirement already satisfied: idna<4,>=2.5 in  
/Users/pupipatsingkhorn/miniconda3/envs/datascience/lib/python3.11/site-packages  
(from requests>=2.22.0->pythainlp) (3.10)  
Requirement already satisfied: urllib3<3,>=1.21.1 in
```

```
/Users/pupipatsingkhorn/miniconda3/envs/datascience/lib/python3.11/site-packages
(from requests>=2.22.0->pythainlp) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/Users/pupipatsingkhorn/miniconda3/envs/datascience/lib/python3.11/site-packages
(from requests>=2.22.0->pythainlp) (2025.1.31)
```

1.1 Import Libs

```
[3]: %matplotlib inline
import pandas
import sklearn
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from torch.utils.data import Dataset
from IPython.display import display
from collections import defaultdict
from sklearn.metrics import accuracy_score

import warnings
warnings.filterwarnings('ignore')
```

```
[4]: SEED = 42
```

1.2 Loading data

First, we load the data from disk into a Dataframe.

A Dataframe is essentially a table, or 2D-array/Matrix with a name for each column.

```
[5]: data_df = pd.read_csv('clean-phone-data-for-students.csv')
```

Let's preview the data.

```
[6]: # Show the top 5 rows
display(data_df.head())
# Summarize the data
data_df.describe()
```

	Sentence	Utterance	Action	Object
0	<PHONE_NUMBER_REMOVED>	Counte...	enquire	payment
1	internet	enquire	package	
2	...	report	suspend	
3	internet	...	enquire	internet
4	...	report	phone_issues	

```
[6]:      Sentence  Utterance  Action  Object
count      16175      16175      16175
```

unique	13389	10	33
top	enquire	service	
freq	97	10377	2525

1.3 Data cleaning

We call the `DataFrame.describe()` again. Notice that there are 33 unique labels/classes for object and 10 unique labels for action that the model will try to predict. But there are unwanted duplications e.g. `Idd,idd,lotalty_card,Lotalty_card`

Also note that, there are 13389 unique sentence utterances from 16175 utterances. You have to clean that too!

1.4 #TODO 0.1:

- You will have to remove unwanted label duplications as well as duplications in text inputs.
- Also, you will have to trim out unwanted whitespaces from the text inputs.

This shouldn't be too hard, as you have already seen it in the demo.

```
[7]: display(data_df.describe())
display(data_df.Object.unique())
display(data_df.Action.unique())
```

	Sentence Utterance	Action	Object
count	16175	16175	16175
unique	13389	10	33
top	enquire	service	
freq	97	10377	2525

```
array(['payment', 'package', 'suspend', 'internet', 'phone_issues',
       'service', 'nonTrueMove', 'balance', 'detail', 'bill', 'credit',
       'promotion', 'mobile_setting', 'iservice', 'roaming', 'truemoney',
       'information', 'lost_stolen', 'balance_minutes', 'idd',
       'TrueMoney', 'garbage', 'Payment', 'IDD', 'ringtone', 'Idd',
       'rate', 'loyalty_card', 'contact', 'officer', 'Balance', 'Service',
       'Loyalty_card'], dtype=object)
```

```
array(['enquire', 'report', 'cancel', 'Enquire', 'buy', 'activate',
       'request', 'Report', 'garbage', 'change'], dtype=object)
```

```
[8]: # TODO 1: Data Cleaning
```

```
# Filter cols
```

```
cols = ["Sentence Utterance", "Object"]
```

```
data_df = data_df[cols]
```

```
data_df.columns = ['input', 'raw_label']
```

```
# Lowercase: label
```

```
data_df['clean_label']=data_df['raw_label'].str.lower().copy()
```

```

data_df.drop('raw_label', axis=1, inplace=True)

# Trim white spaces: input
data_df['input'] = data_df['input'].str.strip()

# Remove duplicate: input
data_df = data_df.drop_duplicates(subset=['input'], keep='first')

# Display summary
display(data_df.describe())
display(data_df['clean_label'].unique())

```

		input	clean_label
count		13367	13367
unique		13367	26
top	<PHONE_NUMBER_REMOVED>	Counter...	service
freq		1	2108

```

array(['payment', 'package', 'suspend', 'internet', 'phone_issues',
       'service', 'nontruemove', 'balance', 'detail', 'bill', 'credit',
       'promotion', 'mobile_setting', 'iservice', 'roaming', 'truemoney',
       'information', 'lost_stolen', 'balance_minutes', 'idd', 'garbage',
       'ringtone', 'rate', 'loyalty_card', 'contact', 'officer'],
      dtype=object)

```

Split data into train, validation, and test sets (normally the ratio will be 80:10:10 , respectively). We recommend to use `train_test_split` from `scikit-learn` to split the data into train, validation, test set.

In addition, it should split the data that distribution of the labels in train, validation, test set are similar. There is **stratify** option to handle this issue.

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

Make sure the same data splitting is used for all models.

```

[9]: # Mapping
data = data_df.to_numpy()

unique_label = data_df.clean_label.unique()

label_2_num_map = dict(zip(unique_label, range(len(unique_label))))
num_2_label_map = dict(zip(range(len(unique_label)), unique_label))

print("Create Mappings")
display(num_2_label_map)
display(label_2_num_map)

print("Before Mappings")
display(data[:, 1])

```

```

data[:,1] = np.vectorize(label_2_num_map.get)(data[:,1]) # Mapping...
print("After Mappings")
display(data[:, 1])

# Trim
def strip_str(string):
    return string.strip()
print("Before")
print(data)
data[:,0] = np.vectorize(strip_str)(data[:,0]) # Trimming...
print("After")
print(data)

```

Create Mappings

```

{0: 'payment',
 1: 'package',
 2: 'suspend',
 3: 'internet',
 4: 'phone_issues',
 5: 'service',
 6: 'nontruemove',
 7: 'balance',
 8: 'detail',
 9: 'bill',
10: 'credit',
11: 'promotion',
12: 'mobile_setting',
13: 'iservice',
14: 'roaming',
15: 'truemoney',
16: 'information',
17: 'lost_stolen',
18: 'balance_minutes',
19: 'idd',
20: 'garbage',
21: 'ringtone',
22: 'rate',
23: 'loyalty_card',
24: 'contact',
25: 'officer'}

{'payment': 0,
 'package': 1,
 'suspend': 2,
 'internet': 3,
 'phone_issues': 4,
 'service': 5,
 'nontruemove': 6,

```

```

'balance': 7,
'detail': 8,
'bill': 9,
'credit': 10,
'promotion': 11,
'mobile_setting': 12,
'iservice': 13,
'roaming': 14,
'truemoney': 15,
'information': 16,
'lost_stolen': 17,
'balance_minutes': 18,
'idd': 19,
'garbage': 20,
'ringtone': 21,
'rate': 22,
'loyalty_card': 23,
'contact': 24,
'officer': 25}

```

Before Mappings

```

array(['payment', 'package', 'suspend', ..., 'balance', 'balance',
      'package'], dtype=object)

```

After Mappings

```

array([0, 1, 2, ..., 7, 7, 1], dtype=object)

```

Before

```

[['<PHONE_NUMBER_REMOVED>          Counter Services    3276.25
  3057.79    '
  0]
['internet          ' 1]
['          ' 2]
...
['          ' 7]
['          ' 7]
['          ' 1]]

```

After

```

[['<PHONE_NUMBER_REMOVED>          Counter Services    3276.25
  3057.79    '
  0]
['internet          ' 1]
['          ' 2]
...
['          ' 7]
['          ' 7]
['          ' 1]]

```

```

[10]: # TODO: Split data
from sklearn.model_selection import train_test_split

SEED = 42

def split_data(data_df, random_state=SEED):
    """split_data splits the data into train:validation:test=8:1:1 sets."""

    def _filter_data(data_df):
        X = data_df["input"]
        y = data_df["clean_label"]
        # Drop classes with fewer than 10(8:1:1) instances
        class_counts = y.value_counts()
        valid_classes = class_counts[class_counts >= 10].index
        filtered_data = data_df[data_df["clean_label"].isin(valid_classes)]
        # Update X and y after filtering
        X = filtered_data["input"]
        y = filtered_data["clean_label"]
        return X, y.astype(int)

    X, y = _filter_data(data_df)

    # First split: Train (80%) and Temp (20%)
    X_train, X_temp, y_train, y_temp = train_test_split(
        X, y, test_size=0.20, stratify=y, random_state=random_state
    )

    # Second split: Validation (10%) and Test (10%)
    X_val, X_test, y_val, y_test = train_test_split(
        X_temp, y_temp, test_size=0.50, stratify=y_temp,
        random_state=random_state
    )

    # Display dataset sizes
    print(f"Train size: {len(X_train)}")
    print(f"Validation size: {len(X_val)}")
    print(f"Test size: {len(X_test)}")

    return X_train, X_val, X_test, y_train, y_val, y_test

# Split
df = pd.DataFrame(data, columns=['input', 'clean_label'])
X_train, X_val, X_test, y_train, y_val, y_test = split_data(df)

```

Train size: 10690

Validation size: 1336

Test size: 1337

2 Model 1 TF-IDF

Build a model to train a tf-idf text classifier. Use a simple logistic regression model for the classifier.

For this part, you may find this [tutorial](#) helpful.

Below are some design choices you need to consider to accomplish this task. Be sure to answer them when you submit your model.

Q. What tokenizer will you use? Why?

Q. Will you ignore some stop words (a, an, the, to, etc. for English) in your tf-idf? Is it important? PythaiNLP provides a list of stopwords if you want to use (https://pythainlp.org/docs/2.0/api/corpus.html#pythainlp.corpus.common.thai_stopwords)

Q. The dictionary of TF-IDF is usually based on the training data. How many words in the test set are OOVs?

```
[11]: # TfidfVectorizer + LoRg
print("TfidfVectorizer + Logistic Regression")
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from pythainlp.corpus import thai_stopwords
import time

# Define Thai stopwords
thai_stopwords_list = list(thai_stopwords())

# Create a TF-IDF vectorizer
vectorizer = TfidfVectorizer(
    tokenizer=None, # Using default tokenizer
    stop_words=thai_stopwords_list, # Ignore Thai stopwords
    max_features=5000 # Limit vocabulary size
)

# Define the Logistic Regression model
model = LogisticRegression(random_state=SEED)

# Create a pipeline: TF-IDF transformation + Logistic Regression
text_clf = Pipeline([
    ('tfidf', vectorizer),
    ('clf', model)
])

# Train the model
start_time = time.time()
text_clf.fit(X_train, y_train) # training...
end_time = time.time()
print(f"Training time: {end_time - start_time:.4f} seconds")
```



```

# Predictions
y_pred_train = text_clf.predict(X_train)
y_pred_val = text_clf.predict(X_val)
y_pred_test = text_clf.predict(X_test)

# Evaluate model accuracy
train_acc = np.mean(y_train.astype(int) == y_pred_train)
val_acc = np.mean(y_val.astype(int) == y_pred_val)
test_acc = np.mean(y_test.astype(int) == y_pred_test)

print(f"Train Accuracy: {train_acc:.4f}")
print(f"Validation Accuracy: {val_acc:.4f}")
print(f"Test Accuracy: {test_acc:.4f}")

```

TfidfVectorizer + Logistic Regression
Training time: 1.7132 seconds
Train Accuracy: 0.7577
Validation Accuracy: 0.6287
Test Accuracy: 0.6185

```

[12]: # TfidfVectorizer + LoRg + pythainlp.tokenize
print("TfidfVectorizer + Logistic Regression + pythainlp.word_tokenize")
from pythainlp.tokenize import word_tokenize

# Define Thai stopwords
thai_stopwords_list = list(thai_stopwords())

# Define a custom tokenizer using pythainlp.word_tokenize
def thai_tokenizer(text):
    return word_tokenize(text, keep_whitespace=False)

# Create a TF-IDF vectorizer with the Thai tokenizer
vectorizer = TfidfVectorizer(
    tokenizer=thai_tokenizer, # Use pythainlp for tokenization
    stop_words=thai_stopwords_list, # Ignore Thai stopwords
    max_features=5000, # Limit vocabulary size
)

# Logistic Regression model
model = LogisticRegression(random_state=SEED)

# Pipeline: TF-IDF transformation + Logistic Regression
text_clf = Pipeline([("tfidf", vectorizer), ("clf", model)])

```

```

# Train the model
start_time = time.time()
text_clf.fit(X_train, y_train) # training...
end_time = time.time()
print(f"Training time: {end_time - start_time:.4f} seconds")

# Predictions
y_pred_train = text_clf.predict(X_train)
y_pred_val = text_clf.predict(X_val)
y_pred_test = text_clf.predict(X_test)

# Evaluate model accuracy
train_acc = np.mean(y_train.astype(int) == y_pred_train)
val_acc = np.mean(y_val.astype(int) == y_pred_val)
test_acc = np.mean(y_test.astype(int) == y_pred_test)

print(f"Train Accuracy: {train_acc:.4f}")
print(f"Validation Accuracy: {val_acc:.4f}")
print(f"Test Accuracy: {test_acc:.4f}")

```

TfidfVectorizer + Logistic Regression + pythainlp.word_tokenize
Training time: 1.5351 seconds
Train Accuracy: 0.7675
Validation Accuracy: 0.6894
Test Accuracy: 0.6911

Q. What tokenizer will you use? Why?

Ans: pythainlp.word_tokenize because it is specifically designed for Thai text processing. As shown in the results: - Higher accuracy for every sets. - Better generalization (Improved val_acc).

Q. Will you ignore some stop words (a, an, the, to, etc. for English) in your tf-idf? Is it important? PythaiNLP provides a list of stopwords if you want to use (https://pythainlp.org/docs/2.0/api/corpus.html#pythainlp.corpus.common.thai_stopwords)

Ans:

Yes, ignored Thai stopwords using pythainlp.thai_stopwords(), because it helps remove unnecessary words that don't contribute to classification, improving model efficiency.

Q. The dictionary of TF-IDF is usually based on the training data. How many words in the test set are OOVs?

```

[13]: test_words = set(" ".join(X_test).split())
      train_words = set(" ".join(X_train).split())
      oov_words = test_words - train_words
      print(f"OOV words in test set: {len(oov_words)}")

```

OOV words in test set: 2245