hw6-3-wanchanberta

February 14, 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, cancle) 2. Object Classification: Identify which object the customer is referring to (e.g. payment, truemoney, internet, roaming)

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

In this homework, you are asked 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 ask to submit 3 different files (.pdf from .ipynb) that does 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
```

```
--2025-02-14 07:03:53--  https://www.dropbox.com/s/37u83g55p19kvrl/clean-phone-
data-for-students.csv
Resolving www.dropbox.com (www.dropbox.com)… 162.125.3.18,
2620:100:6018:18::a27d:312
Connecting to www.dropbox.com (www.dropbox.com)|162.125.3.18|:443… connected.
HTTP request sent, awaiting response… 302 Found
Location: https://www.dropbox.com/scl/fi/8h8hvsw9uj6o0524lfe4i/clean-phone-data-
for-students.csv?rlkey=lwv5xbf16jerehnv3lfgq5ue6 [following]
--2025-02-14 07:03:53--
https://www.dropbox.com/scl/fi/8h8hvsw9uj6o0524lfe4i/clean-phone-data-for-
students.csv?rlkey=lwv5xbf16jerehnv3lfgq5ue6
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response… 302 Found
Location: https://uc279ae1207599e9d9326875105f.dl.dropboxusercontent.com/cd/0/in
```

```
line/CkHOW4KoKEeh3GbbXsnCY1KAfk4Byed22ey5qjE_wN9TVvHj71kfAdPkDI7F7HLgZNUi_w5EgTm
OKEajEE9E01F2yz52dY8VZcxwtbc5o0BsELsNWccsg8xqOJAnJObI7SI/file# [following]
--2025-02-14 07:03:54--  https://uc279ae1207599e9d9326875105f.dl.dropboxusercont
ent.com/cd/0/inline/CkHOW4KoKEeh3GbbXsnCY1KAfk4Byed22ey5qjE_wN9TVvHj71kfAdPkDI7F
7HLgZNUi_w5EgTmOKEajEE9E01F2yz52dY8VZcxwtbc5o0BsELsNWccsg8xqOJAnJObI7SI/file
Resolving uc279ae1207599e9d9326875105f.dl.dropboxusercontent.com
(uc279ae1207599e9d9326875105f.dl.dropboxusercontent.com)… 162.125.3.15,
2620:100:6018:15::a27d:30f
Connecting to uc279ae1207599e9d9326875105f.dl.dropboxusercontent.com
(uc279ae1207599e9d9326875105f.dl.dropboxusercontent.com)|162.125.3.15|:443…
connected.
HTTP request sent, awaiting response… 200 OK
Length: 2518977 (2.4M) [text/plain]
Saving to: 'clean-phone-data-for-students.csv'

clean-phone-data-fo 100%[===================>]   2.40M  --.-KB/s    in 0.07s

2025-02-14 07:03:54 (32.7 MB/s) - 'clean-phone-data-for-students.csv' saved
[2518977/2518977]
```

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

```
Collecting pythainlp
  Downloading pythainlp-5.0.5-py3-none-any.whl.metadata (7.5 kB)
Requirement already satisfied: requests>=2.22.0 in
/usr/local/lib/python3.10/dist-packages (from pythainlp) (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.22.0->pythainlp)
(3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests>=2.22.0->pythainlp) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.22.0->pythainlp)
(2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.22.0->pythainlp)
(2025.1.31)
Downloading pythainlp-5.0.5-py3-none-any.whl (17.9 MB)
                         17.9/17.9 MB
88.7 MB/s eta 0:00:00:00:0100:01
Installing collected packages: pythainlp
Successfully installed pythainlp-5.0.5
Note: you may need to restart the kernel to use updated packages.
```

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')
     import time
```

```
[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 unqiue 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                         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, valdation, and test sets (normally the ratio will be 80:10:10 , respectively). We recommend to use train_test_spilt 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
```

```python
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]:
```python
def filter_data(df):
    class_counts = df["label"].value_counts()
    valid_classes = class_counts[class_counts >= 10].index
    invalid_classes = class_counts[class_counts < 10].index
    correct_classes = [i for i, _ in enumerate(valid_classes.sort_values())]
    change_map = dict(zip(valid_classes.sort_values(), correct_classes))

    # drop invalid classes
    df = df[df["label"].isin(valid_classes)]
    for num in invalid_classes:
        label = num_2_label_map[num]
        del num_2_label_map[num]
        del label_2_num_map[label]

    # update change
    df["label"] = df["label"].apply(lambda x: change_map.get(x, -1))
    for old_num, new_num in change_map.items():
        num_2_label_map[new_num] = num_2_label_map.pop(old_num)
        label_2_num_map[num_2_label_map[new_num]] = new_num
    return df

df = pd.DataFrame(data, columns=['input', 'label'])
df_filtered = filter_data(df)
```

[11]:
```
# TODO: Split data
from sklearn.model_selection import train_test_split

def split_data(data_df, random_state=SEED):
    """split_data splits the data into train:validation:test=8:1:1 sets."""
    X = data_df["input"]
    y = data_df["label"]

    # 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
```

```
X_train, X_val, X_test, y_train, y_val, y_test = split_data(df_filtered)
```

```
Train size: 10690
Validation size: 1336
Test size: 1337
```

2 Model 3 WangchanBERTa

We ask you to train a WangchanBERTa-based model.

We recommend you use the thaixtransformers fork (which we used in the PoS homework). https://github.com/PyThaiNLP/thaixtransformers

The structure of the code will be very similar to the PoS homework. You will also find the huggingface tutorial useful. Or you can also add a softmax layer by yourself just like in the previous homework.

Which WangchanBERTa model will you use? Why? (Don't forget to clean your text accordingly).

Ans: `airesearch/wangchanberta-base-att-spm-uncased` because - specifically trained for Thai text. - SentencePiece tokenization, which is ideal for Thai, unlike space-based tokenization. - It has state-of-the-art performance for Thai text classification tasks.

[12]:
```
# Data preprocessing
for i in range(len(data)):
    data[i][0] = data[i][0].replace(' ', " ")
```

[13]:
```
from kaggle_secrets import UserSecretsClient
user_secrets = UserSecretsClient()
secret_value_0 = user_secrets.get_secret("WANDB_API_KEY")

import wandb
wandb.login(key=secret_value_0)
```

```
wandb: Using wandb-core as the SDK backend.  Please refer to
https://wandb.me/wandb-core for more information.
wandb: Currently logged in as: pupipat-sk
(pupipatsk). Use `wandb login --relogin` to force relogin
wandb: WARNING If you're specifying your api key in code,
ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY
environment variable, or running `wandb login` from the command line.
wandb: Appending key for api.wandb.ai to your netrc file:
/root/.netrc
```

[13]: True

[14]:
```
import torch
def find_device() -> str:
    device = "cpu"
    if torch.cuda.is_available():
        device = "cuda"
    elif torch.backends.mps.is_available():
        device = "mps"
    else:
        device = "cpu"
    print(f"device: {device}")
    return device
device = find_device()
```

```
device: cuda
```

[ ]:

```
import numpy as np
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification,␣
 ↪Trainer, TrainingArguments, DataCollatorWithPadding
from datasets import Dataset
from sklearn.metrics import accuracy_score

start_time = time.time()

# Define the WangchanBERTa model
MODEL_NAME = "airesearch/wangchanberta-base-att-spm-uncased"
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

# Tokenization function with dynamic padding
def tokenize_function(examples):
    return tokenizer(examples["text"], truncation=True)  # Removed "max_length"␣
 ↪and fixed padding

# Convert data into Hugging Face Dataset format
train_data = Dataset.from_dict({"text": X_train.tolist(), "label": y_train.
 ↪tolist()})
val_data = Dataset.from_dict({"text": X_val.tolist(), "label": y_val.tolist()})
test_data = Dataset.from_dict({"text": X_test.tolist(), "label": y_test.
 ↪tolist()})

# Tokenize datasets
train_dataset = train_data.map(tokenize_function, batched=True)
val_dataset = val_data.map(tokenize_function, batched=True)
test_dataset = test_data.map(tokenize_function, batched=True)

# Load pre-trained WangchanBERTa model for classification
model = AutoModelForSequenceClassification.from_pretrained(MODEL_NAME,
                                                                        ␣
 ↪num_labels=len(num_2_label_map),
                                                                  ␣
 ↪id2label=num_2_label_map,
                                                                  ␣
 ↪label2id=label_2_num_map)
model.to(device)

# Define training arguments
EPOCHS = 5
BATCH_SIZE = 32
training_args = TrainingArguments(
    output_dir="./results",
    learning_rate=2e-5,
    per_device_train_batch_size=BATCH_SIZE,
    per_device_eval_batch_size=BATCH_SIZE,
    num_train_epochs=EPOCHS,
    weight_decay=0.01,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    push_to_hub=False,
    logging_dir="./logs",
    logging_steps=500,
)

# Compute accuracy
def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    preds = np.argmax(predictions, axis=1)
    return {"accuracy": accuracy_score(labels, preds)}

# Use dynamic padding
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

# Define trainer with dynamic padding
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics,
    data_collator=data_collator  # Enables dynamic padding
)

# Train model...
print("Start training...")
trainer.train()
print("Finish training.")
```

```python
# Evaluate model on validation and test sets
train_results = trainer.evaluate(train_dataset)
val_results = trainer.evaluate(val_dataset)
test_results = trainer.evaluate(test_dataset)

print(f"Train Accuracy: {train_results['eval_accuracy']:.4f}")
print(f"Validation Accuracy: {val_results['eval_accuracy']:.4f}")
print(f"Test Accuracy: {test_results['eval_accuracy']:.4f}")
end_time = time.time()
print(f"Total Time: {end_time - start_time:.4f} seconds")
```

```
tokenizer_config.json:   0%|          | 0.00/282 [00:00<?, ?B/s]

config.json:   0%|          | 0.00/546 [00:00<?, ?B/s]

sentencepiece.bpe.model:   0%|          | 0.00/905k [00:00<?, ?B/s]

Map:   0%|          | 0/10690 [00:00<?, ? examples/s]

Asking to truncate to max_length but no maximum length is provided and the model
has no predefined maximum length. Default to no truncation.

Map:   0%|          | 0/1336 [00:00<?, ? examples/s]

Map:   0%|          | 0/1337 [00:00<?, ? examples/s]

model.safetensors:   0%|          | 0.00/423M [00:00<?, ?B/s]

Some weights of CamembertForSequenceClassification were not initialized from the
model checkpoint at airesearch/wangchanberta-base-att-spm-uncased and are newly
initialized: ['classifier.dense.bias', 'classifier.dense.weight',
'classifier.out_proj.bias', 'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
wandb: WARNING The `run_name` is currently set to the same
value as `TrainingArguments.output_dir`. If this was not intended, please
specify a different run name by setting the `TrainingArguments.run_name`
parameter.

Start training…

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

[17]:
```
# Evaluate model on validation and test sets
train_results = trainer.evaluate(train_dataset)
val_results = trainer.evaluate(val_dataset)
test_results = trainer.evaluate(test_dataset)

print(f"Train Accuracy: {train_results['eval_accuracy']:.4f}")
print(f"Validation Accuracy: {val_results['eval_accuracy']:.4f}")
print(f"Test Accuracy: {test_results['eval_accuracy']:.4f}")
```

```
Train Accuracy: 0.7889
Validation Accuracy: 0.7657
Test Accuracy: 0.7450
```

3 Comparison

After you have completed the 3 models, compare the accuracy, ease of implementation, and inference speed (from cleaning, tokenization, till model compute) between the three models in mycourseville.

Model 1: TfidfVectorizer + Logistic Regression + pythainlp.word_tokenize - Training time: 1.5351 seconds - Train Accuracy: 0.7675 - Validation Accuracy: 0.6894 - Test Accuracy: 0.6911

Model 2: MUSE + Logistic Regression - Encoding Time: 47.9169 seconds - Training Time: 25.3683 seconds - Train Accuracy: 0.7367 - Validation Accuracy: 0.7066 - Test Accuracy: 0.7016 - Total Time: 81.2206 seconds

Model 3: WangchanBERTa - Train Accuracy: 0.7889 - Validation Accuracy: 0.7657 - Test Accuracy: 0.7450 - Total Time: 5 min 10 sec

Based on the performance of the three models, which one do you think is best for this use case (Callcenter Chatbot)?

Answer: WangchanBERTa

because we want CallcenterChatbot that doesn't need that fast respond(compare to current answer bot when we call for customer service) - Highest accuracy, callcenter question can be repetitive question - Slow inference time, but can speedup by increase computation - Better generalization - Deep contextual understanding for handles nuances and variations in Thai language