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.

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

```
--2025-02-15 17:48:01--
https://www.dropbox.com/s/37u83g55p19kvrl/clean-phone-data-for-
students.csv
Resolving www.dropbox.com (www.dropbox.com)... 162.125.1.18,
2620:100:6016:18::a27d:112
Connecting to www.dropbox.com (www.dropbox.com)|162.125.1.18|: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]
--2025-02-15 17:48:01--
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://ucf7cfebc37547e840b4907af352.dl.dropboxusercontent.com/cd/0/
inline/
CkLXTGTTsPIvZrDlmR9Omr_SLvsQ8MPpi0L6OgiT8lVd90KrH9gwzq7iswv2Whu7T0FnU9
D4Ma4nqkK6-YeJ_dUKI_BG_wnrlqRXjoGs6PI8u_YSHuB72Dq87BlEE13c_S8/file#
[following]
--2025-02-15 17:48:02--
https://ucf7cfebc37547e840b4907af352.dl.dropboxusercontent.com/cd/0/
inline/
CkLXTGTTsPIvZrDlmR9Omr_SLvsQ8MPpi0L6OgiT8lVd90KrH9gwzq7iswv2Whu7T0FnU9
D4Ma4nqkK6-YeJ_dUKI_BG_wnrlqRXjoGs6PI8u_YSHuB72Dq87BlEE13c_S8/file
Resolving ucf7cfebc37547e840b4907af352.dl.dropboxusercontent.com
(ucf7cfebc37547e840b4907af352.dl.dropboxusercontent.com)...
162.125.1.15, 2620:100:6016:15::a27d:10f
Connecting to ucf7cfebc37547e840b4907af352.dl.dropboxusercontent.com
(ucf7cfebc37547e840b4907af352.dl.dropboxusercontent.com)|
162.125.1.15|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2518977 (2.4M) [text/plain]
Saving to: ‘clean-phone-data-for-students.csv.1’

clean-phone-data-fo 100%[===================>]   2.40M  --.-KB/s    in
0.04s

2025-02-15 17:48:02 (53.9 MB/s) - ‘clean-phone-data-for-
students.csv.1’ saved [2518977/2518977]
```

```
!pip install pythainlp
```

```
Requirement already satisfied: pythainlp in
/usr/local/lib/python3.10/dist-packages (5.0.5)
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)
```

Import Libs

```
%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

from kaggle_secrets import UserSecretsClient
secret_label = "wandb_api_key"
secret_value = UserSecretsClient().get_secret(secret_label)

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

```
wandb: Using wandb-core as the SDK backend.  Please refer to
https://wandb.me/wandb-core for more information.
wandb: Currently logged in as: nacnano (nacnano2). 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
```

```
True
```

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

```
def clean_data(df):
    """Cleans the dataset by selecting relevant columns, normalizing
labels,
    trimming whitespace, and removing duplicates."""

    # Select and rename columns
    df = df[["Sentence Utterance",
"Object"]].rename(columns={"Sentence Utterance": "input", "Object":
"raw_label"})

    # Normalize label (lowercase)
    df["clean_label"] = df["raw_label"].str.lower()

    # Trim white spaces in input column
    df["input"] = df["input"].str.strip()
    # Remove duplicates based on input
    df = df.drop_duplicates(subset="input", keep="first")

    # Drop the raw label column
    df.drop(columns=["raw_label"], inplace=True)

    return df

# Apply cleaning function
data_df = clean_data(data_df)

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

```
                                       input clean_label
count                                  13367       13367
unique                                 13367          26
top     สอบถามโปรโมชั่นปัจจุบันที่ใช้อยู่ค่ะ   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)
```

```
# Mapping and Trimming
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))

data[:,1] = np.vectorize(label_2_num_map.get)(data[:,1])

def strip_str(string):
    return string.strip()
data[:,0] = np.vectorize(strip_str)(data[:,0])

display(data)
```

```
array([['<PHONE_NUMBER_REMOVED> ผมไปจ่ายเงินที่ Counter Services เค้าเช็ต
3276.25 บาท เมื่อวานที่ผมเช็คที่ศูนย์บอกมียอด 3057.79 บาท',
        0],
       ['internet ยังความเร็วอยุ่เท่าไหร ครับ', 1],
       ['ตะกี้ไปชำระค่าบริการไปแล้ว แต่ยังใช้งานไม่ได้ ค่ะ', 2],
       ...,
       ['ยอดเงินเหลือเท่าไหร่ค่ะ', 7],
       ['ยอดเงินในระบบ', 7],
       ['สอบถามโปรโมชั่นปัจจุบันที่ใช้อยู่ค่ะ', 1]], dtype=object)

# Split
from sklearn.model_selection import train_test_split

# Constants
SEED = 42
MIN_INSTANCES = 10  # Minimum instances per class


def filter_data(data_df, min_instances=MIN_INSTANCES):
    """
    Filters classes with fewer than `min_instances` occurrences.
    Returns filtered input (X) and labels (y).
    """
    class_counts = data_df["clean_label"].value_counts()
    valid_classes = class_counts[class_counts >= min_instances].index

    filtered_data =
data_df[data_df["clean_label"].isin(valid_classes)]
    return filtered_data["input"],
filtered_data["clean_label"].astype(int)

def split_data(data_df, random_state=SEED,
min_instances=MIN_INSTANCES):
    """
    Splits data into train (80%), validation (10%), and test (10%)
sets.
    Ensures stratification and filtering of rare classes.
    """
    # Filter classes
    X, y = filter_data(data_df, min_instances)

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

    # Split 10% Validation, 10% Test
    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
    )

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

    return (
        np.array(X_train), np.array(X_val), np.array(X_test),
        np.array(y_train), np.array(y_val), np.array(y_test)
    )

# Convert to DataFrame
df = pd.DataFrame(data, columns=['input', 'clean_label'])

# Split dataset
X_train, X_val, X_test, y_train, y_val, y_test = split_data(df)
```

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

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: I will use airesearch/wangchanberta-base-att-spm-uncased because:

- It is specifically trained for Thai text, making it well-suited for Thai NLP tasks.
- It uses SentencePiece tokenization, which is more effective for Thai than space-based tokenization.
- It achieves state-of-the-art performance for Thai text classification tasks.

```
for i in range(len(data)):
    data[i][0] = data[i][0].replace('ำ', "ํา")

device = "cuda"

import time
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()

MODEL_NAME = "airesearch/wangchanberta-base-att-spm-uncased"
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

def tokenize_function(examples):
    return tokenizer(examples["text"], truncation=True)

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()})

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)

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)

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,
)

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

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics,
    data_collator=data_collator
)

trainer.train()

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")
```

```
{"model_id":"55ea3540a727436787ee34d3c404f852","version_major":2,"version_minor":0}

{"model_id":"fa773fd42be8403282f7eb69ef3c09b2","version_major":2,"version_minor":0}

{"model_id":"660dcd1b021d4449a05a3490f554e038","version_major":2,"version_minor":0}

{"model_id":"4a55f96b3aa24c37a958355319b5757a","version_major":2,"version_minor":0}
```

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

```
{"model_id":"837981bd023a4fddb829abe30d4c802f","version_major":2,"version_minor":0}

{"model_id":"843d62ab99544e34b3a5977520d59360","version_major":2,"version_minor":0}

{"model_id":"ba07c9372475420184a2970124361f76","version_major":2,"version_minor":0}
```

```
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.
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:
1575: FutureWarning: `evaluation_strategy` is deprecated and will be
removed in version 4.46 of 🤗 Transformers. Use `eval_strategy` instead
  warnings.warn(
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.

<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>

/usr/local/lib/python3.10/dist-packages/torch/nn/parallel/
_functions.py:71: UserWarning: Was asked to gather along dimension 0,
but all input tensors were scalars; will instead unsqueeze and return
a vector.
  warnings.warn(

<IPython.core.display.HTML object>

/usr/local/lib/python3.10/dist-packages/torch/nn/parallel/
_functions.py:71: UserWarning: Was asked to gather along dimension 0,
but all input tensors were scalars; will instead unsqueeze and return
a vector.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/nn/parallel/_functions.p
y:71: UserWarning: Was asked to gather along dimension 0, but all
input tensors were scalars; will instead unsqueeze and return a
vector.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/nn/parallel/_functions.p
y:71: UserWarning: Was asked to gather along dimension 0, but all
input tensors were scalars; will instead unsqueeze and return a
vector.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/nn/parallel/_functions.p
y:71: UserWarning: Was asked to gather along dimension 0, but all
input tensors were scalars; will instead unsqueeze and return a
vector.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/nn/parallel/_functions.p
y:71: UserWarning: Was asked to gather along dimension 0, but all
input tensors were scalars; will instead unsqueeze and return a
vector.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/nn/parallel/_functions.p
y:71: UserWarning: Was asked to gather along dimension 0, but all
input tensors were scalars; will instead unsqueeze and return a
vector.
  warnings.warn(
```

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

```
/usr/local/lib/python3.10/dist-packages/torch/nn/parallel/
_functions.py:71: UserWarning: Was asked to gather along dimension 0,
but all input tensors were scalars; will instead unsqueeze and return
a vector.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/nn/parallel/_functions.p
y:71: UserWarning: Was asked to gather along dimension 0, but all
input tensors were scalars; will instead unsqueeze and return a
vector.
  warnings.warn(
```

```
Train Accuracy: 0.7821
Validation Accuracy: 0.7545
Test Accuracy: 0.7472
Total Time: 369.9648 seconds
```

```python
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}")
```

```
/usr/local/lib/python3.10/dist-packages/torch/nn/parallel/
_functions.py:71: UserWarning: Was asked to gather along dimension 0,
but all input tensors were scalars; will instead unsqueeze and return
a vector.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/nn/parallel/_functions.p
y:71: UserWarning: Was asked to gather along dimension 0, but all
input tensors were scalars; will instead unsqueeze and return a
vector.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/nn/parallel/_functions.p
y:71: UserWarning: Was asked to gather along dimension 0, but all
input tensors were scalars; will instead unsqueeze and return a
```

```
vector.
  warnings.warn(

Train Accuracy: 0.7821
Validation Accuracy: 0.7545
Test Accuracy: 0.7472
```

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.

TF-IDF

Training time: 2.8185 seconds Train Accuracy: 0.7675 Validation Accuracy: 0.6886 Test Accuracy: 0.6933

MUSE

Encoding Time: 21.6718 seconds Training Time: 2.2648 seconds Train Accuracy: 0.7373 Validation Accuracy: 0.7073 Test Accuracy: 0.7023 Total Time: 38.5585 seconds

wangchanberta

Train Accuracy: 0.7821 Validation Accuracy: 0.7545 Test Accuracy: 0.7472 Total Time: 369.9648 seconds

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

A: WangchanBERTa

- Highest accuracy: Call center questions are often repetitive, so accuracy is a priority.
- Better generalization: It can handle a variety of customer queries effectively.
- Deep contextual understanding: Helps interpret nuances and variations in Thai language.
- Inference speed trade-off: While inference is slower than traditional models, it can be optimized with increased computation, and real-time speed is not as critical as accuracy in this case.