Introduction to Artificial Intelligence
SWE3011_41
Fall 2023

Project
Anonymous ACL submission

Sungkyunkwan University

November 30, 2023

Abstract

Within the context of this assignment, a substantial hands-on encounter with applications of artificial intelligence will be attained, with a particular focus on two pivotal tasks: traditional supervised learning methodologies and prompt engineering. Upon the culmination of this assignment, an adept understanding of the practical implementation of artificial intelligence techniques for the purpose of classification shall be cultivated. Furthermore, through collaborative participation in the curation of prompts and the execution of experiments, the acquisition of insights and skills that are notably advantageous for a group project is anticipated.

1 Introduction

In this lab journal, the primary objective is to offer a comprehensive practical experience in the field of artificial intelligence, with a specific focus on supervised learning. Upon the conclusion of this project, a solid and pragmatic understanding of traditional machine learning techniques for text classification is expected to be acquired. Furthermore, the utilization of Langchain for prompt engineering tasks, especially in the context of Large Language Models (LLMs), will have been explored. This multifaceted approach will equip us with a versatile skill set and a deeper insight into the practical aspects of artificial intelligence.
This assignment categorizes tasks into three main problems:

1. **Supervised Text Classification using Traditional Machine Learning Methods:** The task involves implementing a text classification process using traditional machine learning methods. A dataset and skeleton code utilizing the scikit-learn library, with specific sections left blank for completion, will be provided. The responsibility is to finalize the code, conduct experiments, and evaluate to derive results.

2. **Prompt Engineering for LLMs via Langchain Framework:** Using Langchain, a tool designed for crafting prompts for language models, a series of prompts will be created to elicit specific and varied responses. Baseline code for conducting prompt engineering experiments with ChatGPT and Large Language Models on Hugging Face Hub is provided. The task involves completing the code and optimizing the prompts for sentiment classification.

3. **LATEX:** The third focus of this assignment is report writing using LATEX, a document preparation system that aids in creating structured documents. This part helps students develop skills in professional communication and documentation within the AI field.

This journal serves as a documentation of the academic journey in addressing these AI challenges and acquiring practical skills in the field of artificial intelligence.

2 Dataset

The GLUE benchmark stands as a compendium of diverse Natural Language Processing (NLP) tasks, employed to assess the efficacy of language models. A significant component of the GLUE benchmark, known as Stanford Sentiment Treebank V2 (SST-2), is dedicated to evaluating the performance of models in sentiment classification on the SST-2 dataset and other related tasks, serving as a litmus test for their broader language comprehension and reasoning capabilities (Wang et al., 2018).

The SST-2 dataset is a renowned benchmark dataset in the realm of NLP, frequently employed for sentiment analysis undertakings. An extension of the Stanford Sentiment Treebank (SST), which originally featured fine-grained sentiment labels encompassing categories such as very positive, positive, neutral, negative, and very negative. SST-2 consists of sentences sourced from movie reviews, each meticulously annotated with binary labels signifying either positive or negative sentiment (Socher et al., 2013).

3 Methology

For both Task 1 and Task 2, distinct methods have been chosen to address their specific objectives.

For Task 1, which centers on supervised text classification, the chosen trio of methods comprises logistic regression, random forest, and naive Bayes classifiers. These algorithms were meticulously selected for their suitability and proven effectiveness in tackling text classification challenges.

For Task 2, which focuses on prompt engineering for Large Language Models, LangChain was chosen as the tool to design both zero-shot and few-shot prompts. These prompts were to be tested using ChatGPT and the Large Language Models available on the Hugging Face Hub in order to solve the sentiment classification problem on the SST-2 dataset.
It's important to note that, for this task, the decision was made to abstain from using OpenAI models. Instead, the focus was solely on experimenting with freely available models on the Hugging Face Hub. This approach allowed exploration of the capabilities of large language models within the constraints of free resources.

4 Experiments

4.0.1 Task 1

In the experimental phase of this project, a comprehensive analysis was undertaken to evaluate the effectiveness of three distinct machine learning models: Logistic Regression, Random Forest, and the Naive Bayes Classifier. The approach adopted was two-fold, entailing not only model selection but also the intricate process of hyperparameter tuning and cross-validation. In this context, the Logistic Regression model demonstrated remarkable performance, optimizing its hyperparameters to yield the most accurate results. Specifically, the observation revealed that a regularization strength parameter, denoted as 'C,' was fine-tuned to the value of 100, and the regularization term 'penalty' was set to 'l2'. In essence, 'C' was controlled to determine the degree of regularization, with larger values signifying less regularization and a potentially higher sensitivity to the training data, while 'penalty' indicated the type of regularization applied, with 'l2' corresponding to the L2 penalty. On the other hand, when the Random Forest model was employed, the hyperparameter optimization revealed that the ideal configuration consisted of a maximum depth set to 'None' and an ensemble size, or 'n_estimators,' fixed at 200. Here, 'max_depth' signified the depth of each tree in the Random Forest, with 'None' indicating no restrictions on the depth. As for 'n_estimators,' it represented the number of trees within the ensemble. Lastly, in the case of the Naive Bayes Classifier, it was determined that the hyperparameter 'alpha' was most effective at a value of 0.1, indicating the Laplace smoothing factor for the model. Through conducting this detailed experimentation and elucidating the significance of these parameter choices, the objective was not only to optimize model performance but also to gain a profound understanding of the underlying mechanisms that govern these machine learning algorithms.

4.0.2 Task 2

For the experimental part of Task 2, both zero-shot and few-shot prompts were prepared in order to utilize the Hugging Face Hub's Large Language Models' ability to perform sentiment analysis of the SST-2 dataset. "Zero-shot prompt" refers to prompt engineering where the model is not trained with examples on how to perform the given task, relying only on its existing knowledge, while "few-shot prompt" refers to prompts where the model is trained with specific examples on how to complete the task. For both the zero-shot and few-shot prompts, the model was given the following prefix, as the instruction for its task of sentiment classification:

> *"You are a sentiment classifier. Classify the given text input as "positive" or "negative" based on its tone. Please read each sentence carefully and determine its sentiment based on common definitions.*
>
> - *<positive>: A positive sentiment indicates a favorable or optimistic tone. It expresses approval, satisfaction, or positive emotions.*
> - *<negative>: A negative sentiment indicates an unfavorable or pessimistic tone. It expresses disapproval, dissatisfaction, or negative emotions.*
>
> *Please read each sentence and classify it as either "positive" or "negative" based on the provided definitions."*

Apart from the prefix that serves as the task description, for the few-shot prompt, the model was trained with a provided dataset that shares similar characteristics to the SST-2 dataset. Due to the constraints of using only free resources, it was not possible to provide the Large Language Model with more than 27 training examples per execution. Since the training dataset was larger than that, a different set of examples was used for each execution. Each of the examples in the set was selected randomly out of all the available ones in the dataset; however, in order to maximize the accuracy of the Large Language Model predictions, a constraint was applied to maintain consistent proportions of positive and negative instances in each set. This approach was taken to prevent any imbalance between positive and negative examples in the generated sets, which could impact the model's training. After executing the prompts multiple times, it became apparent that the accuracy of few-shot prompts varied with each iteration due to changing examples influencing the model. Consequently, three iterations were conducted to capture the accuracy range, constrained by limited resources. In the case of zero-shot prompts, a single iteration was performed for assessment. The resulting accuracy of the zero-shot prompt and the accuracy ranges of the few-shot prompts were compared in chapters "Results" and "Discussions".

5 Results

5.0.1 Task 1

For the logistic regression model, the achieved accuracy was 0.77. The classification report demonstrated that for class 0, the precision was 0.83, recall was 0.72, and the F1-score was 0.77, with a support of 54 instances. For class 1, the precision was 0.72, recall was 0.83, and the F1-score was 0.77, with a support of 46 instances. The overall accuracy for this model was 0.77.

```
Accuracy: 0.77
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.72      0.77        54
           1       0.72      0.83      0.77        46

    accuracy                           0.77       100
   macro avg       0.77      0.77      0.77       100
weighted avg       0.78      0.77      0.77       100
```

Figure 1: Result of Logistic Regression model

In the case of the random forest model, the accuracy obtained was 0.71. The classification report provided the following results: for class 0, the precision was 0.75, recall was 0.70, and the F1-score was 0.72, with a support of 54 instances. For class 1, the precision was 0.67, recall was 0.72, and the F1-score was 0.69, with a support of 46 instances. The overall accuracy for this model was 0.71.

```
Accuracy: 0.71
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.70      0.72        54
           1       0.67      0.72      0.69        46

    accuracy                           0.71       100
   macro avg       0.71      0.71      0.71       100
weighted avg       0.71      0.71      0.71       100
```

Figure 2: Result of Random Forest model

Regarding the naive Bayes model, the achieved accuracy was 0.70. The classification report revealed that for class 0, the precision was 0.74, recall was 0.69, and the F1-score was 0.71, with a support of 54 instances. For class 1, the precision was 0.66, recall was 0.72, and the F1-score was 0.69, with a support of 46 instances. The overall accuracy for this model was 0.70.

```
Accuracy: 0.70
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.69      0.71        54
           1       0.66      0.72      0.69        46

    accuracy                           0.70       100
   macro avg       0.70      0.70      0.70       100
weighted avg       0.70      0.70      0.70       100
```

Figure 3: Result of Naive Bayes model

5.0.2 Task 2

There was a significant difference between the accuracy of sentiment classification obtained from the Large Language Models from the zero-shot and few-shot prompts. The zero-shot prompt had a prediction accuracy of 51%, while the few-shot prompt yielded a prediction accuracy in the range of 82-89%. The most important thing to note is the greater accuracy of the few-shot prompt for the task of sentiment classification. Additionally, the figures are included in the appendix for more detailed examination.

```
100%|██████████| 100/100 [00:07<00:00, 14.10it/s]
Predicted Categories: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
Actual Categories: [1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0,
Accuracy: 51.00%
```

Figure 4: Result of prediction using zero-shot prompt

```
100%|██████████| 100/100 [01:07<00:00,  1.49it/s]
Predicted Categories: [1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 1, 0, -1, 1,
Actual Categories: [1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0,
Accuracy: 83.00%
```

Figure 5: Result of prediction with the first execution of few-shot prompt

```
Iteration 1:
100%|██████████| 100/100 [00:28<00:00,  3.56it/s]
Predicted Categories: [0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
Actual Categories: [1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1,
Accuracy: 89.00%

Iteration 2:
100%|██████████| 100/100 [00:28<00:00,  3.49it/s]
Predicted Categories: [0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
Actual Categories: [1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1,
Accuracy: 87.00%

Iteration 3:
100%|██████████| 100/100 [00:31<00:00,  3.13it/s]Predicted Categories: [0, 1, 0, 0, 1, 1, 1, -1, 1, -1, 0, 0, 0, 0, -1, 0, -1, 1, 0, 0, 0,
Actual Categories: [1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1,
Accuracy: 82.00%

All Accuracies: [0.89, 0.87, 0.82]
```

Figure 6: Result of prediction with subsequent executions of few-shot prompt

6 Discussions

6.0.1 Task 1

Accuracy was chosen as the primary evaluation metric for Task 1, aligning with the approach taken in the GLUE benchmark. To evaluate the models, the provided test_dataset was employed.

It is evident that the logistic regression model outperformed the other two models, namely random forest and naive Bayes, with the highest accuracy of 0.77. The logistic regression model exhibited a balanced performance in terms of precision, recall, and F1-scores for both classes 0 and 1. This indicates its effectiveness in handling text classification tasks.

On the other hand, the random forest model achieved an accuracy of 0.71. While it provided competitive results, its performance was slightly lower compared to logistic regression. The classification report demonstrated reasonable precision, recall, and F1-scores for both classes, making it a viable choice but not the top-performing one.

Lastly, the naive Bayes model exhibited an accuracy of 0.70, which was consistent with random forest but slightly lower than logistic regression. The classification report also showed balanced precision, recall, and F1-scores for both classes.

6.0.2 Task 2

Accuracy was also chosen as the primary evaluation metric for Task 2, in order to stay consistent with the findings of Task 1, as well as provide insight into the capability of the Large Language Model to solve the sentiment classification problem on the SST-2 dataset.

The most important result was the difference in accuracy when employing few-shot prompts instead of zero-shot prompts. This was consistent with our expectation that sentiment classification yields better results when the model is trained with specific examples rather than when the task is simply performed based on the model's pre-existing knowledge. This poses the idea that few-shot prompts are superior for solving sentiment classification problems. Because of this, choosing a proper dataset and improving the quality, diversity and proportionality of positive and negative emotion examples is an important part of solving problems like this. Lastly, while there is a positive correlation between the amount of examples provided and the accuracy of the predictions, it is important to not overtune the model with an excessive amount of examples.

7 Conclusion

7.0.1 Task 1

In conclusion, this project delved into the application of various machine learning models, including logistic regression, random forest, and naive Bayes, for the task of text classification. Through rigorous experimentation and evaluation, it was observed that the logistic regression model emerged as the top-performing model, achieving the highest accuracy and demonstrating balanced precision, recall, and F1-scores for both classes. This emphasizes the significance of selecting an appropriate machine learning model tailored to the task at hand.

Despite the success of the logistic regression model, there are potential areas for improvement. First and foremost, further exploration of feature engineering techniques and text preprocessing methods could enhance the performance of all models. Additionally, the incorporation of more advanced machine learning models or deep learning architectures may lead to improved results in text classification tasks. Furthermore, the dataset size and diversity could be increased to bolster the models' ability to generalize across different textual data.

Moreover, an investigation into hyperparameter tuning techniques, such as grid search or Bayesian optimization, could be beneficial to fine-tune the models further. Finally, ensembling methods, which combine the strengths of multiple models, may provide an opportunity to boost overall performance.

In summary, while logistic regression proved to be the most effective model in this specific context, there remains room for enhancements through various avenues, including feature engineering, model selection, and hyperparameter tuning. This project serves as a valuable foundation for future research and applications in the field of text classification and natural language processing.

7.0.2 Task 2

In conclusion, Task 2 of this project allowed us to explore prompt engineering and how it is a fundamental part of AI projects. In this case, a sentiment classification problem was solved through zero-shot and few-shot prompts. It was undeniable that the few-shot prompt performed significantly better than the zero-shot prompt for this task. Not only that, but is also relevant to point out that the few-shot prompt achieved a range of accuracy at sentiment classification superior to that achieved by the machine learning methods utilized in Task 1. This could be attributed to many factors, most notably the nature of the SST-2 dataset. This dataset is text based, which makes it well suited to the use of prompt engineering. The machine learning methods explored in Task 1 are better suited for dataframes with attributes, especially Logistic Regression, which had the biggest accuracy rate in Task 1 and works better with numerical datasets.

References

Momojit Biswas. 2020. Prompt engineering: Unleashing the power of few-shot learning for accurate spam detection. Medium.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Code

Task_1

```
# -*- coding: utf-8 -*-
"""SWE3011_41_Task1.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1RKBXBhSWJVU9MqRcyZBCIKWBRx6SMf_d

# SWE3011_41 Task1

**Supervised Text Classification using traditional machine learning methods**

1. Complete all the functions given.
2. Conduct various experiments including hyper-parameter tuning, cross validation, etc.
3. Write a report on the analysis of experiment results.

**0. Installation**

**1. Load Dataset**
"""

pip install datasets

"""
Evaluation should be done using **provided test dataset**"""

from datasets import load_dataset

train_ds = load_dataset("glue", "sst2", split="train")

# Evaluation should be done using test_ds
test_ds = load_dataset("csv", data_files="./test_dataset.csv")['train']

"""**2. Preparing Dataset**"""

from sklearn.feature_extraction.text import TfidfVectorizer

def transform_data(X_train, X_test):
    """
    Input:
    - X_train, X_test: Series containing the text data for training and testing respectively.

    Output:
    - X_train_tfidf, X_test_tfidf: Transformed text data in TF-IDF format for training and testing
    ↪  respectively.
    - vectorizer: Fitted TfidfVectorizer object.
    """
    ########################################
    # TODO: Convert the text data to TF-IDF format and return the transformed data and the vectorizer
    # Create a TfidfVectorizer
    vectorizer = TfidfVectorizer()

    # Fit and transform X_train
    X_train_tfidf = vectorizer.fit_transform(X_train)

    # Transform X_test
    X_test_tfidf = vectorizer.transform(X_test)
    ########################################
    return X_train_tfidf, X_test_tfidf, vectorizer

X_train, y_train = train_ds['sentence'], train_ds['label']
X_test, y_test = test_ds['sentence'], test_ds['label']
X_train_tfidf, X_test_tfidf, vectorizer = transform_data(X_train, X_test)

"""**3. Train**"""

from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB

def logistic_regression(X_train_tfidf, y_train):
    """
    Input:
    - X_train_tfidf: Transformed text data in TF-IDF format for training.
    - y_train: Series containing the labels for training.

    Output:
    - clf: Trained Logistic Regression model.
    """
    ########################################
    # Define a logistic regression classifier with max_iter
```

```python
    clf = LogisticRegression(max_iter=1000)

    # Define hyperparameters to tune
    param_grid = {
        'C': [0.001, 0.01, 0.1, 1, 10, 100],
        'penalty': ['l1', 'l2']
    }

    # Perform grid search with cross-validation to find the best hyperparameters
    grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
    grid_search.fit(X_train_tfidf, y_train)

    # Print the best parameters
    print("Best Parameters: ", grid_search.best_params_)

    # Get the best model with the optimal hyperparameters
    clf = grid_search.best_estimator_

    # Train the final model with the entire training data
    clf.fit(X_train_tfidf, y_train)

    ########################################
    return clf

def random_forest(X_train_tfidf, y_train):
    """
    Input:
    - X_train_tfidf: Transformed text data in TF-IDF format for training.
    - y_train: Series containing the labels for training.

    Output:
    - clf: Trained Random Forest classifier.
    """
    ########################################
    # Define a Random Forest classifier
    clf = RandomForestClassifier()

    # Define hyperparameters to tune
    param_grid = {
        'n_estimators': [100, 200, 300],  # Number of trees in the forest
        'max_depth': [None, 10, 20],  # Maximum depth of the tree
    }

    # Perform grid search with cross-validation to find the best hyperparameters
    grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
    grid_search.fit(X_train_tfidf, y_train)

    # Print the best parameters
    print("Best Parameters: ", grid_search.best_params_)

    # Get the best model with the optimal hyperparameters
    clf = grid_search.best_estimator_

    # Train the final model with the entire training data
    clf.fit(X_train_tfidf, y_train)

    ########################################
    return clf

def naive_bayes_classifier(X_train_tfidf, y_train):
    """
    Input:
    - X_train_tfidf: Transformed text data in TF-IDF format for training.
    - y_train: Series containing the labels for training.

    Output:
    - clf: Trained Multinomial Naive Bayes classifier.
    """
    ########################################
    # Define a Multinomial Naive Bayes classifier
    clf = MultinomialNB()

    # Define hyperparameters to tune
    param_grid = {
        'alpha': [0.1, 0.5, 1.0, 2.0]  # Smoothing parameter (Laplace/Lidstone smoothing)
    }

    # Perform grid search with cross-validation to find the best hyperparameters
    grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
    grid_search.fit(X_train_tfidf, y_train)

    # Print the best parameters
    print("Best Parameters: ", grid_search.best_params_)

    # Get the best model with the optimal hyperparameters
    clf = grid_search.best_estimator_

```

```python
    # Train the final model with the entire training data
    clf.fit(X_train_tfidf, y_train)

    #########################################
    return clf

clf = logistic_regression(X_train_tfidf, y_train)

clf_rf = random_forest(X_train_tfidf, y_train)

clf_nb = naive_bayes_classifier(X_train_tfidf, y_train)

"""**4. Evaluation**"""

from sklearn.metrics import accuracy_score, classification_report

def evaluate_model(clf, X_test_tfidf, y_test):
    """
    Input:
    - clf: Trained Logistic Regression model.
    - X_test_tfidf: Transformed text data in TF-IDF format for testing.
    - y_test: Series containing the labels for testing.

    Output:
    - None (This function will print the evaluation results.)
    """
    #########################################
    # TODO: Evaluate the model and print the results
    # Predict the labels using the trained classifier
    y_pred = clf.predict(X_test_tfidf)

    # Calculate the accuracy
    accuracy = accuracy_score(y_test, y_pred)

    #########################################
    print(f"Accuracy: {accuracy:.2f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred))

evaluate_model(clf, X_test_tfidf, y_test)

evaluate_model(clf_rf, X_test_tfidf, y_test)

evaluate_model(clf_nb, X_test_tfidf, y_test)
```

Task_2

(Biswas, 2020)

```python
# -*- coding: utf-8 -*-
"""SWE3011_41_Task2.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/12C7W-1nFm4VgbgysgY1E3tIqFN99MLtf

# SWE3011_41 Task2

**Prompt Engineering with Langchain for LLMS**

1. Based on the given code, you need to extend (modify) to a classification task using Langchain.
2. Since large language model baseline is required for the task, you may choose any model from OpenAI API and HuggingfaceHub.

 **OpenAI**: Only available for paid users.

 **HuggingfaceHUB**: Free to use with usage limits (reset hourly).

3. Conduct experiments and document the results in the report. Here, you should consider what kind of prompt design you use so please find some tutorials/resources in our homework description to obtain more information.

**0. Installation**
"""

pip install datasets

pip install openai

from getpass import getpass
import os

# get a token: https://huggingface.co/docs/api-inference/quicktour#get-your-api-token
HUGGINGFACEHUB_API_TOKEN = getpass()

os.environ["HUGGINGFACEHUB_API_TOKEN"] = HUGGINGFACEHUB_API_TOKEN

pip install huggingface_hub

pip install langchain

"""**1. Load Dataset**

Evaluation should be done using **provided test dataset**
"""

from datasets import load_dataset
import pandas as pd

# You can use train_ds for few-shot examples
train_ds = load_dataset("glue", "sst2", split="train")

# Convert the dataset to a Pandas DataFrame
df_train = train_ds.to_pandas()

# Display the DataFrame
print(df_train.head())

# Evaluation should be done using test_ds
test_ds = load_dataset("csv", data_files="./test_dataset.csv")['train']

# Load the dataset from CSV
dataset_path = "/content/test_dataset.csv"  # Replace with the actual path to your CSV file
df_test = pd.read_csv(dataset_path)

# Display the DataFrame
print(df_test.head())

import random
import numpy as np

def gen_data(df, flag=True, samples=27):
    examples = []

    n = len(df) if flag else samples

    # Count occurrences of each category
    category_counts = df["label"].value_counts()

    for i in range(0, n):
```

```python
        # Choose category proportionally based on occurrences
        category = np.random.choice(category_counts.index.tolist(), p=(category_counts / category_counts.sum()).tolist())

        # Get a random row with the chosen category
        category_df = df[df["label"] == category]
        random_row = category_df.sample()

        obj = {
            "messages": random_row["sentence"].iloc[0],
            "categories": random_row["label"].iloc[0]
        }
        examples.append(obj)

    random.shuffle(examples)

    return examples

"""**2. Preparing Prompt**"""

from langchain import PromptTemplate

prompt_template = '''
Messages: {messages}
Categories: {categories}
'''

example_prompt = PromptTemplate(
    input_variables=["messages", "categories"],
    template=prompt_template)

import re

def trim(s):
    if "positive" in s.lower():
        return 1
    elif "negative" in s.lower():
        return 0
    else:
        return -1  # or any default value for the case where no match is found

from sklearn.metrics import accuracy_score
from tqdm import tqdm
import time

def eval(extraction_chain, example):
    preds = []
    actual = []

    for x in tqdm(example):
        p = extraction_chain.predict(messages=x["messages"])
        p = trim(p)
        a = x["categories"]
        preds.append(p)
        actual.append(a)

    accuracy = accuracy_score(actual, preds)
    return preds, actual, accuracy

from langchain.llms import HuggingFaceHub
from langchain.chains import LLMChain

def initialize_llm(model_name, api_key=None):
    """
    Initialize the model using the langchain library.
    """
    llm = HuggingFaceHub(repo_id=model_name, model_kwargs={"temperature": 0.5, "max_length": 64})

    return llm

"""**Few-Shot**"""

from langchain import FewShotPromptTemplate

examples = gen_data(df_train, False)
print(examples)

# Create the FewShotPromptTemplate
final_prompt = FewShotPromptTemplate(
    examples=examples,
    example_prompt=example_prompt,
    suffix="Messages: {messages} \nCategories: ",
    input_variables=["messages"],
    prefix="""You are a sentiment classifier. Classify the given text input as "positive" or "negative" based on its tone.
        Please read each sentence carefully and determine its sentiment based on common definitions.

```

```
            <positive>: A positive sentiment indicates a favorable or optimistic tone. It expresses approval, satisfaction, or positive emotions.

            <negative>: A negative sentiment indicates an unfavorable or pessimistic tone. It expresses disapproval, dissatisfaction, or negative emotions.

            Please read each sentence and classify it as either "positive" or "negative" based on the provided definitions.
    """
)

# Initialize the language model
model_name = "google/flan-t5-xxl"
llm = initialize_llm(model_name)
val = gen_data(df_test)

# Create the LLMChain
extraction_chain = LLMChain(llm=llm, prompt=final_prompt, output_key="categories")

# Evaluate and print results without a new line after a comma
preds, actual, acc = eval(extraction_chain, val)
print("\nPredicted Categories:", preds)

print("Actual Categories:", actual)

print(f"Accuracy: {acc * 100:.2f}%")

from langchain import FewShotPromptTemplate

# Initialize the language model
model_name = "google/flan-t5-xxl"
llm = initialize_llm(model_name)

# List to store accuracies
accuracies = []

# Perform 10 iterations
for iteration in range(1, 4):
    print(f"\nIteration {iteration}:")

    # Generate new examples for each iteration
    examples = gen_data(df_train, flag = False, samples=27)

    # Define your FewShotPromptTemplate
    final_prompt = FewShotPromptTemplate(
        examples=examples,
        example_prompt=example_prompt,
        suffix="Messages: {messages} \nCategories: ",
        input_variables=["messages"],
        prefix="""You are a sentiment classifier. Classify the given text input as "positive" or "negative" based on its tone.
                Please read each sentence carefully and determine its sentiment based on common definitions.

                <positive>: A positive sentiment indicates a favorable or optimistic tone. It expresses approval, satisfaction, or positive emotions.

                <negative>: A negative sentiment indicates an unfavorable or pessimistic tone. It expresses disapproval, dissatisfaction, or negative emotions.

                Please read each sentence and classify it as either "positive" or "negative" based on the provided definitions.
        """
    )

    # Create the LLMChain with the updated prompt
    extraction_chain = LLMChain(llm=llm, prompt=final_prompt, output_key="categories")

    # Evaluate and print results without a new line after a comma
    preds, actual, acc = eval(extraction_chain, val)
    accuracies.append(acc)  # Store accuracy in the list
    print("Predicted Categories:", preds)

    print("Actual Categories:", actual)

    print(f"Accuracy: {acc * 100:.2f}%")

# Print all accuracies at the end
print("\nAll Accuracies:", accuracies)

"""**Zero-Shot**"""

from langchain import PromptTemplate

# Create the zero-shot prompt template
zero_shot_template = PromptTemplate(
    input_variables=["messages"],
```

```
    template="""You are a sentiment classifier. Classify the given text input as "positive" or "negative"
    ↪ based on its tone.
                Please read each sentence carefully and determine its sentiment based on common definitions.

                <positive>: A positive sentiment indicates a favorable or optimistic tone. It expresses
                ↪ approval, satisfaction, or positive emotions.

                <negative>: A negative sentiment indicates an unfavorable or pessimistic tone. It expresses
                ↪ disapproval, dissatisfaction, or negative emotions.

                Please read each sentence and classify it as either "positive" or "negative" based on the
                ↪ provided definitions.
    """
)

# Initialize the language model
model_name = "google/flan-t5-xxl"
llm = initialize_llm(model_name)
val = gen_data(df_test)

# Create the LLMChain
extraction_chain = LLMChain(llm=llm, prompt=zero_shot_template, output_key="categories")

# Evaluate and print results
preds, actual, acc = eval(extraction_chain, val)
print("\nPredicted Categories:", preds)

print("Actual Categories:", actual)

print(f"Accuracy: {acc * 100:.2f}%", end=" ")
```

A Appendix

```
100%|██████████| 100/100 [00:07<00:00, 14.10it/s]
Predicted Categories: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
Actual Categories: [1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0,
Accuracy: 51.00%
```

Figure 7: Result of prediction using zero-shot prompt

```
100%|██████████| 100/100 [01:07<00:00,  1.49it/s]
Predicted Categories: [1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 1, 0, -1, 1,
Actual Categories: [1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0,
Accuracy: 83.00%
```

Figure 8: Result of prediction with the first execution of few-shot prompt

```
Iteration 1:
100%|██████████| 100/100 [00:28<00:00,  3.56it/s]
Predicted Categories: [0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
Actual Categories: [1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1,
Accuracy: 89.00%

Iteration 2:
100%|██████████| 100/100 [00:28<00:00,  3.49it/s]
Predicted Categories: [0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
Actual Categories: [1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1,
Accuracy: 87.00%

Iteration 3:
100%|██████████| 100/100 [00:31<00:00,  3.13it/s]Predicted Categories: [0, 1, 0, 0, 1, 1, 1, -1, 1, -1, 0, 0, 0, 0, -1, 0, -1, 1, 0, 0, 0,
Actual Categories: [1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1,
Accuracy: 82.00%

All Accuracies: [0.89, 0.87, 0.82]
```

Figure 9: Result of prediction with subsequent executions of few-shot prompt