Final Project

Data Processing
Master in Telecommunication Eng.

October 2024

1 Introduction

In this project, students will apply the knowledge and techniques acquired during the course to solve a learning task related to textual documents. Students will work individually or in pairs on documents that contain recipes from around the world.

The tasks to be carried out will necessarily include:

- Text processing and homogenization.
- Vector representation of documents using the following techniques: TFIDF, Word2Vec, and contextual embeddings based on Transformers.
- Solving a regression task.

The project has a **maximum score of 3 points**, broken down as:

- **Basic project:** 2.25 points
- **Extension:** 0.75 points

The submission deadline is December 13, 2024, at 23:59.

The requirements for each part are outlined below.

2 Dataset Description

The dataset is provided in the JSON file *"full_format_recipes.json"*[1]. This dataset consists of 20 130 entries corresponding to recipes from the `www.epicurious.com` website. For each recipe, the following information is provided:

- *directions*: Instructions for making the recipe.

[1] Recall that you can read a JSON file as a Pandas dataFrame with the function `pd.read_json(path_file)`. More details can be found in the Pandas documentation.

- *categories*: Different categories assigned to the dish.
- *desc*: Description.
- *title*: Title of the post.
- *rating*: Rating given by users.
- *fat*: Amount of fat in grams.
- *protein*: Amount of protein in grams.
- *calories*: Amount of calories in grams.
- *sodium*: Amount of sodium in grams.
- *ingredients*: Quantity of each ingredient.
- *date*: Publication date.

The variables *directions*, *desc*, *categories*, and *title* contain text, while the rest contain numerical values. The *rating* variable will be used as the output to solve a regression problem

3 Basic Project

The basic project will consist of solving a regression task, comparing the performance obtained using different vectorizations of the documents and at least two different machine learning strategies, as described below. The steps to follow in your work are as follows:

1. Analysis of input variables. Visualize the relationship between the output variable and some categories in the *categories* variable and explain their potential relevance to the problem.
2. Implementation of a pipeline for text preprocessing. For this task, you may use common libraries (NLTK, Gensim, or SpaCy), or any other library you deem appropriate. Keep in mind that when working with transformers, the text is passed without preprocessing.
3. Vector representation of the documents using three different procedures:
    - TF-IDF
    - Word2Vec (i.e., representing the documents as the average of the embeddings of the words that make them up)
    - Contextual embeddings calculated from transformer-based models (e.g., BERT, RoBERTa, etc.)
4. Training and **evaluation** of regression models using at least the following two machine learning strategies:

- Neural networks using PyTorch for implementation.
- At least one other technique implemented in the Scikit-learn library (e.g., K-NN, SVM, Random Forest, etc.)

5. Comparison of the results obtained in step 3 with fine-tuning a pre-trained model from *Hugging Face*. In this step, you are asked to use a transformer model with a regression head.

You should use the information in the *directions* and/or *desc* variables for all steps of the project, possibly combining this information with metadata from other variables. You should use appropriate metrics for evaluating this task. The performance of the different methods should be estimated using a validation methodology, which you should also explain in the documentation. You should provide a description of the methodology used and analyze the performance obtained according to the input variables.

Keep in mind that the goal is to describe the work carried out and critically analyze the results obtained. Support this with graphs or other representations you consider appropriate. There is no need to describe the algorithms used, but you should explain how you tuned their parameters.

4 Extension

The extension work is open-ended, and you should expand on the basic project in any direction you find appropriate. For example:

- Use of a pre-trained *summarizer* (using Hugging Face pipelines) to provide a summary of the *directions* variable, which is a list of instructions that may contain relatively large texts and repeated steps.
- Study the ability of transformer models to generate new recipes. Here, you could also compare the performance of this with its implementation using prompting techniques on free-to-use language models (LLAMa, Mixtral, etc.).
- Explore the potential of NLP techniques such as bigram usage, part-of-speech tagging, thesauri, etc., (leveraging, for example, the functionality available in Python's *NLTK* library).
- Performance comparison using different contextual embeddings.
- Visualization and analysis using graph-based techniques.

Take this list as a mere suggestion; you may choose any other topic as long as it fits within the course's scope.

In the extension work, creativity and originality in the choice will be valued. If you have any doubts about the suitability of your chosen extension, consult the instructor. In any case, keep in mind that the extension work constitutes

0.75 points of the final grade. Avoid undertaking overly ambitious extension projects that could compromise the project's timely delivery.

If you have any doubts about the appropriateness of any extension work, consult the instructor.

5 Project Submission

GitHub will be used as the collaboration and version control tool for project submission. The submission must include the following elements:

1. **GitHub Repository**:
   - The GitHub repository should be updated with the code used during the project's development. The repository should be well-structured and contain all the necessary information for executing the project.
   - The repository must include:
     - Properly commented source code.
     - Scripts necessary for data preparation and model training.
     - Experimentation notebooks (if applicable) or analysis scripts.
2. **Documentation**:
   - The repository must contain a `README.md` file that clearly describes the project, the methodology used, implementation decisions, and a description of the results.
   - Alternatively or additionally, students can use **GitHub Pages** to create a webpage where they explain the work done, the results obtained, and any relevant analysis or conclusions in more detail.

   The documentation should include:
   - Description of the problem and project objectives.
   - Methodology applied (data processing, models used, etc.).
   - Results analysis: include graphs, metrics, and a discussion on model performance.
3. **Repository link**:
   - The repository must be public.
   - The submission will be made through Aula Global: one student from each group must upload the link to the shared repository.
   - Documentation pages cannot be edited after the deadline.
   - The last commit before the deadline will be evaluated.

Third-party code snippets or any material from external sources, you must clearly specify this in the report.

6 Evaluation

The project will be evaluated on the basis of the submitted documentation as well as an individual presentation to be held the week of 16 December. The evaluation will be made according to the following criteria:

- Basic Project (2.25 points)
  - Methodology (0.75)
  - Quality of documentation (0.3)
  - Code quality (0.3)
  - Presentation quality (0.5)
  - Responses to comments on the presentation (0.4)
- Extension (0.75 points)
  - Originality (0.25)
  - Quality of the work (0.5)

A Additional Information

This appendix provides information that may be useful and should be clear to resolve the problem.

A.1 Extracting Embeddings with a Transformer

In this section, we explain how to extract embeddings using a transformer model, specifically using BERT as an example. This technique is crucial for obtaining vector representations of words or phrases in the context of the NLP problem to be solved.

Step 1: Load the Pre-trained Model

The first step is to load a pre-trained BERT model from the Hugging Face `transformers` library, which will allow us to obtain the embeddings directly. You can do this as follows:

```
from transformers import BertModel, BertTokenizer
import torch

# Load the pre-trained BERT tokenizer and model
tokenizer =
↪  BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

# Tokenize an example sentence
text = "Example of extracting embeddings with BERT"
input_ids = tokenizer.encode(text, return_tensors='pt')
```

Step 2: Get the Embeddings

With the model and tokenized sentence, we can pass the input to the model to obtain the embeddings. BERT will return two outputs: the hidden layer embeddings and the embeddings of the `[CLS]` token (which can be used for classification tasks).

```
# Get the embeddings
with torch.no_grad():
    outputs = model(input_ids)

# Extract the embeddings from the last hidden layer
last_hidden_states = outputs.last_hidden_state
```

The tensor `last_hidden_state` contains the embeddings of all the tokens in the sentence, and you can use them as needed (e.g., averaging them to get a sentence representation).

More Information

For a more detailed explanation of how BERT embeddings work, as well as other usage details, you can refer to the following tutorial available at this link: https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/#why-bert-embeddings.

A.2 Fine-Tuning a Pre-trained Hugging Face Model

Fine-tuning involves adapting a pre-trained model like BERT for a specific task, such as text classification or sentiment analysis. Below, we explain the basic steps for fine-tuning a pre-trained model using the Hugging Face `transformers` library.

Step 1: Load the Model and Data

First, we need to load a pre-trained model and tokenizer, as well as prepare the training data.

```
from transformers import BertTokenizer,
↪ BertForSequenceClassification
from transformers import Trainer, TrainingArguments

# Load the pre-trained BERT tokenizer and classification
↪ model
tokenizer =
↪ BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert
↪ -base-uncased',
↪ num_labels=2)

# Tokenize the input data
train_encodings = tokenizer(train_texts, truncation=True,
↪ padding=True, max_length=128)
test_encodings = tokenizer(test_texts, truncation=True,
↪ padding=True, max_length=128)
```

In this example, `BertForSequenceClassification` is the BERT model adjusted for a classification task with `num_labels=2` for binary classification.

Step 2: Create the Dataset

After tokenizing the texts, we need to create a dataset compatible with the format required by Hugging Face.

```python
import torch

class Dataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in
        ↪  self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

train_dataset = Dataset(train_encodings, train_labels)
test_dataset = Dataset(test_encodings, test_labels)
```

Step 3: Configure the Training

We use the Hugging Face `Trainer` class to configure and run the fine-tuning process. Here we set up the training arguments, such as the number of epochs and the learning rate.

```python
training_args = TrainingArguments(
    output_dir='./results',          # Output directory
    num_train_epochs=3,              # Number of epochs
    per_device_train_batch_size=16,  # Training batch size
    per_device_eval_batch_size=64,   # Evaluation batch
    ↪  size
    warmup_steps=500,                # Warmup steps
    weight_decay=0.01,               # Weight decay
    logging_dir='./logs',            # Logging directory
    logging_steps=10,
)

trainer = Trainer(
    model=model,                     # Pre-trained model
    args=training_args,              # Training arguments
    train_dataset=train_dataset,     # Training data
    eval_dataset=test_dataset        # Evaluation data
)

# Start fine-tuning
trainer.train()
```

Step 4: Evaluation and Saving the Model

After training the model, we can evaluate it on the test dataset and save the fine-tuned model.

```
# Evaluate the model
trainer.evaluate()

# Save the fine-tuned model
model.save_pretrained('./fine-tuned-bert')
tokenizer.save_pretrained('./fine-tuned-bert')
```

More Information

For more details on fine-tuning and other usage examples, you can consult the official Hugging Face documentation at the following link:

https://huggingface.co/transformers/training.html

You can also refer to the Hugging Face fine-tuning tutorial that will be available on Aula Global.