Tasks to-do:

1. Two feature extractor shallow algorithms (e.g. bag of words, n-grams) - optimise the parameter settings to see what performance you can achieve, to have a baseline for the subsequent steps.
2. Two deep learning architectures (LSTM or RNN most likely)
3. Perform a detailed comparison and analysis per class (confusion matrix), to identify if the two approaches lead to different types of errors in the different classes, and also try to identify other patterns, and a detailed comparison of runtime, considering both time for training and testing, including also the feature extraction components.

Exercise pdf - task

The main goal of this exercise is to get a feeling and understanding on the importance of representation and extraction of information from complex media content, in this case images or text. You will thus get some datasets that have an image classification target.

(1) In the first step, you shall try to find a good classifier with „traditional“ feature extraction methods. Thus, pick for text:

One feature extractor based on e.g. Bag Of Words, or n-grams, or similar

You shall evaluate them on two shallow algorithms, optimising the parameter settings to see what performance you can achieve, to have a baseline for the subsequent steps.

(2) Then, try to use deep learning approaches, such as convolutional neural networks (for images) or recurrent neural networks (for text), or other approaches (but likely not just simple MLPs), and see how your performance differs. Try at least two different architectures, they can be (reusing or be based on) existing, well-known ones.

Compare not just the overall measures, but perform a detailed comparison and analysis per class (confusion matrix), to identify if the two approaches lead to different types of errors in the different classes, and also try to identify other patterns.

Also perform a detailed comparison of runtime, considering both time for training and testing, including also the feature extraction components.

Text classification (Feature Extraction & Shallow vs. Deep Learning)

Machine Learning WS 2022

Exercise 3, Topic 3.2.1

Group 37

Datasets

Amazon Reviews [1]

The dataset contains 17k user reviews scraped from Amazon marketplace along with their rating categorized into three classes: **Positive**, **Neutral** and **Negative**. Therefore, we model the problem of **multi-class classification**. The target variable is slightly unbalanced, as it is visible in the Plot Below:

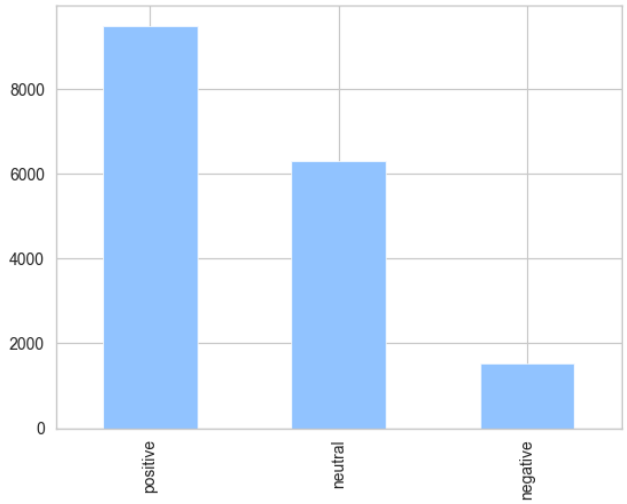

Figure 1: Distribution of Amazon Reviews Sentiments

Due to this class imbalance, the performance of the models will be compared for each class separately. I.e. the accuracy will be macro-averaged.

AG News [2]

The dataset contains the following information about 127k news articles: Title, Details (News Text) and news Category. The news are divided into four categories where 1 represents World,

2 represents Sports, 3 represents Business and 4 represents Sci/Tech. The target classes are distributed evenly in this dataset. The target class distribution is presented in Figure 2.

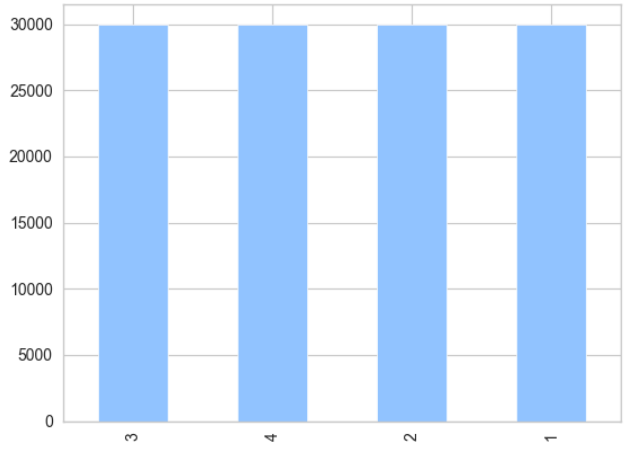

Figure 2: Distribution of the AG News target variable

“Traditional” Feature Extraction

To extract the features from the datasets a composite method consisting of uni-gram and bi-gram vectorizer was used. i.e. both unigrams and bigrams were extracted as tokens. This is because the document lengths are relatively small on average (see Figures 3 and 4). (The idea suggested by [3])

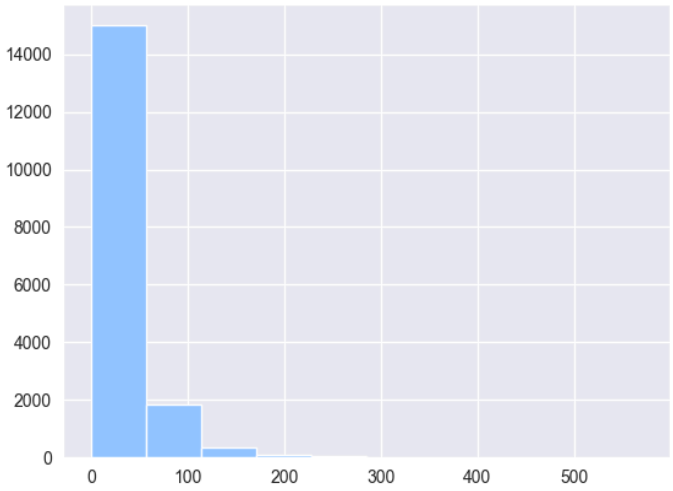

Figure 3: Distribution of review lengths

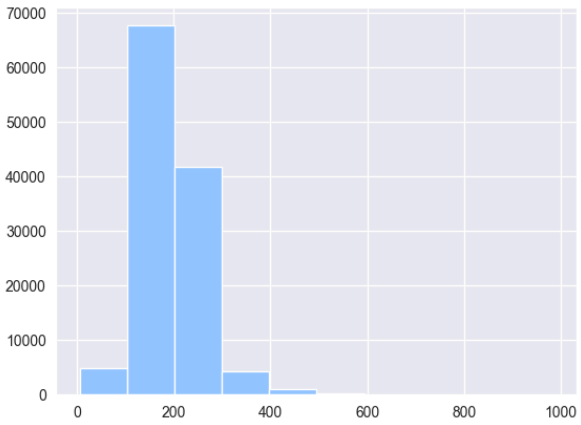

Figure 4: Distribution of news lengths

Shallow Models

Two types of shallow models were trained and evaluated as a baseline. One single hidden layer neural network and an SVM model were selected for each dataset.

Data split

The datasets were split into train-, validation- and test-set with the 60-20-20 rule. I.e. 60% of the samples were dedicated for training, 20% for validating and 20% for testing.

Shallow Neural Networks

As previously stated both NN models had only one hidden layer. The hyperparameters were tuned with the **optuna** framework, with the following Hyperparameter domains:

- Number of neurons in the hidden layer : integer between 10 and 100
- Activation function: One of [tanh, relu, logistic]
- Solver: one of [lbfgs, adam]
- Alpha (learning rate): float between 0.0001 and 0.001

Due to computational constraints wrt size of the datasets the hyperparameter optimization was performed only on the subset of the training set. For the Amazon Reviews dataset, only a half was used for hyperparameter optimization and for the AG news dataset, a sixteenth was used. The neural network models were implemented with SK-Learn's MLPClassifier class. Final models had the following parameters:

- Amazon reviews: hidden_layer_sizes=(85,),
  alpha=0.00045434026867030927,
  solver='adam',
  activation='logistic',
  random_state=42,

learning_rate='adaptive',
max_iter=50,

- AG News: hidden_layer_sizes=(34,),
  alpha=0.0004980605929484772,
  solver='adam',
  activation='tanh',
  random_state=42,
  learning_rate='adaptive',
  verbose=True,
  max_iter=20

The remaining model parameters were set to the MLPClassifier defaults (as of version 1.2.1)

SVMs

The size of datasets used for hyperparameter tuning of SVM models remained the same as for NNs. The hyperparameter grid for the SVM models was:

- C: [0.1, 1, 10],
- gamma: [1, 0.1, 0.01, 0.001, 0.0001],
- kernel: ['poly', 'sigmoid', 'linear', 'rbf']

For the tuning of SVM models, the sk-learn's implementation of RandomizedSearchCV was used with random state of both model and RandomizedSearchCV object set to 42.
After the tuning the optimal hyperparameters for both datasets were **linear** kernel, **gamma=0.001** and **C=10.**
The implementation of an SVM classifier used was **SVC**, offered by the sk-learn framework. Besides the found best hyperparameters, the random state of the models was set to 42 and the remaining hyperparameters were set to their default values as of sklearn version 1.2.1.

On Data Augmentation

The Data augmentation was performed on the first dataset in order to obtain more samples. The technique used was **Synonym replacement**. In particular, for each noun or verb in each sample of the training set, there was a 20% chance of it being replaced with its first synonym.

Since there is already plenty of data available in the second dataset and we have only limited resources regarding computing power, we found it unnecessary to perform data augmentation for this dataset.

Transfer Learning using BERT Model

In order to compare different architectures and approaches to solve text classification challenges, we wanted to see how existing pre-trained models perform on the given datasets compared to the manually built models. The model is downloaded from HuggingFace [4] repository, pretrained on English language, which fits well to our needs. We have chosen the BERT base version, which is a lighter model variation with less parameters, compared to the larger version which is much bigger and requires more training time. The tokenization and model training was done using the **Transformers** library in combination with **Tensorflow** and **Keras** libraries and methods.

The tokenization of input text from datasets was done using corresponding model tokenizer (provided by transformers library which does the data preprocessing). The original model has been extended with the embedded input and output dense layers, to adapt the layers for tokenized inputs and expected output classes.

The hyperparameters were tuned with the following parameters:

- Input activation: sigmoid, tahn, relu
- Learning rate for Adam optimizer: 0.001, 0.0001, 0.00002, 0.00003
- Epsilon for Adam optimizer: 1e-08, 1e-07, 1e-06

The reason why more tuning parameters were not chosen is the long training time and limited free resources with powerful hardware, which per single epoch takes around 30 minutes on the Amazon news training set. The hyperparameter tuning and training the final model was done using 3 epochs. For AG News it is even longer, because the dataset is much bigger.

From the tuning results, it can be seen that *sigmoid* activation function gives much better performance. It can be also observed that the model gives almost the same or worse performance with the second and third run (epoch).

The final models have following parameters:

Amazon Reviews:
- Learning rate: 2e-05
- Activation function: sigmoid
- Adam epsilon: 1e-07

AG News:
- Learning rate: 3e-06
- Activation function: sigmoid
- Adam epsilon: 1e-08

Deep Learning with LSTM

Bidirectional LSTM was used as one of the two deep learning architectures. We created our own embeddings from scratch, by getting all of the unique words in a corpus, and then giving them number values based on the indexes in the corpus. Then we added paddings to the instances of the data set so that every sentence (row) was the same size. Then it could be easily fed to the embedding layer of the LSTM network in keras. We didn't use the pre-trained embeddings, and that would turn out to be a significant disadvantage when dealing with small data sets. More about that in the model comparison, and conclusion.

Furthermore, we focused on experimenting with model architecture, and different parameters like vocabulary size, sentence length, embedding size, etc. We created 5 different models for the Reviews data set, and 3 different models for the News data set. We first tried out manually a couple of different models to see which one produced better results, and how it relates with respect to training time, and then we did hyperparameter optimization. The LSTM model was implemented using the keras library in Python.

The hyperparameters were tuned with the **keras-tuner** optimizer, with the following Hyperparameter domains:

- Vocabulary size: min - 5000, max - 20000, step - 2500
- Embedding size: min - 32, max - 128, step - 32
- Sentence length: set to 100/200 respectively because of preprocessing purposes
- LSTM units: min - 32, max - 128, step - 32
- 1st dense layer units: min - 10, max - 50, step - 10
- 1st dense layer activation functions: relu, sigmoid, tanh
- Learning_rate: 0.01, 0.001, 0.0001

For hyperparameter optimization, the same split was used as in the shallow model tuning.

The final models had the following parameters:

- Amazon reviews:
    - Vocabulary size: 7500
    - Embedding size: 64
    - Sentence length: 90
    - LSTM units: 64
    - Dense units: 50
    - Dense activation function: relu
    - Learning rate: 0.01
- AG News:
    - Vocabulary size: 20000
    - Embedding size: 100
    - Sentence length: 200
    - LSTM units: 100
    - Learning rate: 0.001

The remaining model parameters were set to the keras LSTM defaults.

Evaluation of the results

Shallow NN - News Dataset Large models

Due to a significantly larger vocabulary which is caused not only by the larger number of samples, but also the nature of the dataset, the shallow neural network models fitted with this dataset are very large, as the number of inputs is equal to the vocabulary size (count of all uni-bigrams). The corpus of words used in the news is significantly wider and more diverse when compared to the one used in online reviews, which is the second reason for the vocabulary being larger for the news dataset.

LSTM

The first thing when considering LSTM in an NLP situation is to choose the size of the vocabulary and the sentence length. The bigger the size, the model will be able to capture more information, but it would also take longer to train. Also, regarding the model architecture, we were not able to get significantly better results by stacking more LSTM layers in a network which was interesting. This was due to the problem not being complex enough to require a multi-layer LSTM, or we were not able to find the appropriate architecture for it.

Reviews:

What was interesting with the LSTM model was that for the reviews data set, we were not able to get better results than 84% balanced accuracy, while the MLP was able to get significantly better results. The BERT performance was expected to be great, but the MLP superiority didn't make sense. We were able to improve the model performance a bit by creating the augmented data set, but that still didn't make it better than the MLP. We think the issue is that the embeddings were trained by us, and as the data set was small, the model didn't have time to properly learn the embeddings. Further improvement to our LSTM solution would be to use pre-trained embeddings like Word2Vec, or GloVe, but we didn't do it due to constraints with computational resources and time. The best model was a single-layer LSTM with parameters as mentioned in the above LSTM section. Overall, it was the worst model out of the 3 on the reviews data set.

News:

Here, we were able to get a performance that was on par with MLP, and the BERT with 90% balanced accuracy. We believe this is the case because the data set was much larger, and therefore our network had more time and instances to learn the embeddings. The best parameters were found manually in this case where we used the base single-layer LSTM model

we created, and increased the vocabulary size, sentence length, and embedding size. This improved the model performance a bit and turned out to be our final model. The hyperparameter we did was using the multi-layer LSTM, and it didn't provide us with a better model or results.

BERT

Comparison of shallow and deep models

Lessons learned

Version Management with Large Files

Since the models were too large for normal git pushes, we firstly resorted to git's large file system (LFS), which worked for a few days until we reached the GitHub's data traffic limit of 1GB. Then we removed the large files from the repository and uploaded them to the responsible person's GoogleDrive and shared them via a link. Another, possibly cleaner way of handling this, would be to upload the models to an Amazon S3 bucket and share them that way.

Time-consuming Retraining

There have been multiple occasions where plenty of time was used to train the models, only to be found out that something was missing and the models had to be retrained.
When using cloud environments with the powerful GPUs (e.g. Google Collab or Kaggle Notebook), the notebook needs to stay active (frequent tool checks to extend the session time), so the training process cannot be just left during the night without having a look at it, but requires the attendance, which is not very comfortable if the training process takes more time (1h or more).

Unbalanced data sets regarding the target variable

In the Reviews data set the target variable was heavily imbalanced with negative instances at around 10%, positive instances at around 50%, and neutral at around 30%. This was an issue in the model, where it predicted most of the time the most frequent class, and we could see that from the metrics (the recall of the negative class). This hurt our models' performance, and we

should have used undersampling or oversampling to take care of that issue. Again, one of the things that could be improved on this project, we were not able to accomplish because of time restraints.

Common architecture for runtime comparison

We have established that running all training on one of our local machines would be too time consuming. Therefore we have decided to measure the training and test time on **Google Colab** and **Kaggle Notebook**, since it is a **free** framework where the resources should behave very similarly. It uses its own RAM, GPU, Disk Space and Cloud Compute Engine. We have also looked into Amazon's SageMaker not only for a unified architecture comparison, but also for faster training. However the overhead to set it up was too large, so we did not continue with that option.

References

[1] - Amazon reviews dataset, Accessed on 19.02.2023
https://www.kaggle.com/datasets/danielihenacho/amazon-reviews-dataset
[2] - AG News Dataset, Accessed on 19.02.2023
https://www.kaggle.com/datasets/amananandrai/ag-news-classification-dataset
[3] - SKLearn Feature extraction tutorial Accessed on 19.02.2023
https://scikit-learn.org/stable/modules/feature_extraction.html
[4] - HuggingFace Bert Repository; Accessed on 24.03.2023
https://huggingface.co/bert-base-uncased