Machine Learning I-II: Report

Introduction

The report at hand aims at concisely reporting on the four experiments, conducted with two different methods of text analysis as part of the Machine Learning course for the MA Digital Text Analysis at the University of Antwerp. The objective of the experiments was to train at least one machine learning and one deep learning model to distinguish the speeches by political affiliation. Four algorithms were implemented in total.

Corpus

Origin

The dataset used in all four experiments is a corpus of speeches delivered by Democratic and Republican Presidents of the United States of America. The corpus is available on Kaggle, however the speeches are collected up to September 2021. To include the more recent ones, a dedicated scraping Python script was written that would add the text and useful information (metadata) about it in the original `csv` file.

Preprocessing

Prior to training, the preprocessing step took place. Originally, the dataset included 998 speeches from as early as George Washington's presidency, however the two major parties, Democratic and Republican were formed in 1828 and 1854, respectively. Interestingly, both parties descend from a single party, called Democratic-Republican, which was split during the 1824 election. Speeches that were delivered by Presidents of the latter party were removed from the corpus in order for the stylometric fingerprint not to 'pollute' the classification, bringing thus the total number of available texts to 878.

Then, the texts were converted into lowercase, and the punctuation was removed, along with the most common words that do not carry any significant meaning (*stopwords*), i.e., *the*, *who*, *won't*, *get*, *full*, *again* etc. The rationale behind each of these steps ultimately aims at improving the performance of the models, as (i) words with capitalization will not be treated as different from the lowercased ones; (ii) punctuation is proven to be a more useful feature in authorship profiling or attribution tasks (in case, e.g., the name of the President was the target variable); and (iii) stopwords are Paretto-distributed which means that they roughly account for the 80% of the words in a text even though they almost always are among the 20% most common words in a language – ultimately making a model heavier without resulting to significant performance improvement. Finally, prior to every model being trained, the texts were converted into vectors using Sklearn's Term Frequency-Inverse Document Frequency (`TfidfVectorizer`) and Keras's `Tokenizer` for the machine learning and deep learning approaches respectively.

Machine Learning

For the first part of the assignment, two traditional, machine learning alogirthms were implemented: A Logistic Regression (LR) and a Support Vector Machines (SVM) classifier. Both implementations were conducted through Grid Search for parameter optimization and 10-fold Cross-Validation for training on different splits of the data and ensuring better predictions on any unseen, test set. Both models included a variety vectorization n-gram ranges:

```
'columntransformer__tfidfvectorizer__ngram_range': [(1, 1), (1, 2), (2, 2), (2,
3)]
```

Support Vector Machines

Support Vector Machines (SVM) is one of the go-to algorithms of supervised machine learning, as it aims at finding the optimal hyperplane (decision boundary) that distinctly classifies the data points.

During the experiments, and after the vectorization of the texts, there were two parameters that needed attention. The kernel of the model and the C–used in the grid search for the SVM classifier. The kernel is the main parameter of an SVM model, as it specifies how it will work behind the scenes. Here, `linear` was chosen over the default `rbf`. The C parameter is essentially responsible for optimizing the model on how to avoid misclassifying the training data. As a rule of thumb, the larger the value, the smaller the margin of the hyperplane if all data are classified correctly. Here, multiples of 0.1 by 10 up to 1000 were selected.

```
'svc__C': [0.1, 1, 10, 100, 1000]
```

Logistic Regression

Logisitc Regression (LR) estimates the probability of an event occurring. In the present case, the probability of a speech coming from a Democratic or a Republican president.

In the context of the experiments at hand, a Logistic Regression with balanced `class_weight` was instantiated. This allows for the model to automatically adjust the weights of each of the target variable classes. In case of an imbalanced dataset –here, there are slightly more Democratic speeches– this saves the trouble of manually adjusting the weight of each class by, for example, using other libraries such as Imbalanced-Learn.

Here, the parameters included in the Grid Search are the `solver`, the `penalty`, and the maximum number of iterations before the solvers start to converge (`max_iter`). According to Sklearn's documentation, not all penalties work with all solvers. This results in actually implementing two different parameter grids and Grid Searches, each containing the solvers that are compatible with the respective penalty, `l1` or `l2`.

```
lr_param_grid = {
-    'logisticregression__solver': ['lbfgs', 'newton-cg', 'sag'],
-    'logisticregression__penalty': ['l2'],
+    'logisticregression__solver': ['liblinear', 'saga'],
+    'logisticregression__penalty': ['l2', 'l1'],
    'logisticregression__max_iter': [100, 250, 500, 750, 1000],
}
```

Deep Learning

The second and final part of the assignment deals with the implementation of at least one Deep Learning pipeline on the same dataset. In this context, two Neural Networks, a Convolutional (CNN) and a Long Short-Term Memory (LSTM) were implemented. Pre-trained word embeddings were used in both implementations to decrease the training time and increase the accuracy of the models. Although the models were trained for a maximum of 50 epochs for both cases, early stopping was applied, meaning that if the loss did not get any lower for at least 5 epochs (`patience`), the training would stop. For the optimizer, `Adam` was selected, as it produced better curves compared to Stohastic Gradient Descent (SGD). The learning rate of the optimizer was tested multiple times, ranging from 0.0001 up to 0.1 stepped by 10.

For pre-trained word embeddings, Stanford's Global Vectors (GloVe) were used as the first layer after input (`n_dimensions=300`).

Convolutional Neural Network

Convolutional Neural Networks (CNNs) is primarily used for image analysis and recognition (*computer vision*) in 2D. However, it can also be used for text (1D sequence), as it progresses horizontally. CNNs innovation is that they are immune to translation of data, namely the inputs can still be recognized in other positions, without the need to be reprocessesed.

In this experiment, the architecture of the CNN was tested a lot of times. `Conv1D` layers, each with different number of filters and kernels, were combined with `Dropout` layers prior to the data been fed into the output layer. For the former layers, Rectified Linear Unit (ReLU) loss function was chosen, while the dropout layers were added to hide half of the available data in each step before moving to next one to prevent overfitting during the training process.

```
model.add(keras.layers.Conv1D(128, 5, activation='relu'))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Conv1D(64, 4, activation='relu'))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Conv1D(32, 3, activation='relu'))
model.add(keras.layers.Dropout(0.5))
```

Long Short-Term Memory (LSTM)

The main drawback of other types of NNs is that thay do not 'remember' the previous input(s). On the other hand, Long Short-Term Memory (LSTM) networks, a subtype of Recurrent Neural Networks (RNNs), solves this issue, as the name suggests.

This is the second Deep Learning algorithm and last for this assignment that was implemented. Conversely to convolutions, LSTM layers do not have a kernel parameter. On the contrary, they have two activation functions, one internal (for the cell state) and one for the node output. For the LSTM layer, the parameters at hand are `units`, `activation`, and `recurrent_activation`. The default options were used in both latter cases (`tanh` and `sigmoid` respectively).

Results and Discussion

This series of experiments has without a doubt shed a light on the differences of potential models in the task of binary classification for textual data. Below, a table with the results on the test/validation datasets is shown.

Algorithm	Accuracy
Logistic Regression	.796
Support Vector Machines	.846
Convolutional Neural Network	.443
Long Short-Term Memory	.710

The low accuracy, especially in the case of the CNN, could be attributed, as previously mentioned, to that CNNs perform better on images. The accuracy of the LSTM, albeit higher than the CNN's, seems to suggest that neural networks require a lot more data points to train on. It goes without saying that their architecture is more complicated that traditional machine learning approaches as well. Luckily, recent advancements in NLP with the advent of Transformers, the process of working on pre-trained models becomes easier and more promising for higher results.

Last but not least, it seems that Democratic presidents even from the early years of the party been formed tend to write similarly or, rather, distinctively enough for the models to separate them from the Republican counterparts.