ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Corso di Laurea Magistrale in Informatica

Caputo Giuseppe
Matricola: 1056533

Application for italian POS tagging using DNN

A.A. 2022-2023

Contents

Chapter 1

Introduction

Natural Language Processing (NLP) is a field of study that focuses on enabling computers to understand and process human language. Within NLP, **part-of-speech (POS) tagging** plays a fundamental role in analyzing and interpreting textual data. POS tagging involves assigning grammatical tags to each word in a sentence, indicating its syntactic category and role within the sentence structure. This task holds significant importance in numerous NLP applications, including machine translation, information extraction, sentiment analysis, and text generation.

POS tagging serves as a crucial initial step in the linguistic analysis of text. By assigning specific tags to words, it provides valuable insights into their grammatical properties, such as whether a word is a noun, verb, adjective, adverb, pronoun, or other grammatical categories. These tags help disambiguate word meanings and contribute to understanding the overall structure and meaning of a sentence.

1.1 Project Objective

This work aims to develop an application for part-of-speech tagging in the Italian language using a DNN (Deep Neural Network) approach, specifically utilizing the tagged dataset from EVALITA 2007. Part-of-speech tagging plays a crucial role in linguistic analysis and natural language processing tasks by assigning grammatical tags to each word in a given text. Accurate and efficient tagging of Italian words enables users to analyze and process Italian text with enhanced linguistic understanding.

The EVALITA 2007 tagging dataset was used as the basis for the training, model evaluation and evaluation part. It comprises a large collection of Italian text samples, manually annotated with the corresponding part-of-speech tags. The composition of the dataset includes a variety of textual genres, domains and linguistic complexities that make it suitable to capture the nuances and diversity of the Italian language. Prior to training the model, the dataset underwent pre-processing steps to ensure data integrity and consistency.

In selecting the DNN approach for part-of-speech tagging, an RNN (Recurrent Neural Network) architecture was specifically chosen due to its superior performance in sequence labeling tasks. RNNs have the inherent ability to model sequential dependencies and capture contextual information by considering the preceding words in a sentence. This makes them particularly well-suited for tasks like part-of-speech tagging, where the grammatical role of a word is often influenced by its neighboring words. By leveraging the power of RNNs, the developed application achieves enhanced accuracy and linguistic understanding in part-of-speech tagging for the Italian language.

1.2 EVALITA 2007

EVALITA is a periodic evaluation campaign of Natural Language Processing (NLP) and speech tools for the Italian language. The general objective of EVALITA is to promote the development of language and speech technologies for the Italian language, providing a shared framework where different systems and approaches can be evaluated in a consistent manner.

The **EVALITA 2007** dataset consists of a diverse collection of Italian texts from various domains, including news articles, opinion pieces, and literary works. These texts have been meticulously annotated with POS tags, ensuring the availability of high-quality labeled data for training and evaluation purposes. The dataset's annotations encompass a wide range of grammatical categories, allowing for the development and evaluation of robust POS tagging systems that can handle the intricacies of Italian language.

Using this dataset, some tasks were carried out in 2007:

- Part of Speec Tagging
- Parsing

- Word Sense Disambiguation
- Temporal Expression Recognition and Normalization
- Named Entity Recognition

Chapter 2

Data Preprocessing

In this chapter, will explore the process of data preprocessing for Italian POS tagging. Data preprocessing plays a crucial role in preparing the dataset for model training and evaluation. It involves various steps such as data cleaning, tokenization, word and tag encoding, dataset splitting, and data transformation.

2.1 Dataset description

Il dataset consiste in una raccolta di frasi italiane annotate con i corrispondenti tag Part-of-Speech (POS). Il dataset proviene da EVALITA 2007. Comprende diverse frasi e copre un'ampia gamma di argomenti e modelli linguistici.

The dataset consists of 133,756 tagged tokens to which several pre-processing steps were performed and subsequently used for training the model; in addition, there is also a test set consisting of 17313 tokens for the evaluation of the model.

There are two types of datasets with two different names *'DevSet_DISTRIB.txt'* - *'TestSet_DISTRIB.txt'* and *'DevSet_EAGLES.txt'* - *'TestSet_EAGLES.txt'*, which have differents tagsets. The one used for this work is the EAGLES one.

2.2 Reading and Splitting data

The initial step involves reading the data and breaking them down into sentences, where each sentence contains word-tag pairs. This process ensures that the data is structured correctly for subsequent analysis. The subdivision into sentences is carried out taking '.' into account, so that a complete sentence can be reconstructed.

Next, the data is divided into input sequences (X) and output sequences (Y). The input sequences represent the words of the sentences, while the output sequences correspond to the corresponding POS tags. The X and Y sequences are constructed by iterating the word-tag pairs and extracting the word and tag information.

Two key operations were applied to the words:

- **html.unescape**, ensuring that special characters or entities are correctly represented
- **lower**, the word is converted to lowercase; converting all words the model becomes more robust and better able to generalize across different capitalization variations

To gain insights into the data, some basic analysis is performed. The total number of tagged sentences is calculated (5793), providing an overview of the dataset's size. Additionally, the vocabulary size (19792) and the total number of unique tags (32) are computed, giving an understanding of the diversity of words and tags present in the dataset.

```
word_tag_sequences = []
for file_path in file_paths:
  with open(file_path, 'r') as file:
      sentence = []
      for line in file:
          line = line.strip().split()
          if line:  # Skip empty lines
              sentence.append(tuple(line))
              if line[0] == '.':
                  word_tag_sequences.append(sentence)
                  sentence = []

X = [] # store input sequence
Y = [] # store output sequence

for sentence in word_tag_sequences:
    X_sentence = []
    Y_sentence = []
    for entity in sentence:
        X_sentence.append(html.unescape(entity[0].lower()))  # entity
            [0] contains the word
        Y_sentence.append(entity[1])  # entity[1] contains
            corresponding tag

    X.append(X_sentence)
    Y.append(Y_sentence)
```

2.3 Data Encoding

As part of the data preprocessing phase, encoding the input and output sequences is crucial for preparing the data for model training.

The input sequences (X) and output sequences (Y) are encoded using tokenization. The Tokenizer class is used to instantiate tokenizers for both the words in X and the corresponding POS tags in Y. The tokenizers are then fit on the respective sequences using the **fit_on_texts()** method, which builds the vocabulary based on the unique words and tags in the data.

Next, the **texts_to_sequences()** method of the tokenizers is applied to convert the X and Y sequences from text to sequences of integers. This encoding step assigns a unique numerical value to each word and POS tag in the vocabulary, allowing the model to process the data effectively.

After this, it was necessary to make sure that the conversion was done correctly and that the lengths of the input and output sequences did not differ.

2.4 Padding

To ensure that all sequences have the same length, padding is applied to a predetermined length, in this case 100, to truncate longer sequences or fill shorter sequences with zeros. Padding can occur at the beginning or end of sequences, depending on the "padding" and "truncating" arguments provided.

By applying sequence padding, all input and output sequences are standardized to a fixed length, regardless of their initial length. This step is critical to facilitate batch processing during model training, ensure compatibility between different sequences, and maintain consistent input sizes.

2.5 Word2Vec and One-Hot Encoding

Word embeddings play a crucial role in capturing the semantic meaning of words and enhancing the performance of natural language processing models. In this section, is discussed the training of a Word2Vec model, the creation of word embeddings, and the subsequent one-hot encoding of the output sequences.

```
embedding_dim = 100
word2vec_model = Word2Vec(X, vector_size=embedding_dim, min_count=1,
    window = 5, workers =1, negative=25,sample = 1e-4)

VOCABULARY_SIZE = len(word_tokenizer.word_index) + 1

embedding_weights = np.zeros((VOCABULARY_SIZE, embedding_dim))

word2id = word_tokenizer.word_index

for word, index in word2id.items():
    try:
        embedding_weights[index, :] = word2vec_model.wv[word]
    except KeyError:
        pass
```

To begin, the **Word2Vec model** is trained using the input sequences (X) to obtain word embeddings. The model is instantiated with the desired dimensionality of the embeddings and various parameters such as *min_count*, *window*, *workers*, *negative*, and *sample*. Training the Word2Vec model allows it to learn the underlying relationships and contextual information among the words in the dataset.

The next step involves creating an empty **embedding matrix** with dimensions corresponding to the **vocabulary size** and the **embedding dimension**. The vocabulary size is determined by the number of unique words in the input sequences.

A word-to-index dictionary (word2id) is created using the word tokenizer, which maps each word to its corresponding index in the vocabulary. This dictionary will be used to assign the word embeddings to their respective positions in the embedding matrix.

To populate the embedding matrix, the vectors from the trained Word2Vec model are copied for the words present in the corpus. Each word's index and corresponding embedding vector are retrieved from the word2id dictionary and the Word2Vec model, respectively. In case a word is not found in the Word2Vec model (indicated by a KeyError), the embedding weights for that word remain as zeros.

The resulting embedding matrix represents each word in the corpus using a vector of embedding_dim dimensions. These embeddings capture the semantic relationships and contextual information of the words, forming a foundation for the subsequent model training.

To prepare the output sequences (Y) for training, one-hot encoding is performed using **Keras** *to_categorical()* function. This encoding transforms the Y sequences into one-hot encoded vectors, where each tag is represented as a binary vector of length equal to the number of unique tags. This encoding scheme allows the model to learn and predict the POS tags accurately.

2.6 Data Splitting

First, it's worth noting that in this particular scenario, the DevSet file and the test file were concatenated early in the process to form a single dataset.

Once the total set is prepared, it is divided into training, validation, and testing sets using the **train_test_split()** function. the test set size is calculated taking the 15% of the entire dataset. Subsequently, the training data is further divided into training and validation sets using the 15% of the set. This division allows for the creation of a separate validation set, which serves as an independent subset used for monitoring the model's performance and tuning hyperparameters during the training process.

Chapter 3

The Model

In this chapter, the focus will be on the part concerning the model developed and the reasons for choosing it.

3.1 Model Architecture

The model is constructed using the Keras Sequential API. The architecture consists of the following layers:

1. **Embedding Layer**: This layer serves as the first layer in the model and is responsible for representing each word in the input sequence using a vector of a specified length. The embedding layer takes in the following parameters:

    - **input_dim**: The vocabulary size, which corresponds to the number of unique words in the data
    - **output_dim**: The length of the vector with which each word is represented, also known as the embedding dimension
    - **input_length**: The length of the input sequence, which is set to **MAX_SEQ_LENGTH**.
    - **weights**: The word embedding matrix derived from pre-trained embeddings using Word2Vec
    - **trainable**: Specifies whether the embedding weights should be updated during training, setted to **True**.

```
DNN_model = Sequential()
DNN_model.add(Embedding(input_dim      =  VOCABULARY_SIZE,
                        output_dim    =  embedding_dim,
                        input_length  =  MAX_SEQ_LENGTH,
                        weights       = [embedding_weights],
                        trainable     = True
))
DNN_model.add(Bidirectional(LSTM(lstm_units,
            return_sequences=True
)))

DNN_model.add(Dropout(dropout_rate))
```

2. **Bidirectional LSTM Layer**: The bidirectional LSTM layer processes the embedded input sequence in both the forward and backward directions. By capturing information from both directions, the model gains a better understanding of the contextual relationships between words.

3. **Dropout Layer**: To prevent overfitting and improve generalization, a dropout layer is introduced. Dropout randomly sets a fraction of input units to zero during training.

4. **TimeDistributed Dense Layer**: This layer is responsible for generating output predictions for each element in the sequence. It applies a dense (fully connected) layer with a softmax activation function to produce a probability distribution over the POS tag classes. To this layer have been passed the **number of classes**, that represents the total number of unique tags in dataset.

3.2 Hyperparameters

Hyperparameters play a crucial role in the performance and behavior of the part-of-speech tagging application. They are adjustable parameters that are set prior to the training process and impact the model's architecture, learning process, and generalization capabilities. In this application, there are various hyperparameters to consider, including those related to Word2Vec, dropout, and the number of LSTM units in the layer. As there are many in this work, a study was not made using combinations of hyperparameters, but the most common ones found in the literature were used.

3.2.1 Word2Vec Hyperparameters

Word2Vec is utilized to generate word embeddings, representing each word in a continuous vector space. The hyperparameters associated with Word2Vec influence the quality and characteristics of these embeddings. Some of the important Word2Vec hyperparameters include:

- **Vector Size**: This hyperparameter, set to 100 in this application, determines the dimensionality of the word embeddings. A higher vector size allows for a more expressive representation of words, capturing more fine-grained semantic and syntactic information.
- **Min Count**: This hyperparameter, set to 1, specifies the minimum frequency threshold for words to be included in the Word2Vec model. Words occurring less frequently than the min count are excluded from the vocabulary.
- **window**: The window size, set to 5, determines the maximum distance between the current word and the context words used to predict it, that focus on more local context, leading to more specific and contextually relevant embeddings.

3.2.2 Model Hyperparameters

As mentioned above, there are several hyperparameters, with which different combinations have not been tested:

- **dropout_rate**: used in the Dropout layer has been set to 0.5
- **epochs**: the study was carried out on 20 epochs
- **lstm_units**: that determines the complexity and expressive power of the model's recurrent layer, three different values have been considered: 128, 256, and 512
- **batch_size**: is calculated as one-fourth (1/4) of the number of LSTM units in the layer. This calculation allows for a balance between computational efficiency and model optimization, ensuring that the model receives sufficient training examples

3.2.3 Compile Hyperparameters

The compile hyperparameters influence the training process and the optimization algorithm used in the model. The specific compile hyperparameters considered in this application include:

- **loss_function**: set to *categorical_crossentropy*, that is a standard choice for multi-class classification tasks like part-of-speech tagging
- **Optimizer**: set to *adam*, determines the algorithm used to update the model's weights based on the calculated gradients
- **Metrics**: set to *['acc']*, specify the evaluation metrics used to monitor the model's performance during training

Chapter 4

The Evalution

The developed model was evaluated using three different configurations of LSTM units: 128, 256, and 512. The evaluation focused on measuring the accuracy of the model in part-of-speech tagging for the Italian language. Additionally, tagging accuracy was computed to assess the model's ability to assign the correct part-of-speech tags to individual words.

The accuracy values represent the overall performance of the model in correctly predicting the part-of-speech tags for the words in the test set. These values indicate the proportion of correctly predicted tags out of the total number of words in the test set.

The tagging accuracy values reflect the accuracy at the word level, considering whether each individual word was assigned the correct part-of-speech tag. These values provide a more granular assessment of the model's performance, measuring the ability to accurately label each word in the text.

Test set accuracy and tagged accuracy are found to be identical, indicating that the model correctly predicts part-of-speech tags for words in the test set with a word-level accuracy that is consistent with its overall accuracy.

The evaluation results for the different LSTM unit configurations are as follows:

- **LSTM Units: 128**: $98,4\%$, with 85525 correct tokens over 86900
- **LSTM Units: 256**: $98,39\%$, with 85502 correct tokens over 86900
- **LSTM Units: 512**: $98,37\%$, with 84485 correct tokens over 86900

Chapter 5

Conclusion

In this study, has been developed a part-of-speech tagger for the Italian language using a Deep Neural Network (DNN) approach. The tagger was trained and evaluated on the EVALITA 2007 dataset, which provided a comprehensive collection of tagged Italian text.

The results of the evaluation indicate that the developed part-of-speech tagger performs effectively and accurately on the EVALITA 2007 dataset. The model demonstrated a high level of accuracy in correctly predicting part-of-speech tags for Italian words. The test set accuracy and tagging accuracy showcased the model's capability to capture and understand the linguistic characteristics of Italian text.

However, it is important to note that the effectiveness of the part-of-speech tagger is constrained to the EVALITA 2007 dataset. The performance observed in this study does not guarantee the same level of accuracy when applied to other datasets or different contexts. Further evaluation and testing on diverse datasets are necessary to ascertain the generalizability and robustness of the part-of-speech tagger.

References

[1] *Fabio Tamburini* - "(Better than) State-of-the-Art PoS-tagging for Italian Texts" 2007

[2] *Fabio Tamburini* - "CORISTagger: a highperformance PoS tagger for Italian." 2007b

[3] *Fabio Tamburini* - "How "BERTology" Changed the State-of-the-Art also for Italian NLP"

[4] *Fabio Tamburini* - Natural language processing course material 2022

EVA EVALITA 2007 - *https://www.evalita.it/campaigns/evalita-2007/*

[5] KERAS - *https://keras.io/guides/* 2023

[6] Tensorflow - *https://www.tensorflow.org/?hl=it* 2023