Project NLP: Resume Extraction - Week 9 Deliverables

NLP Interns at Data Glacier

BatchCode: LISUM20

Group Name: TeamNLP

Name	Email	Country	College/Company	Specialization
Nilsu Bozan	bozannilsu@gmail.com	Turkey	Binghamton University/Istanbul Technical University	NLP
Nishchay Vaid	Nishchay89@gmail.com	USA	Rutgers University	NLP
Anish Mitra	anishmitra9666@gmail.com	USA	Montana State University	NLP
Sukriti Macker	sm11017@nyu.edu	USA	New York University	NLP

Problem Description

Resumes contain surfeit information that is not relevant for the HR/authority, and they have to manually process the resumes to shortlist the promising candidates for them. And, thus making the shortlisting task a herculean task for HR. By making use of the NER(Named Entity Recognition) model of NLP this problem can be solved by finding and classifying the entities that are present in each resume into predefined classes such as person name, college name, academics information, relevant experiences, skill set, etc.

GitHub Link Repo

https://github.com/nishchayvaid/Resume-Extraction/blob/main/Resources/Week9/Week9_Deliverables.pdf

Data Cleansing & Transformation

- Nischay

Importing libraries:

- import pandas as pd: Imports the pandas library for data manipulation and analysis.
- import re: Imports the re module for regular expression operations.
- import nltk: Imports the Natural Language Toolkit (NLTK) library for natural language processing.
- import spacy: Imports the Spacy library for advanced natural language processing.

- from num2words import num2words: Imports the num2words module for converting numbers to words.

Data preprocessing:

- Lowercasing the text using resume.lower().
- Removing newlines using re.sub("\n", ' ', resume).
- Removing special characters using re.sub(r'[,•()➢❑]', ' ', resume).
- Removing extra whitespaces, dashes, and dots using re.sub(r'\s\s+|\s-\s|.\s', ' ', resume).
- Tokenizing the text into words using resume.split(" ").
- Converting digits to words using num2words(tokenized_words[i]).
- Removing stopwords using NLTK's stopwords and a custom implementation.
- Joining the tokens back into a string. Reading and saving data:
- Reading a JSON file into a DataFrame using pd.read_json('Resume.json', lines=True).
- Saving the DataFrame to a CSV file using df.to_csv('dataframe.csv', index=None). Vectorization and feature extraction:
- Using the Tf-Idf vectorizer from scikit-learn (sklearn.feature_extraction.text.TfidfVectorizer) to calculate the Tf-Idf values of words and collocations.
- Creating an instance of the Tf-Idf vectorizer with a specified ngram range using TfidfVectorizer(ngram_range=(1, 3)).
- Applying the Tf-Idf vectorizer to the text data using vect.fit_transform(content_resumes).
- Retrieving the terms (words and collocations) in the same order as they appear in the Tf-Idf matrix using vect.get_feature_names_out().

Part-of-speech tagging:

- Using NLTK's pos_tag function to assign part-of-speech tags to the terms obtained from the Tf-Idf matrix using nltk.pos_tag(terms).
- Using Spacy's language model to process the text and assign part-of-speech tags to each word using nlp(text) and word.pos_.

By Anish

Importing libraries(block 1):

- Imported the necessary libraries(json, pandas, numpy and pickle).

Opening JSON file(block 2):

- Used the pickle module to convert the JSON file to a list as shown in one of the modules we completed individually.

Taking a high-level look at the raw data(block 3):

- Printed out each resume to see what the raw resumes looked like.

Creating dataframe(block 4):

- I created a dataframe with two separate columns(the annotations and lists).
- I printed out the dataframe

Converting to list(block 5):

- I firstly converted the annotations dataframe to a list and printed it out.
- I then counted the number of values in the dataframe.
- Finally, I created a new list with just the information.

Converting new list to dataframe(block 6):

- I then converted the new list I created into a dataframe.

Creating lists for the row number, information and field asked(block 7):

- I used dictionaries to separate the information about the candidate to the field where the question was asked.
- I also printed out row numbers.

Counted the number of elements in each list and printed out the elements as well as their type(block 8):

- I counted the number of elements in the row number, candidate information and the field it is in.
- I then created a window function to group each candidate by the row number.

Created final csv(block 9):

- I created a dataframe based on the lists of row number, candidate number, candidate information and desired field.
- I converted this list to csv.
- I edited a few inconsistencies in excel like missing values and duplicate values.
- I printed out the csv.

By Nilsu

Importing libraries:

- The code imports the necessary libraries for data manipulation and analysis (numpy, pandas), regular expressions (re), string operations (string), stopwords and tokenization (nltk.corpus, nltk.tokenize), lemmatization (nltk.corpus, nltk.stem), and part-of-speech tagging (nltk).

Data preprocessing:

- The code preprocesses the text data by performing the following steps:
    - Lowercasing the text.
    - LRemoving newlines.
    - LRemoving special characters.
    - LRemoving extra whitespaces, dashes, and dots.
    - LTokenizing the text into words.
    - Converting digits to words.
    - LRemoving stopwords.
    - LJoining the tokens back into a string.
- Reading and saving data:

    - The code reads a JSON file into a pandas DataFrame.
    - The DataFrame is then saved to a CSV file.

- Lemmatization and part-of-speech tagging:

    - The code applies lemmatization to the 'content' column in the DataFrame by tokenizing the text, lemmatizing each token based on its part-of-speech tag, and joining the lemmatized tokens back into a single string.
    - It also performs part-of-speech tagging on the lemmatized 'content' column and creates a new column with the POS tags.

- Spacy part-of-speech tagging:

    - The code uses the Spacy library to process the text and assign part-of-speech tags to each word in the 'content' column.
    - It defines a function that prints the token text and its corresponding part-of-speech tag using the Spacy language model.

By Sukriti

- **Importing libraries:** The code is importing a number of Python libraries, including:

    - numpy: A library for scientific computing
    - pandas: A library for data analysis
    - re: A library for regular expressions
    - string: A library for working with strings
    - contractions: A library for expanding contractions
    - spacy: A library for natural language processing
    - nltk: A library for natural language processing
    - stopwords: A library for stopwords
    - warnings library is also imported, and the warnings.filterwarnings() function is used to ignore warnings. This is often done to prevent warnings from cluttering up the output of a program.

The two columns *content* and *annotation* provided in the json have been handled separately.

Content

1. The clean_text() function converts text to lowercase, expands contractions, removes newlines, urls, special chars & digits (except @ and .), and removes extra whitespace. All the values are stored in a column called 'cleaned_content'.

2. The tokenization() function tokenizes text using spaCy, a natural language processing library. It loads the English language model, processes the text, and extracts the tokens from the processed document. All the values are stored in a column called 'tokenized_content'.

3. The code first creates a set of stopwords, which are words that are often ignored when processing text. It then creates a list of all the words in the tokenized_content column of the df dataframe. Finally, it iterates over the list of words and adds any words that are not stopwords and not punctuation to a new list called totalWords.

4. The code uses the nltk.FreqDist() function to create a frequency distribution of the words in the totalWords list. This function counts the number of times each word appears in the list and returns a dictionary that maps each word to its frequency.

5. The code uses the wordfreqdist.most_common() function to get the 20 most common words in the frequency distribution. This function returns a list of tuples, where each tuple contains a word and its frequency.

Annotation

1. The extract_text_from_label is a function that iterates over the data list and extracts the text from the labels. It then creates a dictionary that maps each label to a list of texts. The dictionary is returned. All the values are stored in a column called 'annotation_info_extraction'.

2. The clean_annotation is a function that iterates over the dictionary dictionary and cleans the annotations. It then returns a new dictionary with the cleaned annotations. The dictionary is returned. All the values are stored in a column called 'annotation_info_extraction'.

Then, the data is saved in a csv format with the file name "cleaned_df".