Dolphin Documentation

Release 0.0.1

Infodevelopers Pvt Ltd

Jul 11, 2020

Table of Contents

dolphin

Dolphin is the **Dolphwin**'s Artificial Intelligence component – as intelligent as a Dolphin.

Directory Structure

```
└── dolphwin                # Main Directory
    ├── dolphin                 # AI package for Dolphwin
        ├── archives
        ├── data            # data for pool parsing
            ├── languages.csv          # predefined pool for languages
            ├── nationality_data.csv    # predefined pool for nationalities
            ├── soft_skills_resume.csv  # predefined pool for soft skills
            ├── technical_skills.csv    # predefined pool for technical skills
        ├── datasets            # data of resume, job description and other files
        ├── docs                # ? project document files
        ├── labs                # ? contains research program files
        ├── models              # machine learning models saved
            ├── NER_model.ser.gz   # NER model saved from training with Stanford NLP
            ├── stanford_ner.jar   # pretrained stanford model
            ├── segment_identifier.pkl  # model trained for resume segmentation
            ├── new_model_svc.pkl   # SVC model for document classification
            ├── tfidf1.pkl         # tfidf model used for document classification
        ├── segmentation           # module for resume segmentation
            ├── segment_resume.py   # python module for resume segmentation
        ├── stanfordNER            # module for parsing using NER
            ├── formatdata.py       # module for formatting data
            ├── namedentityrecognition.py   # module for getting named entities
            ├── standardizedata.py         # module for standardizing data
        ├── datareader.py            # python module for reading textual content frommultiple
    extension files
```

```
        ├── pipeline.py              # python module for mantaining every task in custom spacy pipeline
        ├── evaluate_pipeline.py     # python module for testing working of pipeline with a test resume
        ├── settings.py              # python module for mantaining configurations
        ├── settings.json            # configuration in json format
    ├── tests                        # all unit testing files
        ├── test_datareader.py       # unit testing file for datareader module
        ├── test_identify_resume.py  # unit testing file for document categorization module
        ├── test_ner_parsing.py      # unit testing file for testing ner parsing module
        ├── test_segment_resume.py   # unit testing file for testing resume segmentation module
    ├── setup.py                     # setup file for pip
    ├── LICENSE                      # License details
    ├── info.log                     # Logs mantained using logging module
    ├── requirements.txt             # list of python modules required to run the system
    ├── README.md                    # markdown documentation of the system
```

—

Pipelines

The pipelines used in this system are shown in the below diagram. ![pipeline](pipeline.png)

Document Categorization pipeline

This is a custom spacy pipeline which is used for identifying the category of any textual documents. The input documents supports following file formats:

- .pdf
- .docx
- .doc
- .txt

The cateogory defined in the pipeline are as follows:

1. Resume
2. Job Decription
3. Others

This pipeline is developed training a Support Vector Classifier with training data containing 400 samples each of resume, job descriptions and other files.

> Trained model for document categorization ——> new_model_svc.pkl

Usage

``` from pipeline import nlp

doc = nlp(textual_content_of_file)
```

category = doc._.category

```

## Segmentation pipeline

This pipeline is used to segment any resume to following segments:

- Profile segment
- Objective segment
- SKills segment
- Academics segment
- Experience segment
- Language segment
- Projects segment
- Rewards segment
- References segment
- Links segment

If any document is categorized as resume, it is passed to the segmentation pipeline to break down into different resume segments.

> Trained model for resume segmentation –> segment_identifier.pkl

### Usage

``` from pipeline import nlp

doc = nlp(textual_content_of_file)

profile_segment = doc._.profile_segment objective_segment = doc._.objective_segment skills_segment = doc._.skills_segment academics_segment = doc._.academics_segment experience_segment = doc._.experience_segment language_segment = doc._.language_segment projects_segment = doc._.projects_segment rewards_segment = doc._.rewards_segment references_segment = doc._.references_segmen links_segment = doc._.links_segment ```

## NER Parser pipeline

This pipeline is developed for the extraction of vital informations from any resume such as experience, education, personal information with name and address from any resume. For extractoin of such information from any resume, custom **Named Entity Recognition(NER)** model has been trained. This model has been trained using the **StanfordNLP** NER training interface. This model has been trained with the tokens of about 285 resumes.

The information extraction from resume using this pipeline are as follows:

- Name
- Address
- Education
- Experience

> Trained model for Named Entity Recognition —-> NER_model.ser.gz

### Usage

``` from pipeline import nlp
```

doc = nlp(textual_content_of_the_file) name = doc._.name address = doc._.address experience = doc._.experience education = doc._.education ```

Pattern Matching Pipeline

The information which follows a specific pattern or is contained with finite pool have been extracted using Spacy Matcher. The information extracted using this pipelines are as follows:

- technical skills
- soft skills
- nationality
- languages
- date
- phone number
- gender
- email

Usage

``` from pipeline import nlp doc = nlp(textual_content_of_file)

nationality = doc._.nationality language = doc._.language email = doc._.email phone_number = doc._.phone technical_skills = doc._.technical_skills soft_skills = doc._.soft_skills date = doc._.date gender = doc._.gender ```

# Modules Used

- PyMuPDF
- texteract
- datefinder
- docx2xt
- gensim
- nltk
- spacy
- pytesseract
- sklearn

> Other requirement —> Downloading en_core_web_sm modle from spacy —-> python -m spacy download en_core_web_sm

`dolphin.datareader.`**`clean_text`** ( *text*, *dolower* )

**Accepts the plain text and makes use of regex for cleaning the noise** :param: text :type:str :return:-cleaned text :type str

`dolphin.datareader.`**`doc_to_text`** ( *filepath*, *dolower* )

**Takes the doc file from the file path param and returns the cleaned the text from the file.** :param filepath: path/directory of the doc file in the system :return: Returns the cleaned text from the file

`dolphin.datareader.`**`docx_to_text`** ( *file_path*, *dolower* )

**Takes docx files and extracts plain text from the docx files** :param file_path :type str :return:text :type str
```

dolphin.datareader.**img_to_text** (*filepath*, *dolower*)

Takes the image file from the file path param and returns the cleaned the text from the image file. :param filepath: path/directory of the image file in the system :return: Returns the cleaned text from the image file

dolphin.datareader.**pdf_to_text** (*file_path*, *dolower*)

Takes filepath and extracts the plain text from pdf for training the word to vec model :param file_path :type str :return:text :type str

dolphin.datareader.**prepare_text** (*file*, *dolower*)

Takes the resume or any other doc; checks the extension of doc and then uses suitable methods to extract and clean the text :param: file :type str :return: cleaned tokenized sentences :type list

dolphin.datareader.**prepare_text_from_string** (*text*, *dolower*)

takes string of text and then cleans the noise :param: text :type str :return: cleaned_text :type str

dolphin.datareader.**txt_to_text** (*file_path*, *dolower*)

Extracts plain text from txt files :param file_path :type str :return:text :type str

class dolphin.document_categorization.doc_classifier_test.**DocClassifier** (*tfidfmodelpath*, *classifiermodelpath*, *nlp*)

This class is developed for the preprocessing of textual contents in any document and use such preprocessed content for the prediction of category of the document using tfidf vectorizer developed and SVC classifier model developed

Param tfidfmodelpath: The path of the tfidf vectorizer developed from the training data

Type str

Param classifiermodelpath: The path of the classifier model developed by training SVC classifier

classify (*file_content*)

Function to apply preprocessed text to the document categorization model :param: file_content: textual file content of the document :type: str :return: category of the document :rtype: str

preprocess_words (*texts, stop_words, bigram_mod, trigram_mod, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']*)

Remove Stopwords, Form Bigrams, Trigrams and Lemmatization :param: texts: list of tokens in the document type: list :param: stop_words: list of stopword from nltk corpus :type: list :param: bigram_mod: bigrams from document evaluated using gensim Phraser :type: list :param: trigram_mod: trigrams from document evaluated using gensim Phraser :type: list :param: allowedpos_tags: list of allowed POS tags in the document :type: list :return: preprocessed list of tokens from document :rtype: list

sent_to_words (*file_content*)

remove emails and preprocess the sentences in the document using gensim simple_preprocess function :param: file_content: the textual content of the document :type: str :return: list of preprocessed sentences :rtype: list

class dolphin.segmentation.segmentresume.**ResumeSegmentCreator**

This class is developed for the segmentation of resumes into several segments. The segments are personal info, objective, skills, experiences, projects, academics, rewards, languages, references and links

format_segment (*text*)

This function formats the resume segments :param: text : textual content of the resume : type:

str :return: resume segments extracted :rtype: dict

sliced_resume_text ( )

This function returns the sliced resume text return: sliced text rtype: dict

unique_index_headings ( )

This function extracts the unique index headings from the resume content return: unique index headings rtype: list

dolphin.stanfordNER.formatdata.**formatPersonalinfo** (*personal_info*)

This functions takes all the unstructured personal information and returns the formatted and structured personal information. :param personal_info: (List) of all the fields required as personal info. :return: List(Returns the formatted personal info will all the required fields.)

class dolphin.stanfordNER.namedentityrecognition.**StanfordNER**

This class applies the trained custom NER model to return the academics, experiences and personal info from any resume

static education_parser (*tagged_tuples*, *text*)

** Function to gather all the education predicted tokens and format the tokens** param: tagged_tuples: tagged_tuples from NER model type: list of tuples param: text: textual content to be applied for education parsing type: str

static experience_parser (*tagged_tuples*, *text*)

Function to get out the experience related tokens from NER mmodel and format the parsed information param: tagged_tuples: tagged_tuples from NER model type: list of tuples param: text: textual content to be applied for experience parsing type: str return: formatted_data rtype: dict

static ner_parser (*model*, *text*, *mode*)

NER model that chooses the particular model and parser param: model: Stanford NLP model param: text: textual content of the resume type: str param: mode: to select the respective parser type: str

static personal_info_parser (*tagged_tuples*, *text*)

Function to be applied for the parsing of personal information from resumes param: tagged_tuples: tagged_tuples from NER model type: list of tuples param: text: textual content to be applied for experience parsing type: str

dolphin.stanfordNER.namedentityrecognition.**gather_ner** (*tagged_tuples*, *ner='DESIG'*)

This method gathers the predicted tokens related to academics, experiences and personal info from NER :param:tagged_tuples :list of tuples :return : list of entities :rtype:list

dolphin.stanfordNER.standarizedata.**capitalizeinput** (*inputparameter*)

This will take the raw input text and tokenize the string and capitalize all the token and returns the formatted capitalized text. :param inputparameter :type str :return: string(text will all the tokens capitalized)

dolphin.stanfordNER.standarizedata.**comparedates** (*datelist*)

this function takes the list of the date parse them and compares them and provide larger date and smaller date :param datelist:type list of str :return: sorted dates

class dolphin.pipeline.**NlpPipeline**

This class is developed to add the custom spacy pipelines for document categorization, resume

segmentation, and resume parsing.

add_category_component ( )
Function to add document categorization component to the spacy custom pipeline

add_ner_parsing ( )
Function for adding custom NER pipeline in nlp model

add_pattern_matching ( )
Function for adding pattern matching component in pipeline

add_segmentation_component ( )
Function to add resume segmentation component to the spacy custom pipeline

category_component (*doc*)
custom function to add resume identification component to the pipeline params: spacy Doc of textual data return : Doc object with category embedded

ner_parsing_component (*doc*)
Function for preparing custom ner parsing component params: doc of content from file return: doc containing parsed information from keyword

pattern_matching_component (*doc*)
Function for preparing pattern matching component in spacy params: doc from document return : doc containing matched keywords

process_text (*content*)
Function to process as spacy doc

segmentation_component (*doc*)
custom function to add resume identification component to the pipeline params: spacy Doc of textual data return : Doc object with category embedded

- *Index*
- *Module Index*
- *Search Page*

d

A

C

D

E

F

G

I

N

P

R

S

T

U