ДОДАТОК А

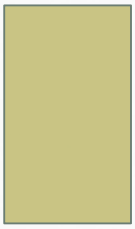
Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки
кафедра ЕОМ

МОДЕЛЬ КЛАСИФІКАЦІЇ ТЕКСТОВИХ ДОКУМЕНТІВ ІЗ АДАПТИВНИМ ОНОВЛЕННЯМ КЛЮЧЕВИХ СЛІВ ВИХІДНИХ КЛАСІВ

Кваліфікаційна робота

Другий рівень (магістр)



Автор
Могилевський Д.І.
ст. гр. СПм-21-2

Керівник
Барковська О.Ю.
доц. каф. ЕОМ

АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

- збільшення обсягів інформації у цифровому вигляді;
- зростання вимог щодо швидкості обробки та доступу до інформації;
- підвищення продуктивності в пошуку інформації;
- широкий спектр областей застосування.

Сфери застосування:

ОГЛЯД БІБЛІОТЕК NLP

	Переваги	Недоліки
Natural Language ToolKit	-найпопулярніша та повна NLP бібліотека; -швидка токенізація речень; -підтримує найбільшу кількість мов порівняно з іншими бібліотеками.	-повільна; -не підтримує моделі нейронних мереж;
SpaCy	-найшвидший NLP фреймворк; -використовує нейронні мережі для навчання деяких моделей;	-токенізація речень повільніша, ніж у NLTK; -не підтримує багато мов. Існують моделі тільки для 7 мов.
Gensim	-підтримує глибоке навчання; -працює з великими базами даних і обробляє потоки даних;	-призначений для неконтрольованого (unsupervised) моделювання тексту

3

МЕТА КВАЛІФІКАЦІЙНОЇ РОБОТИ

- є роздробка моделі яка буде класифікувати кваліфікаційні роботи. Також протестувати вплив методів препроцесінгу на класифікацію

4

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

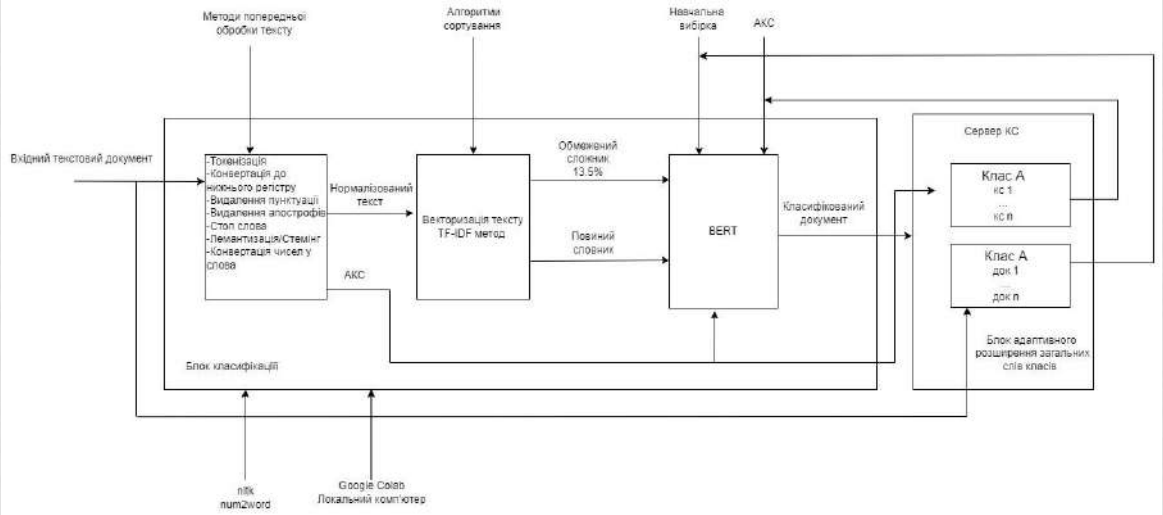
- аналіз предметної області;
- визначення діапазону TF словнику, у який потрапляють авторські ключові слова;
- розробка моделі класифікації із адаптивним оновленням ключових слів;
- експериментальні дослідження для визначення впливу лематизації, стемінгу та кількості значущих слів на точність класифікації;
- аналіз отриманих результатів.

5

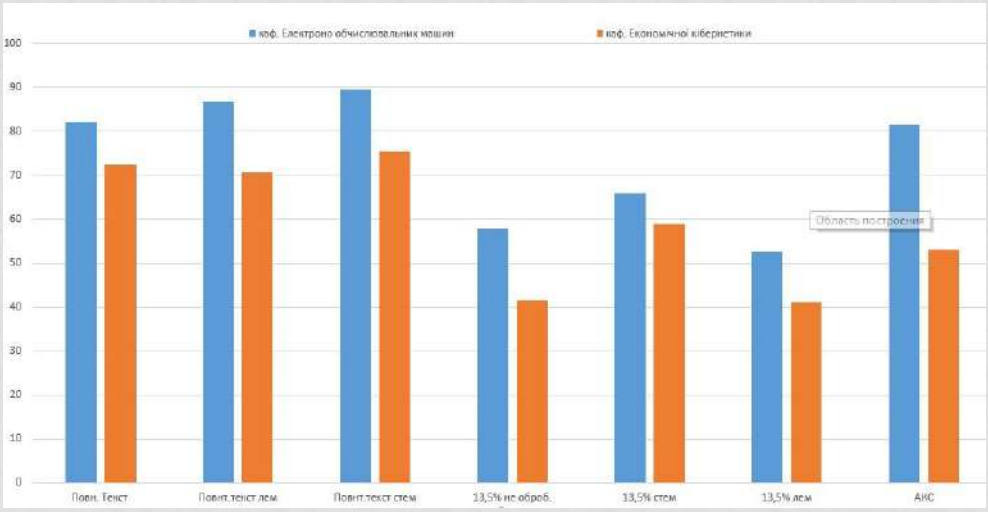
РІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ. МОДЕЛЬ СИСТЕМИ ВПОРЯДКОВАНОГО ЗБЕРІГАННЯ ДОКУМЕНТІВ

6

МОДЕЛЬ ЗА АДАПТИВНИМ ОНОВЛЕННЯМ КЛЮЧОВИХ СЛІВ



ТЕСТИ КЛАСИФІКАЦІЇ



ВИСНОВКИ

В ході кваліфікаційної роботи було проведено аналіз предметної області. Приведені приклади використання класифікаторів у різних галузях.

Було запропоновано підхід для визначення значущих слів документу для подальшого їх використання у процесі класифікації в якості вектора ознак.

Також в ході кваліфікаційної роботи були проведені експерементальні досліджения для виявлення впливу лематизації, стемінгу та кількості значущих слів на точність класифікації.

Також була запропонована модель класифікації текстових документів з адаптивним оновленням ключових слів.

9

ПУБЛІКАЦІЇ

Olesia BARKOVSKA, Dmytro MOHYLEVSKYI, Yuliia IVANENKO, Dmytro ROSINSKIY "WAYS TO DETERMINE THE RANGE OF KEYWORDS IN A FREQUENCY DICTIONARY FOR TEXT CLASSIFICATION" , International scientific journal "Computer Systems and Information Technologies", 2023, No 1, pp.14-20.

Olesia Barkovska, Vladyslav Kholiev, Anton Havrashenko, Dmytro Mohylevskyi , Andriy Kovalenko, A Conceptual Text Classification Model Based on Two-Factor Selection of Significant Words. Proceedings of the 7th International Conference on Computational Linguistics and Intelligent Systems. Volume II: Computational Linguistics Workshop, Kharkiv, Ukraine, April 20-21, 2023, 244-255

10

ДОДАТОК Б

Код програми

Лістинг Б.1 – parser.py

```
import pdfplumber
import os

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer , WordNetLemmatizer
from collections import Counter
from num2words import num2words

import numpy as np

def convert_lower_case(data):
    return np.char.lower(data)

def remove_stop_words(data):
    stop_words = stopwords.words('english')
    words = word_tokenize(str(data))
    new_text = ""
    for w in words:
        if w not in stop_words and len(w) > 1:
            new_text = new_text + " " + w
    return new_text

def remove_punctuation(data):
    symbols = "!\"#$%&()*+-./:;<=>?@[\]^_`{|}~\n"
    for i in range(len(symbols)):
        data = np.char.replace(data, symbols[i], ' ')
        data = np.char.replace(data, "  ", " ")
    data = np.char.replace(data, ',', '')
    return data

def remove_apostrophe(data):
    return np.char.replace(data, "'", "")

def lemmatization(data):
    lem = WordNetLemmatizer()

    tokens = word_tokenize(str(data))
    new_text = ""
    for w in tokens:
        new_text = new_text + " " +  lem.lemmatize(w)
    return new_text

def stemming(data):
    stemmer= PorterStemmer()

    tokens = word_tokenize(str(data))
    new_text = ""
    for w in tokens:
```

```
            new_text = new_text + " " + stemmer.stem(w)
        return new_text

    def convert_numbers(data):
        tokens = word_tokenize(str(data))
        new_text = ""
        for w in tokens:
            try:
                w = num2words(int(w))
            except:
                a = 0
            new_text = new_text + " " + w
        new_text = np.char.replace(new_text, "-", " ")
        return new_text

    def preprocessSTEM(data):
        data = convert_lower_case(data)
        data = remove_punctuation(data) #remove comma seperately
        data = remove_apostrophe(data)
        data = remove_stop_words(data)
        data = convert_numbers(data)
        data = stemming(data)
        data = remove_punctuation(data)
        data = convert_numbers(data)
        data = stemming(data) #needed again as we need to stem the words
        data = remove_punctuation(data) #needed again as num2word is giving
few hypens and commas fourty-one
        data = remove_stop_words(data) #needed again as num2word is giving
stop words 101 - one hundred and one
        return data

    def preprocessLEM(data):
        data = convert_lower_case(data)
        data = remove_punctuation(data) #remove comma seperately
        data = remove_apostrophe(data)
        data = remove_stop_words(data)
        data = convert_numbers(data)
        data = lemmatization(data)
        data = remove_punctuation(data)
        data = convert_numbers(data)
        data = lemmatization(data) #needed again as we need to stem the words
        data = remove_punctuation(data) #needed again as num2word is giving
few hypens and commas fourty-one
        data = remove_stop_words(data) #needed again as num2word is giving
stop words 101 - one hundred and one
        return data

    def preprocess(data):
        data = convert_lower_case(data)
        data = remove_punctuation(data) #remove comma seperately
        data = remove_apostrophe(data)
        data = remove_stop_words(data)
        data = convert_numbers(data)
        data = remove_punctuation(data)
        data = convert_numbers(data)
        data = remove_punctuation(data) #needed again as num2word is giving
few hypens and commas fourty-one
        data = remove_stop_words(data) #needed again as num2word is giving
stop words 101 - one hundred and one
        return data

    dir = os.listdir('./Mag_EOM')
    count = 0
    referat_word = []
```

```
diplom = str()


docNum="30"
stopcycle = 68
with pdfplumber.open("./testdoc/"+docNum+".pdf") as pdf:
    for i in range( 7,len(pdf.pages)):
        tmp = pdf.pages[i].extract_words()
        for j in range( i,len(pdf.pages)):
            tmp2 = pdf.pages[j].extract_words()


            diplom = diplom + pdf.pages[j].extract_text()


            if j == stopcycle:
                print("----- break")
                break

        if j == stopcycle:
            print("----- break")
            break


with open(docNum+"_allDiplom.txt", "w"  ,encoding='utf-8') as file:
    file.write(diplom)

processed_textSTEM = word_tokenize(str(preprocessSTEM(diplom)))
processed_textLEM = word_tokenize(str(preprocessLEM(diplom)))
processed_text = word_tokenize(str(preprocess(diplom)))

def func(processed_text, type, mode=False):
    DF = {}
    for i in range(len(processed_text)):
        tokens = processed_text[i]

        try:
            DF[tokens].add(i)
        except:
            DF[tokens] = {i}

    for i in DF:
        DF[i] = len(DF[i])

    def doc_freq(word):
        c = 0
        try:
            c = DF[word]
        except:
            pass
        return c
    doc = 0



    tokens = processed_text

    counter = Counter(processed_text)
    words_count = len( processed_text)
    list_word = []
    list_tf = []
    i=0

    for token in np.unique(tokens):
```

```
                tf = counter[token]/words_count
                list_tf.append(tf)
                list_word.append(token)

            n = len(list_tf)
            swapped = False
            for i in range(n-1):
                for j in range(0, n-i-1):
                    if list_tf[j] < list_tf[j + 1]:
                        swapped = True
                        list_tf[j], list_tf[j + 1] = list_tf[j + 1], list_tf[j]
                        list_word[j],  list_word[j  +  1]  =  list_word[j  +  1],
list_word[j]

                if not swapped:
                    break


            with open(docNum+type+"_onlywords.txt", "w",encoding='utf-8') as file:
                for i in range(0,235):
                    file.write(f"{list_word[i]}\n")
            if mode:
                with  open(docNum+type+"_onlywords.txt",  "w",encoding='utf-8')  as
file:
                    for i in range(len(list_word)):
                        file.write(f"{list_word[i]}\n")

        dlem = lemmatization(diplom)
        with open(docNum+"_allDiplomLem.txt", "w"  ,encoding='utf-8') as file:
            file.write(dlem)

        dstem = stemming(diplom)
        with open(docNum+"_allDiplomStem.txt", "w"  ,encoding='utf-8') as file:
            file.write(dstem)

        func(processed_textSTEM, "STEM")
        func(processed_textLEM, "LEM")
        func(processed_text, "Ordinary_words")
        func(processed_text, "Ordinary_wordsALL", True)
```

Лістинг Б.2 – train.py

```
        import torch
        from  transformers.file_utils  import  is_tf_available,  is_torch_available,
is_torch_tpu_available
        from transformers import BertTokenizerFast, BertForSequenceClassification
        from transformers import Trainer, TrainingArguments
        import numpy as np
        import random
        from sklearn.datasets import fetch_20newsgroups
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score


        def set_seed(seed: int):
            """
            Helper  function  for  reproducible  behavior  to  set  the  seed  in
``random``, ``numpy``, ``torch`` and/or ``tf`` (if
            installed).
```

```
        Args:
            seed (:obj:`int`): The seed to set.
        """
        random.seed(seed)
        np.random.seed(seed)
        if is_torch_available():
            torch.manual_seed(seed)
            torch.cuda.manual_seed_all(seed)
            # safe to call this function even if cuda is not available
        if is_tf_available():
            import tensorflow as tf
            tf.random.set_seed(seed)

    set_seed(1)

    # the model we gonna train, base uncased BERT
    #        check        text        classification        models        here:
https://huggingface.co/models?filter=text-classification
    model_name = "bert-base-uncased"
    # max sequence length for each document/sentence sample
    max_length = 512
    # load the tokenizer
    tokenizer          =          BertTokenizerFast.from_pretrained(model_name,
do_lower_case=True)

    def read_20newsgroups(test_size=0.2):
        # download & load 20newsgroups dataset from sklearn's repos
        dataset      =      fetch_20newsgroups(subset="all",      shuffle=True,
remove=("headers", "footers", "quotes"))
        documents = dataset.data
        labels = dataset.target
        # split into training & testing a return data as well as label names
        return   train_test_split(documents,   labels,   test_size=test_size),
dataset.target_names

    # call the function
    (train_texts,  valid_texts,  train_labels,  valid_labels),  target_names  =
read_20newsgroups()
    # tokenize the dataset, truncate when passed `max_length`,
    # and pad with 0's when less than `max_length`
    train_encodings  =  tokenizer(train_texts,  truncation=True,  padding=True,
max_length=max_length)
    valid_encodings  =  tokenizer(valid_texts,  truncation=True,  padding=True,
max_length=max_length)

    class NewsGroupsDataset(torch.utils.data.Dataset):
        def __init__(self, encodings, labels):
            self.encodings = encodings
            self.labels = labels

        def __getitem__(self, idx):
            item    =    {k:    torch.tensor(v[idx])    for    k,    v    in
self.encodings.items()}
            item["labels"] = torch.tensor([self.labels[idx]])
            return item

        def __len__(self):
            return len(self.labels)

    # convert our tokenized data into a torch Dataset
    train_dataset = NewsGroupsDataset(train_encodings, train_labels)
    valid_dataset = NewsGroupsDataset(valid_encodings, valid_labels)
    # load the model and pass to CUDA
```

```
      model      =      BertForSequenceClassification.from_pretrained(model_name,
num_labels=len(target_names)).to("cuda")

      def compute_metrics(pred):
          labels = pred.label_ids
          preds = pred.predictions.argmax(-1)
          # calculate accuracy using sklearn's function
          acc = accuracy_score(labels, preds)
          return {
            'accuracy': acc,
          }

      training_args = TrainingArguments(
          output_dir='./results',          # output directory
          num_train_epochs=3,              # total number of training epochs
          per_device_train_batch_size=8,     #  batch  size  per  device  during
training
          per_device_eval_batch_size=20,   # batch size for evaluation
          warmup_steps=500,                # number of warmup steps for learning
rate scheduler
          weight_decay=0.01,               # strength of weight decay
          logging_dir='./logs',            # directory for storing logs
          load_best_model_at_end=True,      # load the best model when finished
training (default metric is loss)
          # but you can specify `metric_for_best_model` argument to change to
accuracy or other metric
          logging_steps=400,                        #  log  &  save  weights  each
logging_steps
          save_steps=400,
          evaluation_strategy="steps",     # evaluate each `logging_steps`
      )

      trainer = Trainer(
          model=model,                           # the instantiated Transformers
model to be trained
          args=training_args,                     # training arguments, defined
above
          train_dataset=train_dataset,         # training dataset
          eval_dataset=valid_dataset,          # evaluation dataset
          compute_metrics=compute_metrics,        #  the  callback  that  computes
metrics of interest
      )
      # train the model
      trainer.train()
      # evaluate the current model after training
      trainer.evaluate()
      # saving the fine tuned model & tokenizer
      model_path = "20newsgroups-bert-base-uncased"
      model.save_pretrained(model_path)
      tokenizer.save_pretrained(model_path)
```