Dokumentation Zuständigkeiten-Identifikator

1. Datei Hochladen

Die Benutzer werden aufgefordert, ihre Dateien zum Klassifizieren hochzuladen. Dies wird mit der Funktion *upload_file()* realisiert. Wenn eine Datei hochgeladen wird, dann ist diese Datei temporell lokal mit einem User-Prefix gespeichert, welcher randomisiert von der Session generiert wird. Abschließend wird das Ergebnis auf der URL der Web-Applikation wiedergegeben (siehe Listing 1).

```
@app.route('/register', methods=['POST'])
def upload_file():
    ctext = ''
    infotext = ''
    notext = ''
    user_id = session['user_id']
    print('user_id1111', user_id)
    if request.method == 'POST':

        if 'file' not in request.files:
            return redirect(request.url)
        file = request.files['file']
        if file.filename == '':
            # ctext='No file selected for uploading'
            return redirect(request.url)
        if file and not allowed_file(file.filename):
            notext = 'Dieses Dateiformat wird nicht unterstützt. Bitte
wählen Sie ein anderes Dateiformat.'
        if file and allowed_file(file.filename):
            notext = ''
            filename = secure_filename(file.filename)
            filename = user_id + filename
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            ctext = 'Ihre Datei "' + file.filename + '" wurde erfolgreich
hochgeladen!'
            infotext = 'Für die von Ihnen hochgeladene Datei wurde die
Zuständigkeit ermittelt. Bitte klicken Sie oben auf den Button
„Zuständigkeit prüfen“ und prüfen Sie im zweiten Schritt die für die Datei
ermittelte Zuständigkeit.'

        return render_template('index.html', ctext=ctext, infotext=infotext,
notext=notext)
```

Listing 1.: Funktion upload_file()

2. Lesen der Dateien und Klassifizierung der Zuständigkeit

Mithilfe der Funktion *save()* werden die temporell lokal gespeicherten Dateien der Benutzer erkannt.

Ist die Datei eine PDF-Datei, wird mithilfe der Python-Bibliothek „pdfminer“ der Text extrahiert.

Ist die Datei eine E-Mail-Datei, wird mithilfe der Python-Bibliothek „extract_msg“ der Text extrahiert.

Ist die Datei eine Audio-Datei, wird mithilfe von vortrainierten Transformer Modellen zur Spracherkennung der Text extrahiert.

Abschließend wird mithilfe von vortrainierten Transformer Modellen zur Textklassifizierung (Zero-Shot-Klassifizierung) die Zuständigkeit erkannt und das Ergebnis als Vorschlag auf der URL der Web-Applikation wiedergegeben. Der Benutzer hat die Möglichkeit, wenn die vorgeschlagene Zuständigkeit nicht korrekt ist, von einem Dropdown-Menü die richtige auszuwählen (siehe Listing 2).

```
@app.route('/save')
def save():
    user_id = session['user_id']
    print('user_id22222', user_id)
    fileread1 = None
    thema1 = None
    thema2 = None
    thema3 = None
    thema = None

    for file in os.listdir("/opt/ulm/uploads"):
        if user_id in file:
            print('ichbin hier')
            if file.endswith(".txt"):

                txtfile = "/opt/ulm/uploads/" + file
                print("txtfile", txtfile)
                with open(txtfile, encoding='utf8', errors='ignore') as inp:
                    data = inp.read().replace('\n', ' ')
                    inp.close()
                print('data', data)
                fileread1 = data
                labels = ["Meldewesen", "Ausländerwesen", "Standesamt"]
                hypothesis_template = "In diesem Satz geht es um das Thema
{}."
                prediction = zershot_pipeline(fileread1, labels,
hypothesis_template=hypothesis_template)

                print(prediction)

                thema = prediction['labels'][0]
                print(thema)
                if thema == 'Meldewesen':
```

```
                    thema1 = 'Meldewesen'
                    thema2 = 'Ausländerwesen'
                    thema3 = 'Standesamt'

                if thema == 'Ausländerwesen':
                    thema1 = 'Ausländerwesen'
                    thema2 = 'Meldewesen'
                    thema3 = 'Standesamt'
                if thema == 'Standesamt':
                    thema1 = 'Standesamt'
                    thema2 = 'Meldewesen'
                    thema3 = 'Ausländerwesen'
                print(fileread1)
            if file.endswith(".msg"):

                msgfile = "/opt/ulm/uploads/" + file
                print("msgfile", msgfile)
                msg = extract_msg.Message(msgfile)
                msg_sender = msg.sender
                msg_date = msg.date
                msg_subj = msg.subject
                fileread1 = msg.body
                msg.close()
                labels = ["Meldewesen", "Ausländerwesen", "Standesamt"]
                hypothesis_template = "In diesem Satz geht es um das Thema
{}."
                prediction = zershot_pipeline(fileread1, labels,
hypothesis_template=hypothesis_template)

                print(prediction)

                thema = prediction['labels'][0]
                if thema == 'Meldewesen':
                    thema1 = 'Meldewesen'
                    thema2 = 'Ausländerwesen'
                    thema3 = 'Standesamt'

                if thema == 'Ausländerwesen':
                    thema1 = 'Ausländerwesen'
                    thema2 = 'Meldewesen'
                    thema3 = 'Standesamt'
                if thema == 'Standesamt':
                    thema1 = 'Standesamt'
                    thema2 = 'Meldewesen'
                    thema3 = 'Ausländerwesen'
                print(thema1)

                print(fileread1)

            if file.endswith(".pdf"):

                pdffile = "/opt/ulm/uploads/" + file
                print("pdffile", pdffile)
                text = extract_text(pdffile)
                fileread1 = str(text)
                print(fileread1)
                labels = ["Meldewesen", "Ausländerwesen", "Standesamt"]
                hypothesis_template = "In diesem Satz geht es um das Thema
{}."
                prediction = zershot_pipeline(fileread1, labels,
hypothesis_template=hypothesis_template)
```

```
                print(prediction)

                thema = prediction['labels'][0]
                if thema == 'Meldewesen':
                    thema1 = 'Meldewesen'
                    thema2 = 'Ausländerwesen'
                    thema3 = 'Standesamt'

                if thema == 'Ausländerwesen':
                    thema1 = 'Ausländerwesen'
                    thema2 = 'Meldewesen'
                    thema3 = 'Standesamt'
                if thema == 'Standesamt':
                    thema1 = 'Standesamt'
                    thema2 = 'Meldewesen'
                    thema3 = 'Ausländerwesen'
                print(thema1)

            if file.endswith(".wav"):
                wavfile = "/opt/ulm/uploads/" + file

                pipe = pipeline(task="automatic-speech-recognition",
model=model1, tokenizer=tokenizer1,
                                feature_extractor=feature_extractor1)

                fileread1 = pipe(wavfile, chunk_length_s=10)
                fileread1 = fileread1['text']
                # load model and tokenizer

                # load any audio file of your choice
                # speech, rate = librosa.load(wavfile, sr=16000)

                # input_values = tokenizer1(speech,
return_tensors='pt').input_values
                # Store logits (non-normalized predictions)
                # logits = model1(input_values).logits

                # Store predicted id's
                # predicted_ids = torch.argmax(logits, dim=-1)
                # decode the audio to generate text
                # fileread1 = tokenizer1.decode(predicted_ids[0])
                print(fileread1)

                labels = ["Meldewesen", "Ausländerwesen", "Standesamt"]
                hypothesis_template = "In diesem Satz geht es um das Thema
{}."
                prediction = zershot_pipeline(fileread1, labels,
hypothesis_template=hypothesis_template)

                print(prediction)

                thema = prediction['labels'][0]
                if thema == 'Meldewesen':
                    thema1 = 'Meldewesen'
                    thema2 = 'Ausländerwesen'
                    thema3 = 'Standesamt'

                if thema == 'Ausländerwesen':
                    thema1 = 'Ausländerwesen'
                    thema2 = 'Meldewesen'
```

```
                    thema3 = 'Standesamt'
                if thema == 'Standesamt':
                    thema1 = 'Standesamt'
                    thema2 = 'Meldewesen'
                    thema3 = 'Ausländerwesen'
                print(thema1)

            # if fileread1 is not None:
            # return render_template('save.html', fileread1=fileread1,
thema1=thema1, fileread2=fileread2, thema2=thema2,
            # fileread3=fileread3, thema3=thema3)

            # data_dic = {
            # 'id': fileread,
            # 'color': thema}
            # columns = ['id', 'color']
            # index = ['1']

            # df = pd.DataFrame(data_dic, columns=columns, index = index)
            # table = df.to_html(index=False)
    return render_template('save.html', fileread1=fileread1, thema1=thema1,
thema2=thema2, thema3=thema3)
```

Listing 2.: Funktion save()

3. Speichern in der Datenbank

Der Text und die dazugehörige Zuständigkeit wird mithilfe der Funktion save_db() und der Python-Bibliothek „flask_sqlalchemy“ in einer SQLite Datenbank gespeichert (siehe Listing 3).

Die gespeicherten Daten dienen als Trainingsdaten für den zweiten Schritt des Prototyps, nämlich einen eigenen Klassifizierungsalgorithmus zu entwickeln.

```
@app.route('/save_db', methods=['POST', 'GET'])
def save_db():
    user_id = session['user_id']
    print('user_id3333', user_id)
    if request.method == "POST":
        qtc_data = request.get_json()
        print('qtc_data', qtc_data)
        print('qtc_data', qtc_data[0]['text'])
        print('qtc_data', qtc_data[1]['thema'])
        db.session.add(Store_QTc_data(text=qtc_data[0]['text'],
thema=qtc_data[1]['thema']))
        db.session.commit()
        ctext = 'Dieser Datensatz wurde erfolgreich gespeichert. Wenn Sie
eine weitere Datei hochladen möchten, beginnen Sie bitte wieder mit dem
ersten Schritt „Datei laden“.'

        print('ctext', ctext)
        for f in os.listdir("/opt/ulm/uploads"):
            if user_id in f:
```

```
                os.remove("/opt/ulm/uploads/" + f)
    return jsonify({'response': ctext})
```

Listing 3.: Funktion save_db()