Verifying and Validating College IDs Documentation

Release 0.1

eYSIP2020

Jul 10, 2020

CONTENTS

CHAPTER

ONE

VERIFYING AND VALIDATING COLLEGE IDS

The objective of this project is to build an intelligent system which correctly identifies a valid colIege ID card from a pool of ID card images and extracts textual data to verify against the details provided by the user

1.1 Project Organization

├── LICENSE ├── Makefile <- Makefile with commands like *make data* or *make train* ├── README.md <- The top-level README for developers using this project. ├── data │ ├── external <- Data from third party sources. │ ├── interim <- Intermediate data that has been transformed. │ ├── processed <- The final, canonical data sets for modeling. │ └── raw <- The original, immutable data dump. │ ├── docs <- A default Sphinx project; see sphinx-doc.org for details │ ├── models <- Trained and serialized models, model predictions, or model summaries │ ├── notebooks <- Jupyter notebooks. Naming convention is a number (for ordering), │ the creator's initials, and a short - delimited description, e.g. │ *1.0-jqp-initial-data-exploration*. │ ├── references <- Data dictionaries, manuals, and all other explanatory materials. │ ├── reports <- Generated analysis as HTML, PDF, LaTeX, etc. │ └── figures <- Generated graphics and figures to be used in reporting │ ├── requirements.txt <- The requirements file for reproducing the analysis environment, e.g. │ generated with *pip freeze > requirements.txt* │ ├── setup.py <- makes project pip installable (pip install -e .) so src can be imported ├── src <- Source code for use in this project. │ ├── __init__.py <- Makes src a Python module │ │ │ ├── data <- Scripts to download or generate data │ │ └── make_dataset.py │ │ │ ├── features <- Scripts to turn raw data into features for modeling │ │ └── build_features.py │ │ │ ├── models <- Scripts to train models and then use trained models to make │ │ │ predictions │ │ ├── predict_model.py │ │ └── train_model.py │ │ │ └── visualization <- Scripts to create exploratory and results oriented visualizations │ └── visualize.py │ └── tox.ini <- tox file with settings for running tox; see tox.readthedocs.io

1.2 Pre-Requisites

This project is built using Python language only, so make sure you have the latest version of Python. To get it (along with other essential packages) in Debian-based distribution run:

# apt install python3 git make

Now go ahead and clone this repository by running the command:

$ git clone https://github.com/eyic1eyantra/eysip2020-4-Verification-and-validation-ID.git

1.3 Setting Up

Most things are automated for you using [make](https://swcarpentry.github.io/make-novice/)

To see the available make recipes, run:

$ make help

To check if you have the required python version, run:

$ make test_environment

For now, the *create_environment* recipe does not work so we will create a virtual environment ourself. For the Debian-based systems, run:

$ sudo apt-get install python3-pip

$ python3 -m pip install –user virtualenv

Now make sure you are in the directory having this repo (or it's parent directory), and run:

$ python3 -m venv env

Here *env* is the name of the virtual environment, you can have any name you want. To activate the environment, run:

$ source env/bin/activate

Now, to install all the required python packages, run:

$ pip install -r requirements.txt

1.4 Getting the models

You should have gotten [*gdown*](https://pypi.org/project/gdown/) with other python packages. You can use it to download the models from Google drive by simply running:

$ python get_models_from_gdrive.py

1.5 Usage

The usage as of now is simple. You will need the trained models to run the code so make sure you have them (.pth files) in the *models* directory. Now, place all the images you have in the *data/raw* directory.

Then run the command:

$ python src/models/predict_model.py

This will take some time depending on your CPU and/or GPU capacities. You can go and grab a cup of coffee or read the output on the console which is designed to help you know what is going on.

After it's done executing, you will find that all images classified as college IDs (ID) are moved to the *data/interim* directory and the others (NON-ID) are left in the *data/raw* directory. Also, you will find a new file called *processing.log* in *src/models* which has the results of ID card classification and text recognition with confidence scores so be sure to check it out using your favorite text editor.

1.6 Using the WebApp

The WebApp of this project is supposed to act as point of interaction between the model and the user. The instructions for using the WebApp is given in the [WebApp specific README](webapp/README.md)

<p><small>Project based on the <a target=”_blank” href=”https://drivendata.github.io/cookiecutter-data-science/”>cookiecutter data science project template</a>. #cookiecutterdatascience</small></p>

CHAPTER

TWO

THE WEBAPP

This webapp is meant to be the point of user interaction with the project either through the Web UI or the REST Api.

2.1 Directory Structure

├── app │ ├── __init__.py │ ├── mod_api │ │ ├── batch_upload.py │ │ ├── __init__.py │ │ └── single_upload.py │ ├── mod_upload │ │ ├── batch_upload.py │ │ ├── __init__.py │ │ └── single_upload.py │ ├── static │ │ └── client │ │ └── csv │ │ └── batch_data_template.csv │ └── templates │ ├── _formhelpers.html │ └── upload │ ├── batch_upload.html │ └── single_upload.html ├── config.py ├── README.md └── run.py

2.2 Configuration

You need to add the configurations in the *config.py* The variables needed to set are:

- *CSRF_SESSION_KEY*
- *SECRET_KEY*
- *MYSQL_HOST*
- *MYSQL_USER*
- *MYSQL_PASSWORD*
- *MYSQL_DB*

2.3 Running the Development Server

Just be in this directory (*webapp*) and run:

python run.py

The development server should start at port *8080*.

2.4 Usage of WebApp

The webapp is designed to be intutive. After deplying the server, just visit to `http://<base_url>:8080/

2.5 Usage of REST Api

The REST Api can be used using the *curl* command.

Single User Uploads

First upload the image of the ID card of the user using the command:

curl -F "image=@<image_file>" <base_url>/api/image

After a successful upload, you will get an *image_name* in response. Use that *image_name* when uploading data of the user using the command:

curl –header "Content-Type: application/json" –data '{"name":"<name>", "college": "<college_name>", "department": "<department>", "year": <year>, "image": "<image_name>"}' <base_url>/api/data

Batch Upload

First upload a *zip* file having the images of ID cards of all the users using the command:

curl -F "images=@<zip_file>" <base_url>/api/batch_image

In response you will get *zip filename* which you need to pass when uploading the csv file.

Then download the template csv file using *wget* command:

wget -O template.csv <base_url>/upload/batch/download

Fill this template csv file with data. **Make sure that you provide the correct image name for the students. If the mapping of the student data and their ID card image name is not correct, it may get rejected by the classifier**

Then upload the filled csv file using the command:

curl -F "data=@<csv_file>" -F "zipfile=<zip filename>" <base_url>/api/batch_data

2.5.1 Single User REST Api

This module is used to upload per student data using REST Api.

class `app.mod_api.single_upload.`**`Data`**

Resource class having the api endpoints for uploading the data of a student.

Attributes

provide_automatic_options

representations

Methods

`as_view`(name, *class_args, **class_kwargs)	Converts the class into an actual view function that can be used with the routing system.
`dispatch_request`(self, *args, **kwargs)	Subclasses have to override this method to implement the actual view function code.
`get`(self)	Defines the HTTP GET method for the class
`post`(self)	Defines the HTTP POST method for the class

mediatypes	

get (*self*)
Defines the HTTP GET method for the class

Returns

json The return message for the user.

post (*self*)
Defines the HTTP POST method for the class

Returns

response [json] The return message for the user having validity score (> 0).

class `app.mod_api.single_upload.`**Image**
Resource class having the api endpoints for uploading the ID image of a student.

Attributes

provide_automatic_options

representations

Methods

`as_view`(name, *class_args, **class_kwargs)	Converts the class into an actual view function that can be used with the routing system.
`dispatch_request`(self, *args, **kwargs)	Subclasses have to override this method to implement the actual view function code.
`get`(self)	Defines the HTTP GET method for the class
`post`(self)	Defines the HTTP POST method for the class

mediatypes	

get (*self*)
Defines the HTTP GET method for the class

Returns

json The return message for the user.

post (*self*)
Defines the HTTP POST method for the class

Returns

response [json] The return message for the user.

`app.mod_api.single_upload.`**`allowed_file`**(*filename*, *extensions*)

Function to check is the given file name has a valid extensions.

Parameters

filename [string] A string having the full name of the file including the extension.

extensions [set] The set of all the allowed extensions for the given file

Returns

bool Is the extension of the file allowed or not

2.5.2 Batch Data REST Api

This module is used to upload bata in batches using REST Api.

`class` `app.mod_api.batch_upload.`**`BatchData`**

Resource class having the api endpoints for uploading the data of students in a csv file.

Attributes

provide_automatic_options

representations

Methods

`as_view`(name, *class_args, **class_kwargs)	Converts the class into an actual view function that can be used with the routing system.
`dispatch_request`(self, *args, **kwargs)	Subclasses have to override this method to implement the actual view function code.
`get`(self)	Defines the HTTP GET method for the class
`post`(self)	Defines the HTTP POST method for the class

mediatypes	

`get`(*self*)

Defines the HTTP GET method for the class

Returns

json The return message for the user.

`post`(*self*)

Defines the HTTP POST method for the class

Returns

response [json] The return message for the user having validity scores (> 0).

`class` `app.mod_api.batch_upload.`**`BatchImage`**

Resource class having the api endpoints for uploading the ID images of students in a zip file.

Attributes

provide_automatic_options

representations

Methods

`as_view`(name, *class_args, **class_kwargs)	Converts the class into an actual view function that can be used with the routing system.
`dispatch_request`(self, *args, **kwargs)	Subclasses have to override this method to implement the actual view function code.
`get`(self)	Defines the HTTP GET method for the class
`post`(self)	Defines the HTTP POST method for the class

mediatypes	

`get` (*self*)
Defines the HTTP GET method for the class

Returns

json The return message for the user.

`post` (*self*)
Defines the HTTP POST method for the class

Returns

response [json] The return message for the user.

`app.mod_api.batch_upload.`**`allowed_file`** (*filename*, *extensions*)
Function to check is the given file name has a valid extensions.

Parameters

filename [string] A string having the full name of the file including the extension.

extensions [set] The set of all the allowed extensions for the given file

Returns

bool Is the extension of the file allowed or not

2.5.3 Single User WebApp UI

This module is used to upload per student data using WebApp UI.

`class` `app.mod_upload.single_upload.`**`IDForm`** (*formdata=None*, *obj=None*, *prefix=''*, *data=None*, *meta=None*, ***kwargs*)
Class to build the form for collecting student data.

Attributes

name [TextField] Variable to hold the name entered by the user through the form

email [EmailField] Variable to hold the email entered by the user through the form

college_name [TextField] Variable to hold the college name entered by the user through the form

department [TextField] Variable to hold the department entered by the user through the form

year [IntegerField] Variable to hold the year entered by the user through the form

image [FileField] Variable to hold the image uploaded by the user through the form

image_by_url [TextField] Variable to hold the url entered by image to fetch the image

Methods

upload_form()	Process the data uploaded by the user through form and return result

`upload_form``()`

Method to process the data uploaded by user through form and return the result.

It takes the data, applies validation checks then pass it to the models to get a validity score which is shown to the user.

Returns

html The rendered template file

`app.mod_upload.single_upload.`**`allowed_file`**`(`*filename*`)`

Function to check is the given file name has a valid extensions.

Parameters

filename [string] A string having the full name of the file including the extension.

extensions [set] The set of all the allowed extensions for the given file

Returns

bool Is the extension of the file allowed or not

2.5.4 Batch Data WebApp UI

`class` `app.mod_upload.batch_upload.`**`Batch_form`**`(`*formdata=None*, *obj=None*, *prefix=''*, *data=None*, *meta=None*, ***kwargs*`)`

Class to build the form for collecting student data in batch.

Attributes

csv_file [FileField] Variable to hold the csv uploaded by the user

images_zip [FileField] Variable to hold the zip file contating images uploaded by user

Methods

upload_batch()	Process the batch data uploaded by the user through form and return result

`upload_batch``()`

Method to process the batch data uploaded by user through form and return the result.

It takes the data, applies validation checks then pass it to the models to get a validity score which is shown to the user. All of this is done in a loop for all the data.

Returns

html The rendered template file

`app.mod_upload.batch_upload.`**`allowed_file`**(*filename*, *extensions*)

Function to check is the given file name has a valid extensions.

Parameters

filename [string] A string having the full name of the file including the extension.

extensions [set] The set of all the allowed extensions for the given file

Returns

bool Is the extension of the file allowed or not

`app.mod_upload.batch_upload.`**`download_csv`**`()`

Method to send the template csv for download.

Returns

file The template csv file as download

CHAPTER

THREE

CODES ASSOCIATED TO USING THE DEEP LEARNING MODELS

3.1 Bridge between WebApp and Models

This module is used to connect the webapp with the underlying models.

It takes the image path and the student data, passes the image through all the models sequentially, applies fuzzy string matching algorithm to generate a validity score.

`src.models.per_user_prediction.`**`get_validity`**`(`*student_data*, *image_name*`)`

Function to pass the image through underlying models and apply fuzzy string matching to get validity score.

Parameters

student_data [dict] A dictionary having the data entered by the student.

image_name [string] The name of the image of ID card associated with the student

Returns

validity_score [int] The score out of hundred telling how valid is the given ID card.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX