CSCI 8360 Data Science Practicum
Department of Computer Science
University of Georgia

Project 2: Ethical Facial Recognition

DUE: Thursday, March 18 by 11:59:59pm

Out February 25, 2021

1 Overview

You've all done extraordinarily well with "document" classification. Now on to image processing!

Your task is to design an algorithm that can determine whether a face in a given image is *male* or *female*. Note: we are not inferring gender from images, as this would be impossible without asking the individuals directly. Instead, you will be building image classifiers that leverage the spatial properties of the images, plus any of the precomputed features that are being made available, to classify the faces according to their sex.

2 Data

The dataset is the *Diversity in Faces* (DiF) dataset that we discussed in our lecture on ethics. Go read the paper to familiarize yourself:

`Merler, Michele, Nalini Ratha, Rogerio S. Feris, and John R. Smith. Diversity in faces. arXiv preprint arXiv:1901.10436 (2019).`

The data are all available on GCP: `gs://uga-dsp/project2` . In that parent folder, you'll find two subfolders: `images` and `files`.

- `images` contains hundreds of thousands of jpg images–just over 800,000, to be exact! These images are not large by any stretch; most aren't any bigger than 500×600 or so, and hardly any exceed 100KB in size. But some of these images contain multiple faces, which is why you'll have some additional files.
- `files` contains several `.csv` comma-separated text files that stipulate what images and faces are part of what dataset. They also contain some additional precomputed features on the images, courtesy of the original dataset source.

The images are "every day" pictures: some candid, others posed; some with one person, others with many. Figure 2.1 is an example image in the dataset with a single subject, while Figure 2.2 has a couple of faces in it. The quality of the images also varies, sometimes dramatically. Again, these are the kinds of pictures your average person would take! This is where the importance of the associated CSV files comes in: not only do they provide you with a listing of which image files are in which datasets (small, large, training, testing), but it also provides myriad additional metadata to help with your image processing pipeline.

Each CSV file has a header in the first line that provides labels for each "column" of data. Each subsequent row is a single *face*: as in, there could be multiple rows that refer to the same image file, but which highlight different faces in that image! So how do you tell the difference?

Here's how these CSV files are organized:

- Column 0, aka "Unnamed: 0": this is the data point index, irrespective of anything else. You don't really need to keep track of this.
- Column 1, aka "Face ID": a unique ID per face in the dataset. You can use this to uniquely identify data points.
- Column 2, aka "Image File": this is the string filename of an image in the `images` folder, where the face given by the previous Face ID can be found. You'll definitely want this for reading the image data!
- Columns 3-4: Image width and height. Nice for reference but you probably won't need it.
- Columns 5-8: Bounding box coordinates of the face! This way you don't need to find the face; you're given the (x, y) coordinate of the upper left corner, and the (x, y) coordinate of the lower right corner.

Those first 9 columns are all about getting the images and finding the faces. The next hundred or so columns are some pre-computed features you are welcome to use, ignore, or any combination therein. They were provided with the data, and given the very large number of images, it could be helpful to give you a start with some simple first-order

Figure 2.1: An image in the dataset with a single subject.

image features to get your models off the ground.

For more information on the information contained in these columns, read the paper! Here are the basic descriptions:

- Columns 9-144: 68 facial keypoints, listed as consecutive (x, y) pairs, computed using the `dlib` library. So they're listed as $x1, y1, x2, y2, ..., x68, y68$.
- Columns 145-152: 8 pairwise craniofacial distances.
- Columns 153-164: 12 craniofacial areas.
- Columns 165-172: 8 craniofacial ratios.
- Columns 173-174: 2 facial symmetry measurements.
- Columns 175-183: 9 facial contrast measurements.

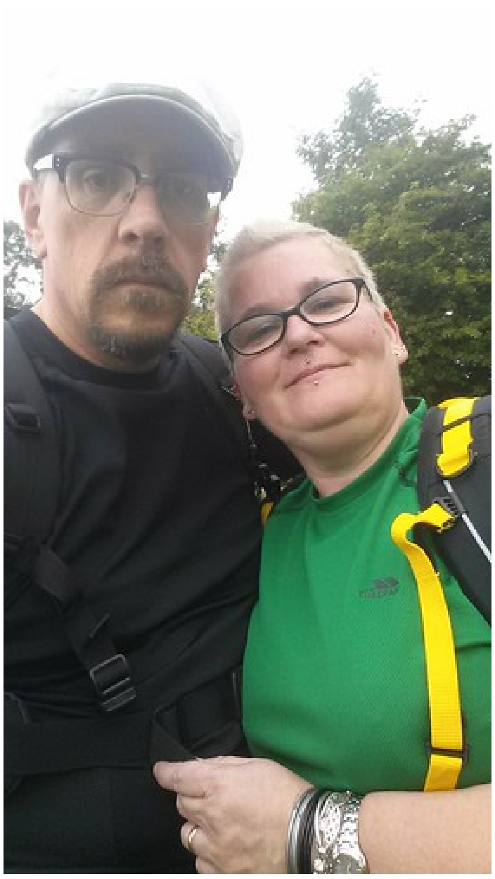
Figure 2.2: An image from the dataset with two subjects.

Again, you're welcome to use all, some, or none of these features in your models.

The last column, present in all BUT the `Xa_test`, `Xb_test`, and `Xc_test` files, is the label: a binary indicator of the sex of the person whose face is depicted. This is identified as **either a 0 (male) or 1 (female)**. These labels were placed manually, hence the "(subj)" descriptor in the header for the final column, to indicate that it was a subjective call on the part of another person.

(Fun aside: I stripped out quite a few image descriptors from the CSV files that had "-1" in this column, as that indicated either disagreement between the human labelers, or enough ambiguity that the labelers couldn't reach a decision. The beauty!horror of manual labeling!)

Your goal is to develop an image processing pipeline that maximizes classification accuracy of a person's sex (male or female) given an image, the bounding box of the face in question, and some other first-order properties.

Specifically, the predictions you will submit to AutoLab for evaluation are of datasets `Xa`, `Xb`, and `Xc`.

The average score of each testing sub-dataset will be posted on the AutoLab leaderboard.

Like in Project 1, the data are publicly accessible for viewing, and are **also accessible through Dataproc APIs so you *do NOT need to copy the data to your own storage!*** (this was a common point of confusion: if you give the Dataproc job the `gs://uga-dsp/project2` path, it'll access it just fine as if it were a local drive). You can also view the data in a web browser at the following URL:

`https://storage.googleapis.com/uga-dsp/project2/images/<img>`

Also remember: the file structure of the CSV files differ from the data files you had in Project 1. These are the files you have access to:

- `X_small_train.csv`
- `X_small_test.csv`
- `X_train.csv`
- `Xa_test.csv`, `Xb_test.csv`, `Xc_test.csv`

You may be asking: where are the `y*` files? Well, they aren't needed: by virtue of the CSV format, the labels are included within each `X*` file... **EXCEPT** the `Xa`, `Xb`, and `Xc` test files. Even the `X_small_test.csv` file has the labels included at the end, so you can verify how your model does on the small dataset. But the three final files are one column shorter than the others, as they are missing the ground truth 0 or 1 labels for each face.

That's what you'll need to predict.

3 Theory Guidelines

We're solidly in imaging territory, which means Convolutional Neural Networks (CNNs) are the Big Thing$^{\mathrm{TM}}$.

That said, a good starting point here would actually be to forgo any deep learning model and instead train a linear model on the features given in the CSV files! Obviously you'll want to strip out the final column on the training datasets and small testing dataset, but this would be an easy way to establish a baseline. However, given the sheer volume of images, you may still need to use something like Spark or dask to handle the training of these models in a scalable way. Fortunately, both Spark and dask have parallelized DataFrame implementations, which CSV file formats lend themselves easily to. From there, any of the built-in classification models (in Spark MLlib or dask-ML) should be a good starting point.

Beyond that, CNNs are going to be your friend. Generating intermediate feature maps with rich correspondence to the labels will bridge the gap between the limits of a linear classifier and the higher accuracy percentages. **CNNs and custom architectures robust to certain noisy features will be critical for reaching the higher echelons of accuracy.**

4 Engineering Guidelines

The gloves are off; anything is fair game. Use whatever framework you want, distributed or not. Just document and cite where you are pulling your work from, if you built on something that somebody else started.

Create a GitHub repository for you and your team to work on the project. You can make it private, and you can create a GitHub team within the DSP-UGA org, but the development must take place under the GitHub organization as I will be using it and the activity you record on it as part of the grading process. Name the repository with the scheme of `teamname-p2` to help differentiate from other teams and future projects. **Seriously, please name it correctly**.

Exercise good software design principles. This means: adopt a consistent coding style, document your code, use the GitHub ticketing system to identify milestones and flag bugs, and provide informative and frequent `git commit` comments. Also, be sure you include an informative `README`, an accurate `CONTRIBUTORS`, a `LICENSE`, and are organizing your code in a clean and logical way.

Shut down your GCP clusters when you are finished testing. GCP clusters incur costs for every hour they are turned on, even if they are not actively running any jobs. That said, it's also expensive to spin them back up if you're running a lot of jobs, so I would encourage you to turn them on, run a bunch of tests, and only when you're out of tests to run, you shut them down until you can get more in the pipeline.

5 Ethics Guidelines

New to this project: an ethics requirement! **READ CAREFULLY AND START EARLY.**

First, go check out and install the deon project. This is just a little command-line utility for generating standardized checklists for points about Data Science Ethics.

Second, everyone needs to run the utility and generate an `ETHICS.md` file to go in the root of their GitHub repository (along with the usual suspects: `CONTRIBUTORS`, `README`, etc).

This checklist should be checked off as you go through the project. If you're confused about certain points, go read the deon examples page for some concrete cases to consider for how the various items in the checklist can be violated. Everyone is also required to create a wiki page named "Ethics," and to provide answers / explanations / clarifications for how each item on the checklist was addressed in the project.

To do this, imagine you and your team have been hired by the University of Georgia Department of Human Resources to design a face-based campus-wide biometric system to replace the current iris scanner. As you are a data scientist who is well aware of the ethical implications of deploying such technologies at scale, you decide to be part of the solution, rather than part of the problem: you're going to make an effort to take the ethics of the situation into account, and build a facial recognition system that is as free of human bias as possible. Using this context, provide explanations and solutions for as many items in the Ethics Checklist as you can. Provide actual implementations wherever possible; everywhere else, use educated speculation (preferably with some citations of real-world examples using what you describe).

In addition to the "Diversity in Faces" paper cited at the beginning, if you want to learn more, read these:

```
A. Chardon, I. Cretois, and C. Hourseau. Skin colour typology and suntanning pathways.
Intl. Journal of Cosmetic Science, 13(4):191-208, 1991.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang.
Deep learning face attributes in the wild.
In IEEE Intl. Conf. on Computer Vision (ICCV), 2015.
```

6 Hints and Reminders

This dataset totals about 50GB; each image is quite tiny, but their sheer volume makes this problem challenging. At the very least, a parallel solution is going to be a must. In downloading this dataset, I set up both parallel and serial download processes. As you can imagine, downloading a single one of these small images was nearly instantaneous, but the serial process was on track to take **72 hours or more**, whereas the parallel process took 40 minutes.

If you run into problems, work with your teammates. Open up GitHub tickets, discuss the problems, find workarounds. If you still run into problems, query the Discord server. Ask questions! I'll be regularly checking Discord and interjecting where I can. Whatever you do, please don't turn into Lone Riders and try to solve the whole thing yourself.

Each image contains AT LEAST ONE face, but each line in the CSV files corresponds to one UNIQUE face. That means that the *faces* are your individual data points, not the images. Just a concept to keep in your mind–it's very possible that some images in GCP storage won't be used at all, whereas others will be used multiple times!

The label for FEMALE is 1, and the label for MALE is 0. Honestly you don't really need to know what the 0 and 1 stand for, just be sure you don't get the labels mixed up.
Who will set this year's top score? This project is *brand-spanking new for Spring 2021*, so whoever wins this semester will also set the all-time record for future classes to follow!

7 Submissions

All submissions will go to **AutoLab**, our brand-new autograder and leaderboard system.

One teammate should be designated for submitting on behalf of the team; they should submit to Autolab *using the team name* so I can easily match submissions with teams. If two people from the same team submit, I will pick one at random and grade that submission, so be sure you're communicating clearly.

When submitting, you will submit one file: **the `tar` archive containing the 3 label prediction files for datasets `Xa`, `Xb`, and `Xc`.** Each of the three text files should have a single prediction per line for each face. For example, if `Xa` had 5 faces, then the output file of label predictions might look like:

```
1
0
0
1
0
```

The tar archive can be named whatever you want (AutoLab renames it upon upload), but the three text files containing your predictions should be named:

```
ya.txt
yb.txt
yc.txt
```

(Note the `.txt` extensions!!!)

You must follow the naming convention EXACTLY, otherwise AutoLab won't see them in the autograder. It's totally fine if you want to submit a tar archive containing only one or two of the text files (especially as you are refining your model)–just know that any missing or incorrectly named files will result in the score for that column on the scoreboard being set to 0 (there's unfortunately no stateful mechanism in AutoLab where the previous scores can be pulled into the current round of autograding, so each new submission forgets the entirety of the previous one).

Another common pitfall is to get a 0 despite correctly naming the label files. This could be because your label has an *incorrect* number of predictions! If the `Xa` file has 10,001 faces / data points, then the `ya` file you submit should have 10,001 predictions in it! That's an early check that is made in the autograder, in fact: if the label file doesn't have the right number of entries, it automatically bails out of even attempting to grade that one (since there's no way to know how the labels are offset, or where the skipped data points were) and a 0 is kicked back to the leaderboard.

Once your model has made its predictions and you have one or some or all of the output files, `tar` them all together with the command:

```
> tar cvf p2.tar *.txt
```

That will create a file named `p2.tar` of all the TXT files in the current directory. Remember, you can name the tar archive whatever you want (AutoLab will automatically rename it), but the TXT files inside *must* be named correctly, otherwise they won't be used at all.

Your average classification accuracy for each of the three testing sub-datasets be tabulated and compared against the true labels (hidden on the server) and should appear on the leaderboard.

AutoLab is programmed to close submissions *promptly* at 11:59pm on March 18, so give yourself plenty of time!

8 Grading

Work with your team, stay in communication, take some risks, and you'll get an A. As discussed in the first lecture, I'm not conducting formal evaluations this semester due to ongoing COVID stress. As long as I don't hear complaints from teammates, you don't need to worry about grades.

Please note: good performance on this project requires a reasonable time investment, both in terms of functionality and algorithm performance, as well as on the software engineering side. Document your customizations you made to your code and how those tweaks resulted in an improved score on the final testing set. If your pipeline is multi-staged, clearly showing that a lot of thought went into the theoretical design–perhaps even using a current method (that is *properly* and *prominently* cited)–would be great.

Furthermore, please take the time to use solid software engineering techniques: your code style should be clean, logical, and well-documented, and you should provide a clear README with instructions to new users on installing and deploying your document classifier. Your `git commit` comments and GitHub tickets should provide a "paper trail" of documentation to show the route your team took in developing the software. Unit testing, continuous integration tools, consistent and thorough use of branches and merges, and adherence to coding style standards (e.g. Python's PEP8) are all excellent ways of engineering solid code.

9 Summary

In summary, these are the things I will look over when determining your team grade:

1. **Code**: Organization, structure, style, and clarity. Strategy and theory will also be a big component (what did you implement? was it implemented correctly? does the theory follow logically from the problem you're trying to solve?).
2. **Documentation**: This includes comments in the code, but also in the repository itself (GitHub wiki, `README` instructions). How easy would it be for someone to get up and running with your code, or to submit a bugfix?
3. **Accuracy**: Testing accuracy as submitted on AutoLab, and how far you are from the very top.
4. **Team**: Like the code, I'm looking for structure and organization, as well as a "paper trail" to get a feel for your team's working dynamics. How were project efforts divided among teammates? What do the git logs reveal about who was contributing to the project? How does this align with the `CONTRIBUTORS` content? Were code reviews conducted properly?
5. **Extras**: These include things like continuous integration tools, project websites, theoretical novelty (within the constraints of the project), unit tests with good coverage, among others. Use your imagination!