CS348 Milestone 1
Label.ai

Kevin Fen
Walker Hildebrand
Bilaal Hussain
Jacob Kim
Amirali Sharifzad

The Application

Purpose

label.ai is an application for companies and researchers in the field of machine learning classification.

The main reason for creating our platform, **label.ai**, is because machine learning and many other techniques in artificial intelligence require large amounts of data, and thus datasets are considered the new gold. The issue is that massive datasets with accurate labelling are extremely time consuming to create. However, there is no shortage of images to classify in this data-driven world. **label.ai** is looking to work with researchers and companies with massive amounts of automatically labelled data, in order to human-validate the data to find misclassifications or under-classifications. This process will refine the large datasets and consequently help improve the state of image classification.

Description

label.ai allows companies to validate their classification dataset by having a large number of people label image datasets. Essentially, companies will provide **label.ai** with labelled images, and we get individuals to verify whether each image actually contains the given labels. For example, we could have an image containing both a dog and a cat, however the company might think the image only contains a dog. Then, through labelers, **label.ai** will also label that this image contains a cat.

Users

label.ai has two types of users. The first type of user is the client. The client represents the party that wants to validate their classification data. Some examples of a client user could be classification machine learning research labs, technology companies that want to train and develop classification machine learning algorithms, such as google. Again, the main reason for a company to use **label.ai** is to validate their classification data.

The second type of user is the labeler user. The labelers represent the individuals who will be manually classifying the data. **label.ai** will crowdsource the labeler users from across the world, and match each labeler to our clients. Each labeler will have a trust score, ranging from 0 to 1, which is a measurement of how reliable and trustworthy the user is on classifying the data. This trust score of a labeler will be calculated using the labeler's history and accuracy of image classifying, increasing when their labels match what other trusted users input, and decreasing when other users disagree with them. This trust score ensures that malicious users or users that aren't as focused on classification do not have their votes given very significant weight. The

main reason for a labeler user to use **label.ai** is to make an income while working from home in a job that requires little skill or training.

User Interface

The user-facing site will be accessible to labellers. The users will interact with the site by…

The front end is implemented with react.js and here is a brief description of the workflow:

- The user will enter the website where they can login/signup.
- They will be prompted with an image and a classification.
- Three options will show:
    - 1. This image has <label>
    - 2. This image doesn't have <label>
    - 3. I don't know
- There will be a menu button available where users can view and edit their profile, and view their reputation/payscale.

Main Features

1. **Get One Classification Prompt - Image and Classification**
   To obtain user confidence towards an image-label pairing, labellers are given a list of images and their respective labels to classify. The following query supplies a list of all the images to label pairings, in which Label.ai's backend will randomly shuffle and send to the user.

   ```
   SELECT image.small_url, label.name
   FROM classification, image, label, (SELECT iid FROM image ORDER
   BY RANDOM() LIMIT 1) as a
   WHERE a.iid = image.iid AND classification.lid = label.lid AND
   image.iid = classification.iid
   ```

2. **Get All Classifications By Label**
   To make AI model testing easier for users, users can query a list of images that matches a specific label. The return list of images must be thoroughly user tested (i.e. a certain number of users have verified it's truth and the confidence factor is above a certain threshold).

   ```
   SELECT i.original_url
   FROM classification as c, image as i, label as l
   WHERE i.iid = c.iid
         AND c.lid = l.lid
         AND c.pre_classified = True
   ```

```
      AND l.name = `${req.label}`
      AND c.confidence > 0.8
      AND c.true_count + c.false_count >= 100;
```

3. **Find Mislabelled Images (SFW)**
   In the database, each picture to label classification is assigned a source of truth (valued True if the label corresponds to the picture and False otherwise). Moreover, each classification has a confidence level, which defines the degree in which labellers believe that the label corresponds to the image. Most of the time, the labellers' confidence should correspond to the predefined source of truth (e.g. high confidence that a label matches an image if the source of truth is True). However, there are some images whose labeller classifications are mismatched with the source of truth. To facilitate the test of AI models in more difficult image sets, Label.ai can supply these mismatched images with the following queries:

```
SELECT iid, l.name
FROM classification as c, label as l
WHERE confidence < 0.5
      AND pre_classified = TRUE
      AND c.lid = l.lid;


SELECT iid, l.name
FROM classification as c, label as l
WHERE confidence > 0.95
      AND pre_classified = TRUE
      AND c.lid = l.lid;
```

4. **Get labels that need more rankings**
   To ensure that all label/pairs are consistently tested, label.ai will occasionally query labels that need more verification from labellers. Labels that fit this category are labels which have an average confidence that is close to 0.5 (i.e. labellers are not sure if the label corresponds to the image) and labels that have a low total number of verifications from labellers. The following query will select these labels-image pairings, which will have a greater chance of appearing on future label-image pairing lists for labellers.

```
SELECT *
FROM
(
      SELECT
            lid,
            (SUM(ABS(confidence))/COUNT(*)) as avg_conf,
            SUM(true_count) as tc,
            SUM(false_count) as fc
      FROM classification
      GROUP BY lid
) AS a
WHERE (a.avg_conf > 0.45 AND a.avg_conf < 0.55) OR a.tc + a.fc
< 10;
```

5. **Custom Addition of Picture/Classifications/Labels**
   Users who want their datasets verified can also make requests to Label.ai for specific images to be added and the list of labels that they want the image to be verified against. These images will be stored in Label.ai's db and will randomly appear on the lists of image-label pairs that the labellers need to verify.

```
INSERT into image Values(10000,
'https://farm6.staticflickr.com/2300/2041178335_8fc60e09fe_o.jp
g','https://c5.staticflickr.com/3/2300/2041178335_da4967c386_z.
jpg?zz=1', 0);
INSERT INTO label VALUES(10000, 'window');
INSERT INTO classification VALUES(10001, 0,0,0, 't', 10000,
10000);
```

Additional Features

We plan on extending our database to be able to accommodate the following features in the future.

1. **Rewarding the best performing Labeller**
   Label.ai will incentivize labellers by rewarding the best labeller of each month with a bonus. The best labeller is a labeller with the highest trust factor amongst all the labellers that went through a certain number of images. Below is the query to find the best labeller.

```
-- Select labelers who have at least 400 classifications
CREATE VIEW QualifiedUsers AS SELECT * FROM member WHERE mid IN
(SELECT mid FROM submission GROUP BY mid HAVING COUNT(*) >
400);

-- Extract the best performing labeller
(SELECT name FROM QualifiedUsers) EXCEPT (SELECT u1.name FROM
QualifiedUsers as u1, QualifiedUsers as u2 WHERE u1.Trust <
u2.Trust);
```

2. **User Authentication**
   Users who want to use the data sets provided by Label.ai should be properly authenticated before being directed to a portal based on their roles (as a user who requests datasets or as a labeller). The sql statement below is a possible query that occurs during the authentication process.

   ```
   SELECT * FROM member WHERE username = 'testuser1' AND password
   = 'password123';
   ```

3. **Increasing Payscales with Increasing Trust**
   Since we are going to have different levels of 'trust' for our users, it would make sense that we pay them accordingly. We would want to implement a relation for payscales and a relation for payscale members that would automatically assign users to a better group when their trust exceeds a threshold, and demote them when it falls below another threshold.

Technical Stack

Our team will be using Postgres as the DBMS, Django (python) as the framework for the REST apis in the backend and react.js+axios for the frontend. The Django backend application will interact with the Postgres database.

Data Generation

We are using Google's Open Images V6 dataset. This dataset consists of thousands of hundreds of thousands of URLs to images and thousands of labels. Each image is assigned to one or more labels. We have implemented scripts to download the labelled test data from https://storage.googleapis.com/openimages/web/download.html.

For processing the data, we want to reduce the scale and complexity of the data and so we programmatically reduced the labels to a subset of 600 down from 19,000 and then purged the image urls, keeping only images labelled from our subset. We cleaned the data, removing whatever was not pertinent to our use using Python scripts with the pandas module.

We then took an even smaller subset of 10 labels, attempting to find a subset where images would have lots of overlap with labels. We then created a script for choosing a selection of the ~40 most labelled images and found that some of the images were under-classified (which is perfect for our application since we want to catch under-classification).

The approach we took started with a large dataset and reduced it to a much more manageable subset, so most of the work in this section will be used in the production of our large-scale dataset. These scripts produce three .csv files for the Classifications table, the Images table and the Labels table.

For the next steps, we will first create fake users and then try to use the confidence value found in Google's dataset to generate data for the submissions.

Database Design

Legend: * - primary key, * - foreign key

Tables

User:

- * uid: serial
- username: text
- password: text
- trust: float
- Name: text

Image:

- * iid: text
- original_url: text
- small_url: text
- rotation: Integer

‘Small_url’ is a URL to a smaller size of the image

Label:

- * lid: text
- name: text

Submission:

- * sid: serial
- * User uid: integer
- * Classification cid: integer
- correct_label: bool

‘correct_label’ is true if the user’s submission says the label is true and false otherwise

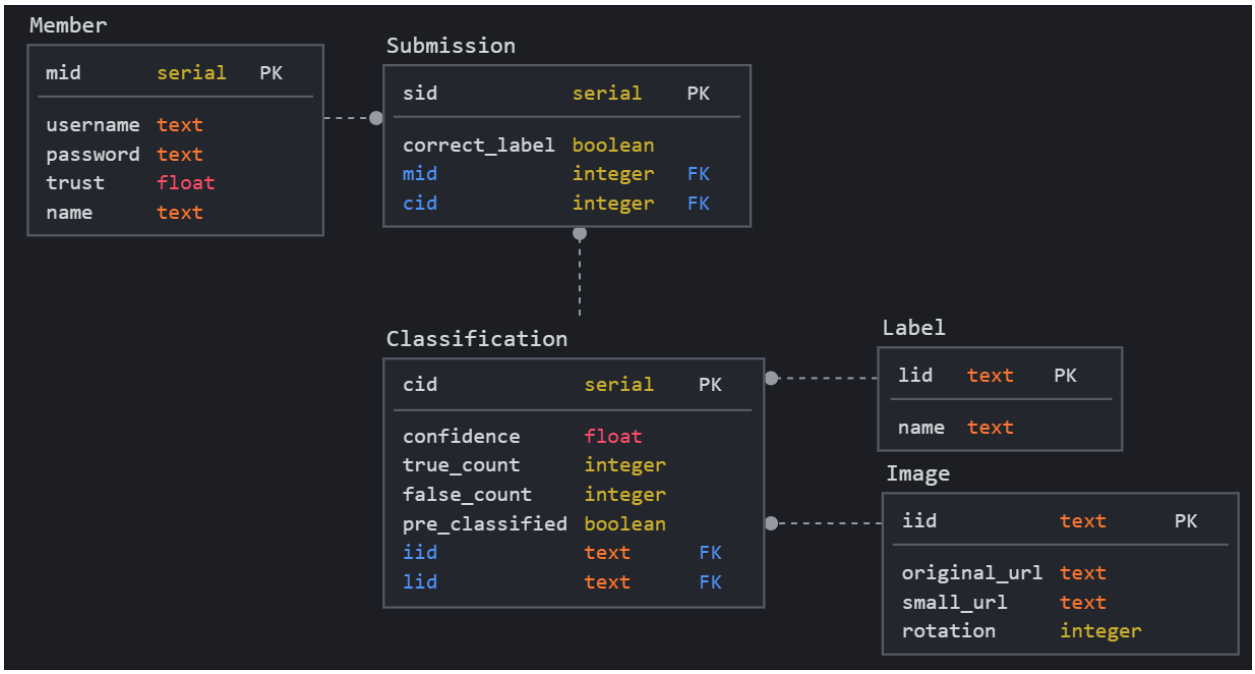
Classification:

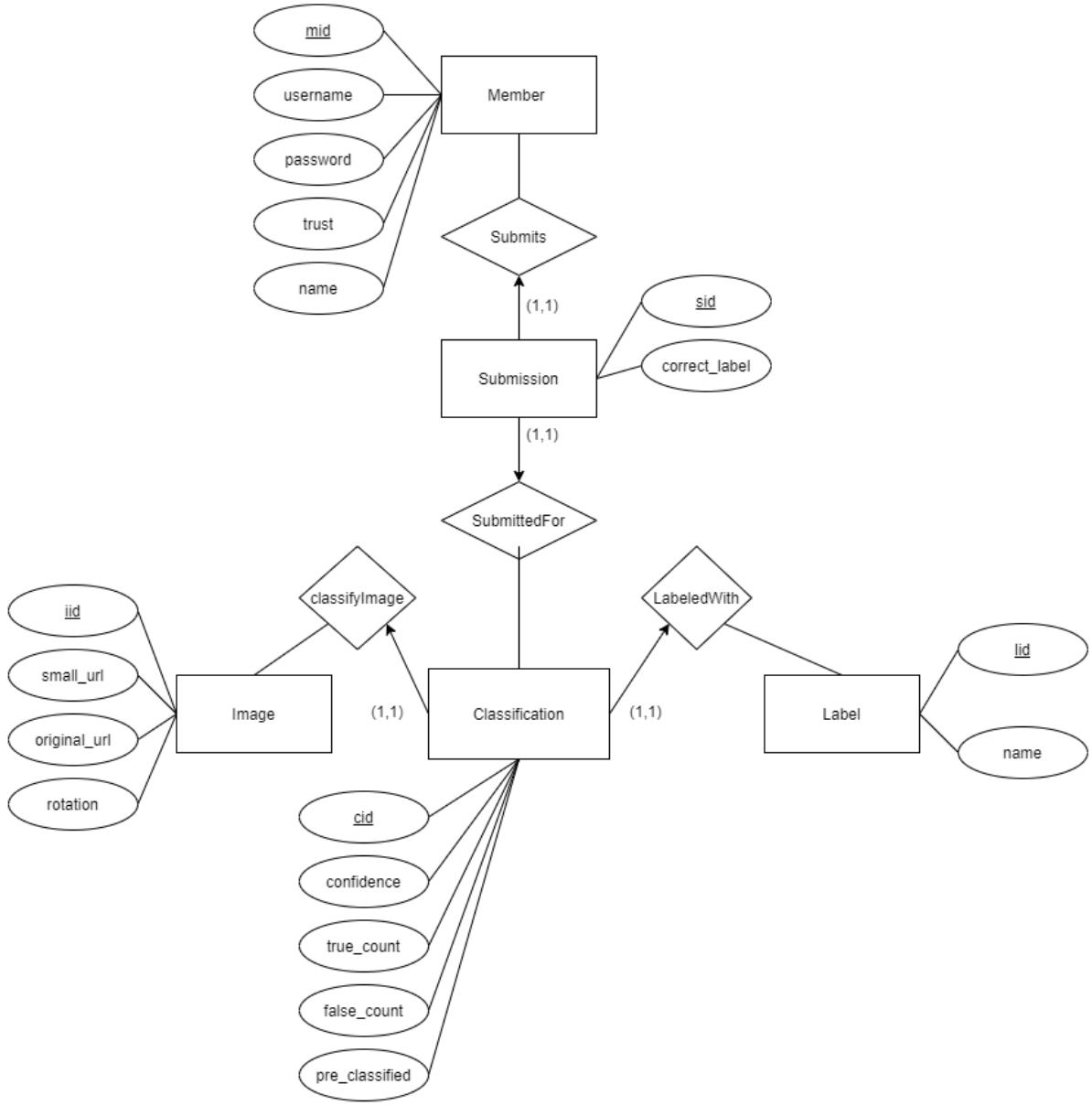
- * cid: serial
- * Image iid: text
- * Label lid: text
- confidence: float generated always as (true_count/(true_count+false_count)) stored
- true_count: Int
- false_count: Int
- pre_classified: Bool

Note for clarity: ‘pre_classified’ is true when a client gives a list of images and their classifications. Since we do not want to trust the classification entirely, we will assign other classifications to each image, but not mark them as ‘pre_classified’.

Assumptions:

- Member id (mid), image id (iid), label id (lid), classification id (cid), submission id (sid) should be all unique within their respective tables (hence primary keys of their respective tables)
- An image can be classified by 0 or more labels
- A label can relate to 0 or more images
- A label id and an image id pairing is an unique classification
- A member (User) can submit create multiple submission to a classification
- Newly inserted members have the same default, Trust value of 0
- Trust of a user is between 0 and 1
- Confidence of any classification is between -1 and 1
- Confidence of any classification is 0 if there are no submissions for the classification
- All foreign keys relate to existing rows in their corresponding table.





Member Contributions

Kevin Feng

- Created and drafted the ER diagram
- List assumptions that is used for the data model
- Assisted in drafting the application and required features
- Wrote up feature descriptions and revised SQL queries for correct outputs

Walker Hildebrand

- Came up with the early ideas of the project and database schema.
- Researched the dataset to use
- Wrote the scripts for processing the datasets and generated the test data.
- Compiled the report and contributed to various sections of the report

Bilaal Hussain

- Created relational diagram
- Created SQL table generation code
- Created SQL indexing on FKs to solve slow queries
- Helped write SFW queries for the core features
- Revised/fixed SFW queries to work with postgres syntax
- Developed the front-end UI in typescript
- Generated async mock API for getting classification problems, and posting classification solutions (e.g. getClassificationProblem())
- Fixed state update/infinite update loop issues in frontend

Jacob Kim

- Wrote the write up for the purpose and the description
- Came up with features and queries
- Tested queries for the features, validated the output
- Validated the relational diagram
- Contributed to the E/R diagram

Amirali Sharifzad

- Developed the backend in Django, implementing features for listing images and labels.
- Ensured database schema and SQL table creation statements are perfectly integrated with the Django ORM.
- Organized the git repository and created project milestones and tasks in the repository.
- Assisted with setting up the database, populating it, and designing the schema.

Citations

- Django Cookiecutter used to create boilerplate code for the backend: https://cookiecutter-django.readthedocs.io/en/latest/index.html
- *Open Images (V6 - released Feb 2020), Google, https://storage.googleapis.com/openimages/web/download.html*