CS145: Project 3 | ML Warmup (10 points)

Notes (read carefully!):

- Be sure you read the instructions on each cell and understand what it is doing before running
- Don't forget that if you can always re-download the starter notebook from the course website
- You may create new cells to use for testing, debugging, exploring, etc., and this is in fact enc for each question is **in its own cell** and **clearly indicated**.
- Colab will not warn you about how many bytes your SQL query will consume. **Be sure to chec queries here!**
- See the assignment handout for submission instructions.

Setting Up BigQuery and Dependencies

Run the cells below (shift + enter) to authenticate your project.

Note that you need to fill in the `project_id` variable with the Google Cloud project id you are usinc by going to https://console.cloud.google.com/cloud-resource-manager

```
# Run this cell to authenticate yourself to BigQuery
from google.colab import auth
auth.authenticate_user()
project_id = "cs145-fa19-254923"
```

```
# Initialize BiqQuery client
from google.cloud import bigquery
client = bigquery.Client(project=project_id)
```

Overview

This part of Project 3 is meant serve as a brief tutorial for Machine Learning with BigQuery, since y final portion of your explorations.

Don't worry if you've never studied Machine Learning before. This notebook will guide you through in the open-ended part of Project 3.

In the next two sections, we'll give you a bird's eye intro to machine learning and a primer on how B third and last section, you'll walk through an example of how to train and use a machine learning m

Section 1: Machine Learning in a Nutshell

Basic Machine Learning tasks can be framed in terms of **inputs** X, **target values** Y (sometimes ca data points (x_i, y_i)), and a function $h : X \to Y$ historically called the **hypothesis function** that m

Given these primitives, we can think of the canonical Machine Learning task as follows:

> Given that I've seen a ton of training data $(x_1, y_1), \ldots, (x_m, y_m)$, how can I come up with a *unseen* input value x_{m+1}, the value of $h(x_{m+1})$ is a good "prediction" y_{m+1} ?

Example 1: Three Point Shots

Say elements of X are the number of three point shots scored by a team in a basketball game, anc that team lost or won the game. Our training data could look like this:

> $T = \{(2, 0), (3, 0), (6, 0), (5, 0), (10, 0), (11, 0), (5, 1), (15, 1), (18, 0), (17, 1), (16, 1)$

In this case, we'd *train* a machine learning model on T to effectively generate an h that would give
of x, i.e., predict whether a game was won or not based on how many three pointers were scored c
that $h(1) \approx 0$, and $h(20) \approx 1$. Note that in this case, we'd like h to output not only a 0 or 1 but a
hence the approximate equalities.

Example 2: GitHub Revisited

For a richer example, let's say we are trying to predict how many *forks* (i.e., copies of the repo by Gi
Github repo will have at some point in time -- here Y will represent the number of forks of a repo. A
questions are usually not easily answerable with only one or two statistics of a Github repo. We us
together. What are the watch and star count of the repo? How many contributors does the repo hav
is the age of the repo in years?

The values in X *need not be single real numbers*, they can be lists of real numbers as well, and we
these feature lists.

Feature engineering is the most informal process by which we use domain-knowledge to extract nu
GitHub repo in this example), to provide them as training data to a machine learning model.

Here is how a simple feature engineering process for predicting the fork count of a GitHub repo ma

Simple feature engineering process

1. Using domain knowledge, you hypothesize that watch count, star count, number of commits, and age of th
count. You also toss in the average commit length of the repo because you know it has some non-trivial re
evidence from a certain project in your friend's database course.
2. You write some code in your favorite programming language (or SQL if using BigQuery!) to extract these fi
creating 1,000 tuples that look like this:

> $((45, 100, 200, 2.4, 127.65), 30), ((65, 302, 100, 1.2, 164.1), 132) \ldots$

3. You train your model on this data and evaluate it on another 100 repos **which your model has**
(the accuracy of your predictions) is not so good, you may attempt to improve the quality of y
done and have a decent model!
4. If you think the current features you thought about are not good enough, you can go back to

Once you have honed in on a good set features which you have trained with and evaluated, you car
be done on an additional test dataset that **your model has also not yet seen**.

Evaluating your Models

In Example 2, we said that the feature engineering process involves a key step in which you *evaluat*
function h) is doing. Usually this consists of:

1. Running your hypothesis function h on a set of inputs X which you have not already seen to get outputs h
2. Comparing how close your predicted values are to the ground-truth labels $y_{m+1}, \ldots, y_{m+k}$ using a reason

The "reasonable statistical metric" varies depending on the nature of your labels.

Here's a very brief overview of when you might want to use different metrics:

- **Accuracy** - when your classes are balanced (roughly same number of examples expected for
- **F1-score** - when your classes are very unbalanced (some categories are expected to have wa
- **Recall** - when you are more willing to have false positives than false negatives (e.g., predictin than miss an actual case)
- **Precision** - when you are more willing to have false negatives than false positives (e.g., predic spam in your inbox than have important emails go to spam)
- **RMS error** - when trying to predict a real value

There are many more ways to evaluate models, but deeper discussion is beyond the scope of this a

Data Splits

Overall, when doing machine learning, you should have three datasets:

- Training dataset - the bulk of your data, which you use to let your model learn a hypothesis fu
- Validation dataset - a small portion of your data, used to evaluate your trained model as you t hyperparameters.
- Test dataset - a small portion of your data, used to evaluate your **final model** when you are do You should not be running any models on this data to help choose relevant features.

Common splits for machine learning datasets are to take 80% of data for training, 10% for validatic are acceptable depending on how much data is available.

Types of Models

BigQuery supports three types of models: *linear regression, binary logistic regression, and multiclas*

- A *linear regression model* predicts a number, i.e., $Y = \mathbb{R}$.
- A *binary logistic regression model* makes a binary prediction by giving the confidence of an ev
- A *multiclass logistic regression model* is a generalization of the binary logistic regression mo sentence, from 1 (negative) to 5 (positive).

Example 1 above is a binary logistic regression model, and Example 2 is a linear regression model. your project.

If you have not already studied machine learning and are interested in digging into more details, rea
cover the basic topics discussed here and expand on them. However, the information in this noteb

Section 2: BigQuery and ML

In the previous section, we did not cover how the hypothesis function h is actually generated from
abstracts the details of this process away from us and instead exposes a nice SQL interface for M

Machine Learning in BigQuery consists of three steps: creating a model, evaluating the model, and

Creating a Model

This step consists of telling BigQuery that you want to create a model. You tell BigQuery what type
SQL to gather the features and ground-truth values for the model.

The create model statement could look like this:

```
# Don't run me!  My tables don't exist. I'm just here as an example.
%%bigquery --project $project_id

CREATE MODEL `my_awesome_model`
OPTIONS(model_type='logistic_reg') AS
SELECT
  IF(my_awesome_database.ground_truth IS NULL, 0, 1) AS label,
  IFNULL(my_awesome_database.feature1, "") AS feature1,
  my_awesome_database.feature2 AS feature2,
  my_awesome_database.feature3 AS feature3,
  my_awesome_database.feature2 * my_awesome_database.feature3 AS feature4
FROM
  `my_awesome_database`
WHERE
  my_awesome_database.date BETWEEN 2010 AND 2015
```

One thing to note: `CREATE MODEL` will fail if the model with that name has already been created. If
you'll want to use `CREATE OR REPLACE MODEL` as the first line instead.

See this page for documentation: https://cloud.google.com/bigquery/docs/reference/standard-sq

Evaluating the Model

Once you've created your model, BigQuery has already trained it for you -- you already have a h at y
h by asking BigQuery to predict the Y values of **new data unseen by the model** and compare them

To evaluate a model you'd do something like this:

```
# Don't run me!  My tables don't exist. I'm just here as an example.
%%bigquery --project $project_id

SELECT
  *
FROM
  ML.EVALUATE(MODEL `my_awesome_model`, (
SELECT
  IF(my_awesome_database.ground_truth IS NULL, 0, 1) AS label,
  IFNULL(my_awesome_database.feature1, "") AS feature1,
  my_awesome_database.feature2 AS feature2,
  my_awesome_database.feature3 AS feature3,
  my_awesome_database.feature2 * my_awesome_database.feature3 AS feature4
FROM
  `my_awesome_database`
WHERE
  my_awesome_database.date BETWEEN 2016 AND 2017))
```

Note that we are evaluating on data between 2016 and 2017, even though we trained on data betw
did not do this, we would be "cheating" since the model has already seen a training value correspo
the model was a simple lookup table, it would get 100% accuracy on everything it's already seen tri
amount of data for training than for evaluating or testing.

See this page for documentation: https://cloud.google.com/bigquery/docs/reference/standard-sq
`ML.EVALUATE` function is one of three functions you can use to evaluate your model, depending on

Exercising the Model

If your model achieves good evaluation metrics (see this section of the BigQuery ML tutorial for da
evaluation metrics are), you can now utilize your model to predict values. Again, this should be don

Assuming you have a trained model, you can predict values like this:

```
# Don't run me!  My tables don't exist. I'm just here as an example.
%%bigquery --project $project_id

SELECT
  my_awesome_database.key,
  predicted_label
```

```
FROM
  ML.PREDICT(MODEL `my_awesome_model`, (
SELECT
  IFNULL(my_awesome_database.feature1, "") AS feature1,
  my_awesome_database.feature2 AS feature2,
  my_awesome_database.feature3 AS feature3,
  my_awesome_database.feature2 * my_awesome_database.feature3 AS feature4
FROM
  `my_awesome_database`
WHERE
  my_awesome_database.date BETWEEN 2018.01 AND 2018.02))
```

See this page for documentation: https://cloud.google.com/bigquery/docs/reference/standard-sq

For more details and an end-to-end example in BigQuery, read the following article: https://cloud.go analyst-start.

Section 3: Now it's Your Turn!

Let's now dive into an exercise using BigQuery and ML! This is a fairly simple warm-up problem to l with BQML. You'll get to dive into much more depth with your open-ended project! You'll be going th previous section on your own.

For this problem, we're going to be working with the Austin bikeshare dataset available in BigQuery yourself with the data we have at hand.

Notice we have various pieces of information about each trip - for example, the stations where the corresponding latitude/longitude, the date of the ride, the subscriber type, and the duration of the t

Our goal in this exercise will be the following:

> *Given attributes about a ride, can we predict whether a bike ride will be a "quick" ride? Let's defin less than 15 minutes.*

Note this is a *binary logistic regression task*, or classification task, where, given attributes about a r 15 minutes); 0 = not quick (>= 15 minutes).

Once we've trained our model, we can then use it to help predict on unlabeled data. In particular, we bike rides have a different start/end station, but have a duration of 0 minutes (likely missing data).

Let's dive in!

Step 1: Look at the data (1 point)

In any ML task, it's important to first explore the data. Investigating correlations between attributes determine which attributes may be useful as training features, and will be important for your final p labels, you want the prediction to give you a better understanding of the distribution of your data (s For this exercise, we'll dig into the latter.

a) What percentage of rides are "quick"? Recall that we have: quick ride: < 15 minutes; not quick: duration of 0 minutes.

Hint: COUNTIF may be helpful.

```
%%bigquery --project $project_id

SELECT ROUND(COUNTIF(duration_minutes < 15)/COUNT(*)*100, 2) AS quick_ride_percenta
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
WHERE duration_minutes != 0
```

b) What percentage of rides have a different start/end station, but have a value of 0 for their dura that returns the count in one column and the percentage in another. The denominator for the perc duration.

Hint: COUNTIF may be helpful.

```
%%bigquery --project $project_id

SELECT COUNT(a.trip_id) AS count, ROUND(COUNT(a.trip_id)*100/(SELECT COUNT(b.trip_i
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips` a
WHERE a.start_station_id != a.end_station_id
AND a.duration_minutes = 0
```

Step 2: Create a dataset to store the model

When you create and train a model, BigQuery will store the model in a dataset. Before training, you' Note that you only need to do this step once. If you later update your model, it can replace the exis

You can also do this step in the UI (see 'create your dataset': https://cloud.google.com/bigquery/d

Let's call our dataset `bqml_bikeshare`. After either running the cell below, or creating the dataset dataset name appear in the left column of the UI.

```
# Run this cell to create a dataset to store your model, or create in the UI

model_dataset_name = 'bqml_bikeshare'

dataset = bigquery.Dataset(client.dataset(model_dataset_name))
dataset.location = 'US'
client.create_dataset(dataset)
```

Step 3: Extract training data from BigQuery (2 points)

Write a SQL query that extracts training data from the dataset. These are features that you want t not need to do feature engineering - you can simply pull raw features from the tables that you thi

Your query should return a column called `label` with the target label value (our "Y" value), and add to use (our "X" values). Note:

- recall: `label` value is 1 for quick rides (< 15 minutes), and 0 otherwise (>= 15 minutes)
- duration_minutes cannot be a training feature - we're trying to predict (a boolean version of) t
- filter out any rides with a duration of 0 minutes

Display the first 10 rows of the table returned by your query.

```
%%bigquery --project $project_id

SELECT IF(duration_minutes < 15, 1, 0) AS label, subscriber_type, start_time, start
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
WHERE duration_minutes != 0
LIMIT 10
```

Step 4: Train a simple model (1 point)

First, an important note: it's important to have separate datasets to train, evaluate, and finally test data:

1. **Training set**: used to train a model.
   - we'll train on rides before 2017 (start_time < '2017-01-01'), with duration time > 0
2. **Evaluation set**: used to evaluate model after training. This should not be data used during training. It can b performance of different models.
   - we'll evaluate on the next 5 months (start_time between '2017-01-01' and '2017-06-01'),
3. **Test set**: *should only be used once at the end of your entire training process* to say how your model does or training or eval data. Using the test set to tune your model is bad, since it means you are starting to overfit good on a certain dataset at the possible expense of it doing poorly on new data) to that test set as well.
   - we'll test on the 5 months after that (start_time between '2017-06-01' and '2017-11-01').

Note that for all these datasets, we'll filter out rides with duration time = 0. For the purposes of this data.

Now, let's go ahead and train a simple model. **Create a model, using the query you wrote above to truth labels to use.** Remember that we're training only on rides before 2017 (start_time < '2017-01-

Note: it may take a few minutes to run the query. Also, you may get the error `Table has no schem` because notebook cells try to print out the table returned from a SQL query, but the query to create so the notebook complains. The model is still trained successfully though. You may ignore this, an to clear the error message.

```
%%bigquery --project $project_id

CREATE OR REPLACE MODEL `bqml_bikeshare.bikeshare_model`
OPTIONS(model_type='logistic_reg') AS
SELECT
  IF(duration_minutes < 15, 1, 0) AS label,
  subscriber_type,
  start_time,
  start_station_id,
  end_station_id
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
WHERE duration_minutes != 0
AND DATE(start_time) < '2017-01-01'
```

You can get training statistics on your model by running the following cell:

```
%%bigquery --project $project_id

# Run cell to view training stats

SELECT
  *
FROM
  ML.TRAINING_INFO(MODEL `bqml_bikeshare.bikeshare_model`)
```

Step 5: Evaluate (1 point)

Evaluate your model on unseen evaluation data.

Recall for our evaluation set, we're using the 5 months following what we trained on (use: start_tim
with duration time > 0.

```
%%bigquery --project $project_id

SELECT
  *
FROM
  ML.EVALUATE(MODEL `bqml_bikeshare.bikeshare_model`, (
SELECT
  IF(duration_minutes < 15, 1, 0) AS label,
  subscriber_type,
  start_time,
  start_station_id,
  end_station_id
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
WHERE duration_minutes != 0
AND DATE(start_time) BETWEEN '2017-01-01' AND '2017-06-01'))
```

Step 6: Improving our model (3 points)

In general, we can't just throw raw data into the model and expect it to work: in practice, you'll iterat
training/re-evaluating your model. Let's try the following: add engineered features -> re-train model

a) Let's add an engineered feature! You suspect that there is a relationship between the distance
whether it will be a "quick" ride. Let's add the distance between the start station and the end stati

Extend your query from step 3 to also have a feature for the euclidean distance between the start a

You may find the following useful:

- Example from Dr. Lakshmanan's invited talk

- `ST_GeogPoint(longitude, latitude)` - creates geography point from longitude, latitude va
- `ST_DISTANCE(start_pt, end_pt)` - computes distance between 2 geographic points (more

You are welcome, but not required, to experiment with other engineered features as well.

Display the first 10 rows of the table returned by your query.

```
%%bigquery --project $project_id

SELECT
  IF(duration_minutes < 15, 1, 0) AS label,
  subscriber_type,
  start_time,
  start_station_id,
  end_station_id,
  ST_Distance(ST_GeogPoint(pick_longitude, pick_latitude), ST_GeogPoint(drop_longit
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
JOIN (SELECT station_id, latitude AS pick_latitude, longitude AS pick_longitude
      FROM `bigquery-public-data.austin_bikeshare.bikeshare_stations`) s1
ON start_station_id = s1.station_id
JOIN (SELECT station_id, latitude AS drop_latitude, longitude AS drop_longitude
      FROM `bigquery-public-data.austin_bikeshare.bikeshare_stations`) s2
ON end_station_id = s2.station_id
WHERE duration_minutes != 0
LIMIT 10
```

**b) Let's train our model again (using the same training set as before) with the added features. You
new one with a different name. (1 point)**

Note: it may take a few minutes to run the query. Also, you may again get the error `Table has no` model is still trained successfully though. You may ignore this, and can click the (X) in the top left c

```
%%bigquery --project $project_id

# YOUR QUERY HERE

CREATE OR REPLACE MODEL `bqml_bikeshare.bikeshare_model_v2` -- we'll call our model
OPTIONS(model_type='logistic_reg') AS
SELECT
  IF(duration_minutes < 15, 1, 0) AS label,
  subscriber_type,
  start_time,
  start_station_id,
  end_station_id,
  ST_Distance(ST_GeogPoint(pick_longitude, pick_latitude), ST_GeogPoint(drop_longit
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
JOIN (SELECT station_id, latitude AS pick_latitude, longitude AS pick_longitude
      FROM `bigquery-public-data.austin_bikeshare.bikeshare_stations`) s1
ON start_station_id = s1.station_id
JOIN (SELECT station_id, latitude AS drop_latitude, longitude AS drop_longitude
      FROM `bigquery-public-data.austin_bikeshare.bikeshare_stations`) s2
ON end_station_id = s2.station_id
WHERE duration_minutes != 0
AND DATE(start_time) < '2017-01-01'
```

Let's get our training stats again:

```
%%bigquery --project $project_id

# Run cell to view training stats

SELECT
  *
FROM
```

```
FROM
  ML.TRAINING_INFO(MODEL `bqml_bikeshare.bikeshare_model_v2`)
```

You'll should hopefully find that the loss is a bit lower (better) than before, on both on the training c
set (it withholds some data you passed in as training data for reporting eval stats).

c) Now let's evaluate our re-trained model on our evaluation set. You can use a similar evaluation features (note: you may need to change the model name in the query if your new model has a diff

```
%%bigquery --project $project_id

SELECT
  *
FROM
  ML.EVALUATE(MODEL `bqml_bikeshare.bikeshare_model_v2`, (
    SELECT
      IF(duration_minutes < 15, 1, 0) AS label,
      subscriber_type,
      start_time,
      start_station_id,
      end_station_id,
      ST_Distance(ST_GeogPoint(pick_longitude, pick_latitude), ST_GeogPoint(drop_lc
    FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
    JOIN (SELECT station_id, latitude AS pick_latitude, longitude AS pick_longitude
          FROM `bigquery-public-data.austin_bikeshare.bikeshare_stations`) s1
    ON start_station_id = s1.station_id
    JOIN (SELECT station_id, latitude AS drop_latitude, longitude AS drop_longitude
          FROM `bigquery-public-data.austin_bikeshare.bikeshare_stations`) s2
    ON end_station_id = s2.station_id
    WHERE duration_minutes != 0
    AND DATE(start_time) BETWEEN '2017-01-01' AND '2017-06-01'))
```

Step 7: Evaluate final model on test set (1 point)

Once you're done training your model (in practice, you'll likely iterate on updating your model, retrai
the evaluation set several times), you'll evaluate your final model on a test set. The test set consist
before, neither during training nor during evaluation.

Again, the test set should **only be used once at the end of your entire training process**, to see how
only run the cell below once you are finished modifying your features.

Recall that for our test set, we're using the 5 months after our evaluation set (rides with start_time
duration time > 0).

Evaluate your model once on this test set. The query is almost identical to the previous one, exce

```
%%bigquery --project $project_id

SELECT
  *
FROM
  ML.EVALUATE(MODEL `bqml_bikeshare.bikeshare_model_v2`, (
    SELECT
      IF(duration_minutes < 15, 1, 0) AS label,
      subscriber_type,
      start_time,
      start_station_id,
      end_station_id,
      ST_Distance(ST_GeogPoint(pick_longitude, pick_latitude), ST_GeogPoint(drop_lo
    FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
    JOIN (SELECT station_id, latitude AS pick_latitude, longitude AS pick_longitude
          FROM `bigquery-public-data.austin_bikeshare.bikeshare_stations`) s1
    ON start_station_id = s1.station_id
    JOIN (SELECT station_id, latitude AS drop_latitude, longitude AS drop_longitude
          FROM `bigquery-public-data.austin_bikeshare.bikeshare_stations`) s2
    ON end_station_id = s2.station_id
    WHERE duration_minutes != 0
    AND DATE(start_time) BETWEEN '2017-06-01' AND '2017-11-01'))
```

Step 8: Use the trained model to predict (1 point)

Once you've trained your model, you can use it to make predictions! Let's try to use it to fill in some

**Now, let's go ahead and predict on rides that had a duration time of 0 minutes, but had different s
these were quick rides?**

Notice that these samples were never used during training/evaluation/testing, since we filtered ou

Display the features used for prediction and the predicted label for 10 examples. The predicted lab

```
%%bigquery --project $project_id

SELECT
  subscriber_type,
  start_time,
  start_station_id,
  end_station_id,
  distance,
  predicted_label
FROM
  ML.PREDICT(MODEL `bqml_bikeshare.bikeshare_model_v2`, (
    SELECT
      subscriber_type,
      start_time,
      start_station_id,
      end_station_id,
      ST_Distance(ST_GeogPoint(pick_longitude, pick_latitude), ST_GeogPoint(drop_lo
    FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
    JOIN (SELECT station_id, latitude AS pick_latitude, longitude AS pick_longitude
          FROM `bigquery-public-data.austin_bikeshare.bikeshare_stations`) s1
    ON start_station_id = s1.station_id
    JOIN (SELECT station_id, latitude AS drop_latitude, longitude AS drop_longitude
          FROM `bigquery-public-data.austin_bikeshare.bikeshare_stations`) s2
    ON end_station_id = s2.station_id
    WHERE duration_minutes = 0
    AND start_station_id != end_station_id
  ))
LIMIT 10
```