[Website (https://www.coleridgeinitiative.org)](https://www.coleridgeinitiative.org)

Ghani, Rayid, Frauke Kreuter, Julia Lane, Adrianne Bradford, Alex Engler, Nicolas Guetta Jeanrenaud, Graham Henke, Daniela Hochfellner, Clayton Hunter, Brian Kim, Avishek Kumar, Jonathan Morgan, and Benjamin Feder.

source to be updated when notebook added to GitHub

Table of Contents

JupyterLab contains a dynamic Table of Contents that can be accessed by clicking the last of the six icons on the left-hand sidebar.

Dataset Preparation

In this notebook, we will walk through preparing our data for machine learning. In practice, the data preparation should take some time as you will need to think deeply about the question at the heart of your project.

The Machine Learning Process

The Machine Learning Process is as follows:

- **Understand the problem and goal.** *This sounds obvious but is often nontrivial.* Problems typically start as vague descriptions of a goal - improving health outcomes, increasing graduation rates, understanding the effect of a variable *X* on an outcome *Y*, etc. It is really important to work with people who understand the domain being studied to dig deeper and define the problem more concretely. What is the analytical formulation of the metric that you are trying to optimize?
- **Formulate it as a machine learning problem.** Is it a classification problem or a regression problem? Is the goal to build a model that generates a ranked list prioritized by risk, or is it to detect anomalies as new data come in? Knowing what kinds of tasks machine learning can solve will allow you to map the problem you are working on to one or more machine learning settings and give you access to a suite of methods.
- **Data exploration and preparation.** Next, you need to carefully explore the data you have. What additional data do you need or have access to? What variable will you use to match records for integrating different data sources? What variables exist in the data set? Are they continuous or categorical? What about missing values? Can you use the variables in their original form, or do you need to alter them in some way?
- **Feature engineering.** In machine learning language, what you might know as independent variables or predictors or factors or covariates are called "features." Creating good features is probably the most important step in the machine learning process. This involves doing transformations, creating interaction terms, or aggregating over data points or over time and space.
- **Method selection.** Having formulated the problem and created your features, you now have a suite of methods to choose from. It would be great if there were a single method that always worked best for a specific type of problem. Typically, in machine learning, you take a variety of methods and try them, empirically validating which one is the best approach to your problem.
- **Evaluation.** As you build a large number of possible models, you need a way choose the best among them. We'll cover methodology to validate models on historical data and discuss a variety of evaluation metrics. The next step is to validate using a field trial or experiment.
- **Deployment.** Once you have selected the best model and validated it using historical data as well as a field trial, you are ready to put the model into practice. You still have to keep in mind that new data will be coming in, and the model might change over time.

Here, to reiterate, we will work through all the steps we can accomplish querying directly from our Athena database, and then in the following notebook, we will bring our table we created in this notebook into python and complete the machine learning process.

Problem Formulation

First, you need to turn something into a real objective function. What do you care about? Do you have data on that thing? What action can you take based on your findings? Do you risk introducing any bias based on the way you model something?

Four Main Types of ML Tasks for Policy Problems

- **Description**: How can we identify and respond to the most urgent online government petitions? (https://dssg.uchicago.edu/project/improving-government-response-to-citizen-requests-online/)
- **Prediction**: Which students will struggle academically by third grade? (https://dssg.uchicago.edu/project/predicting-students-that-will-struggle-academically-by-third-grade/)
- **Detection**: Which police officers are likely to have an adverse interaction with the public? (https://dssg.uchicago.edu/project/expanding-our-early-intervention-system-for-adverse-police-interactions/)
- **Behavior Change**: How can we prevent juveniles from interacting with the criminal justice system? (https://dssg.uchicago.edu/project/preventing-juvenile-interactions-with-the-criminal-justice-system/)

Our Machine Learning Problem

> Out of low-income households, can we predict which ones did not purchase a 100% whole wheat product in a year's time? If so, what are the most important household features?

This is an example of a *binary prediction classification problem*.

Note the time windows are completely arbitrary. You could use an outcome window of 5, 3, 1 years or 1 day. The outcome window will depend on how often you receive new data, how accurate your predictions are for a given time period, or on what time-scale you can use the output of the data.

> By low-income households, we're referring to only those who are WIC participants or WIC-eligible.

Access the Data

As always, we will bring in the python libraries that we need to use, as well as set up our connection to the database.

In [ ]:
```
# pandas-related imports
import pandas as pd

# database interaction imports
from pyathenajdbc import connect
```

In [ ]:
```
conn = connect(s3_staging_dir = 's3://usda-iri-2019-queryresults/',
               region_name = 'us-gov-west-1',
               LogLevel = '0',
               workgroup = 'workgroup-iri_usda')
```

Define our Cohort

Since the machine learning problem focuses on finding the features most important in predicting if a low-income household will not purchase 100% whole wheat product at least once in a year, we will focus just on households that were either WIC-eligible or participants in a given year. Here, we will train our models on data from low-income households in 2014 and their presence of 100% whole wheat purchases in 2015 and test on low-income households in 2015 buying 100% whole wheat product(s) in 2016.

Let's first see how many of these households we will have in our testing and training datasets.

> We already created our 2014 and 2015 household tables, `init_train` and `init_test` in the `iri_usda_2019_db` database, by changing the years from the code used to create `project_q2_cohort` in the Second Data Exploration (02_02_Data_Exploration_Popular_Foods.ipynb) notebook. We also subsetted the `panid` to only include households who had static purchasing data (`projection61k` > 0) the year we're predicting on and the year prior (i.e. 2014 and 2015 for our training set). `init_train` and `init_test` contain the exact same variables as the `demo_all` table in the `iri_usda` Athena database.

In [ ]:
```
# get count for 2014 low-income households
qry = '''
select count(*) as num_2014
from iri_usda_2019_db.init_train
'''

pd.read_sql(qry, conn)
```

In [ ]:
```
# get count for 2015 low-income households
qry = '''
select count(*) as num_2015
from iri_usda_2019_db.init_test
'''

pd.read_sql(qry, conn)
```

Create Foundation for Training and Testing Datasets

Now that we have defined our cohorts for our testing and training datasets, we need to combine our available datasets so that each low-income household is a row containing demographic data from the previous year, if they purchased a 100% whole wheat proudct in the following calendar year, and aggregate purchasing data from the prior year. For the purchasing data, we want to aggregate the amount the household spent and their total amount of trips.

To do this, we will first find all households that purchased any 100% whole wheat product in our given prediction years (2015 and 2016), and then we will join it to our low-income household datasets from the previous year. Because we will be relying on the table of households who purchased any 100% whole wheat product to create our desired table in Athena, we will save it as a permanent table. Then, we will join this table with our low-income cohort and one containing aggregate purchasing data for the prior year for these households.

> Note: It is possible to do this process in one step. However, for your understanding and ease in reproducibility, we broke it down into multiple steps to avoid a larger subquerying process.

In [ ]:
```
# see existing table list
table_list = pd.read_sql('show tables IN iri_usda_2019_db;', conn)
print(table_list)

# get a series of tab_name values
s = pd.Series(list(table_list['tab_name']))
```

In [ ]:
```
# create table to find households that bought 100% whole wheat products
 in 2015 or 2016
if('ml_aggregate' not in s.unique()):
    print('creating table')
    qry = '''
    create table iri_usda_2019_db.ml_aggregate
    with(
    format = 'Parquet',
    parquet_compression = 'SNAPPY'
    )
    as
    select t.panid, t.year, sum(t.dollarspaid) as dollarspaid
    from iri_usda.pd_pos_all p, iri_usda.trip_all t
    where p.upc = t.upc and (t.year = '2016' or t.year = '2015') and p.u
pcdesc like '%100% WHOLE WHEAT%' and
    p.year = t.year
    group by t.panid, t.year
    ;
    '''
    with conn.cursor() as cursor:
        cursor.execute(qry)
else:
    print('table already exists')
```

Checkpoint 1: What question are we asking?

Above, we are creating an aggregated table of all purchases in which a product with "100% Whole Wheat" in the description was purchased. However, we might want to broaden the definition to include other whole grains. For example, you might want to include corn tortillas or oatmeal, to make sure you're catching as many of the different types of whole grains that people may purchase. How would you include these other whole grain items in your table?

Creating Train and Test Sets

Now that we've created the aggregated table for households that purchased any 100% whole wheat products, we can combine that with `init_train` and `init_test` to get demographic data and define our label. Let's first take a look at the `ml_aggregate` table to see how it looks. Remember, this is a table that contains each household that purchased a 100% whole wheat product along with the total dollars paid in that year for 100% whole wheat products.

In [ ]:
```
# view ml_aggregate
qry = '''
select *
from iri_usda_2019_db.ml_aggregate
limit 10
'''

pd.read_sql(qry, conn)
```

We can now join `ml_aggregate` with `init_train` and `init_test` to grab the demographic data. Since we would like to match households that purchased 100% whole wheat products in either 2015 or 2016 to low-income households in `init_train` and `init_test` (those with no 100% whole wheat product purchases the following year will have NAs), we will left join `ml_aggregate` to `init_train` and `init_test` . Also, we will add our dependent variable, `label` , using a `case when` statement that is `yes` when the household purchased 100% whole wheat products in the following calendar year.

In [ ]:

```
# match ml_aggregate with demographic data for just our training cohort
# left join so that we maintain all low-income households who didn't buy
any 100% whole wheat products
if('ml_combined_train' not in s.unique()):
    qry = '''
    create table iri_usda_2019_db.ml_combined_train
    with(
    format = 'Parquet',
    parquet_compression = 'SNAPPY'
    )
    as
    select c.panid, c.hhsize, c.hhinc, c.race, c.hisp, c.ac, c.fed, c.fe
mp, c.med,
    c.memp, c.mocc, c.marital, c.rentown, c.cats, c.dogs, c.hhtype, c.re
gion, c.wic_june, c.snap_june,
    c.projection61k,
    case when a.dollarspaid > 0 then 0
        else 1
            end as label
    from iri_usda_2019_db.init_train c
    left join (
        select *
        from iri_usda_2019_db.ml_aggregate a
        where year = '2015'
        ) a
    on c.panid = a.panid
    '''
    with conn.cursor() as cursor:
        cursor.execute(qry)
else:
    print('table already exists')
```

In [ ]:
```
# match ml_aggregate with demographic data for just our testing cohort
# left join so that we maintain all low-income households who didn't buy
any 100% whole wheat products
if('ml_combined_test' not in s.unique()):
    qry = '''
    create table iri_usda_2019_db.ml_combined_test
    with(
    format = 'Parquet',
    parquet_compression = 'SNAPPY'
    )
    as
    select c.panid, c.hhsize, c.hhinc, c.race, c.hisp, c.ac, c.fed, c.fe
mp, c.med,
    c.memp, c.mocc, c.marital, c.rentown, c.cats, c.dogs, c.hhtype, c.re
gion, c.wic_june, c.snap_june,
    c.projection61k,
    case when a.dollarspaid > 0 then 0
        else 1
            end as label
    from iri_usda_2019_db.init_test c
    left join (
        select *
        from iri_usda_2019_db.ml_aggregate a
        where year = '2016'
        ) a
    on c.panid = a.panid
    '''
    with conn.cursor() as cursor:
        cursor.execute(qry)
else:
    print('table already exists')
```

In [ ]:
```
# verify ml_combined_train is what we want
qry = '''
select *
from iri_usda_2019_db.ml_combined_train
limit 5
'''

pd.read_sql(qry, conn)
```

In [ ]:
```
# verify ml_combined_test is what we want
qry = '''
select *
from iri_usda_2019_db.ml_combined_test
limit 5
'''

pd.read_sql(qry, conn)
```

Finally, we want to add in the amount spent and number of trips in 2014 or 2015 for these households in the IRI database. We will first confirm that we can find the amount spent and number of trips a household took according to the `trip_all` table in either 2014 or 2015 for households in `ml_combined_train` and `ml_combined_test` .

> Recall that to calculate the amount spent, you can subtract `coupon` from `dollarspaid` . The number of trips per household is the distinct value of `tripnumber` and `purdate` .

In [ ]:
```
# find aggregate purchasing information by households in 2014 and 2015
qry = '''
select panid, year, round(sum(dollarspaid) - sum(coupon),2) as total,
    count(distinct(purdate, tripnumber)) as num_trips
from iri_usda.trip_all
where year in ('2014', '2015') and panid in
    (
    select distinct panid
    from iri_usda_2019_db.ml_combined
    )
group by year, panid
limit 5
'''

pd.read_sql(qry, conn)
```

Now that we can find aggregate purchasing data in 2014 and 2015 for households in `ml_combined_train` and `ml_combined_test` , we can perform another left join using this query. We just need to make sure that we are matching based on `panid` , and making sure that we are selecting the purchasing data from the year prior for each row in `ml_combined_train` and `ml_combined_test` .

This will be our final table we create before moving onto the Machine Learning (04_02_Machine_Learning.ipynb) notebook.

In [ ]:
```
if('ml_model_train' not in s.unique()):
    qry = '''
    create table iri_usda_2019_db.ml_model_train
        with(
        format = 'Parquet',
        parquet_compression = 'SNAPPY'
        )
        as
    select a.*, b.total, b.num_trips
    from iri_usda_2019_db.ml_combined_train a
    left join
        (select panid, round(sum(dollarspaid) - sum(coupon),2) as total,
            count(distinct(purdate, tripnumber)) as num_trips
        from iri_usda.trip_all
        where year in ('2014') and panid in
            (
            select distinct panid
            from iri_usda_2019_db.ml_combined_train
            )
        group by panid
        ) b
    on a.panid = b.panid
    '''
    with conn.cursor() as cursor:
        cursor.execute(qry)
else:
    print('table already exists')
```

In [ ]:
```
if('ml_model_test' not in s.unique()):
    qry = '''
    create table iri_usda_2019_db.ml_model_test
        with(
        format = 'Parquet',
        parquet_compression = 'SNAPPY'
        )
        as
    select a.*, b.total, b.num_trips
    from iri_usda_2019_db.ml_combined_test a
    left join
        (select panid, round(sum(dollarspaid) - sum(coupon),2) as total,
            count(distinct(purdate, tripnumber)) as num_trips
        from iri_usda.trip_all
        where year in ('2015') and panid in
            (
            select distinct panid
            from iri_usda_2019_db.ml_combined_test
            )
        group by panid
        ) b
    on a.panid = b.panid
    '''
    with conn.cursor() as cursor:
        cursor.execute(qry)
else:
    print('table already exists')
```

```
In [ ]: # verify ml_model_train is what we want
        qry = '''
        select *
        from iri_usda_2019_db.ml_model_train
        limit 5
        '''

        pd.read_sql(qry, conn)
```

```
In [ ]: # verify ml_model_test is what we want
        qry = '''
        select *
        from iri_usda_2019_db.ml_model_test
        limit 5
        '''

        pd.read_sql(qry, conn)
```

```
In [ ]: # and that tables have unique PANID values, ie a row is a household in t
        he given year
        qry = '''
        select count(*) recs, count(distinct panid)
        from iri_usda_2019_db.ml_model_train
        '''
        pd.read_sql(qry, conn)
```

```
In [ ]: # same for test set

        qry = '''
        select count(*) recs, count(distinct panid)
        from iri_usda_2019_db.ml_model_test
        '''
        pd.read_sql(qry, conn)
```

Now we should have everything we need from our Athena data tables to run some machine learning models to tackle our guiding question.