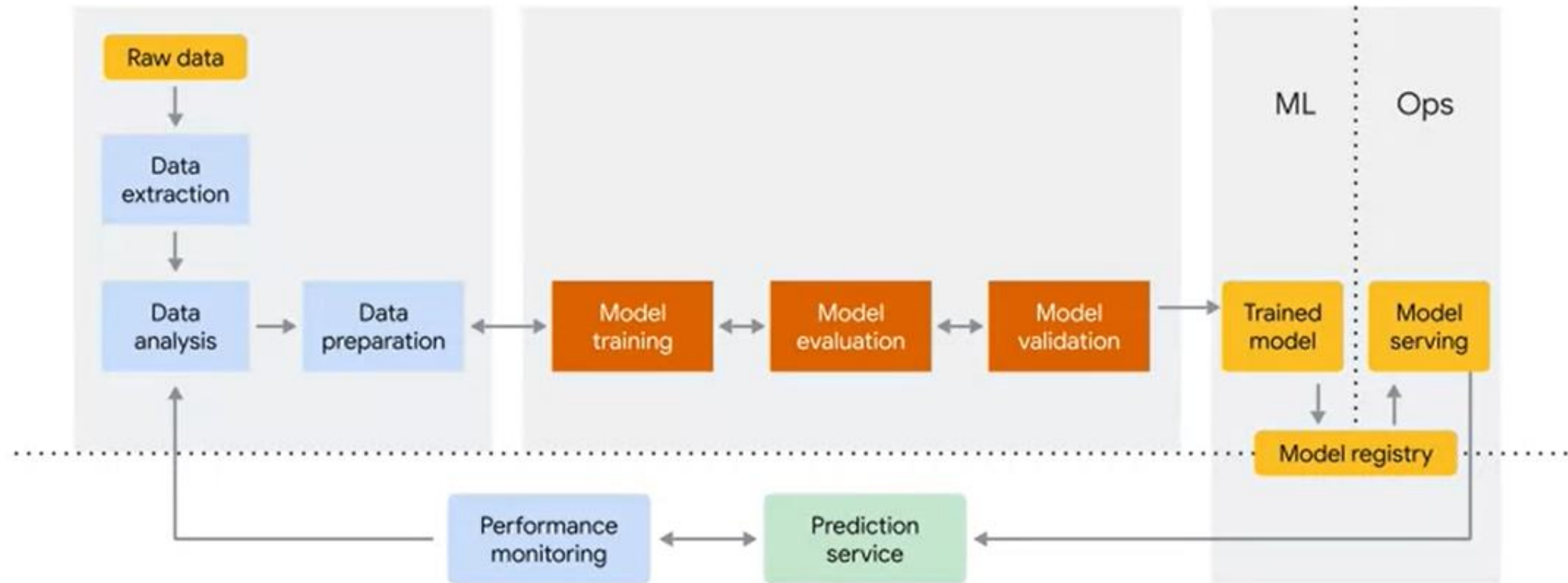


Launching into Machine Learning Google Cloud

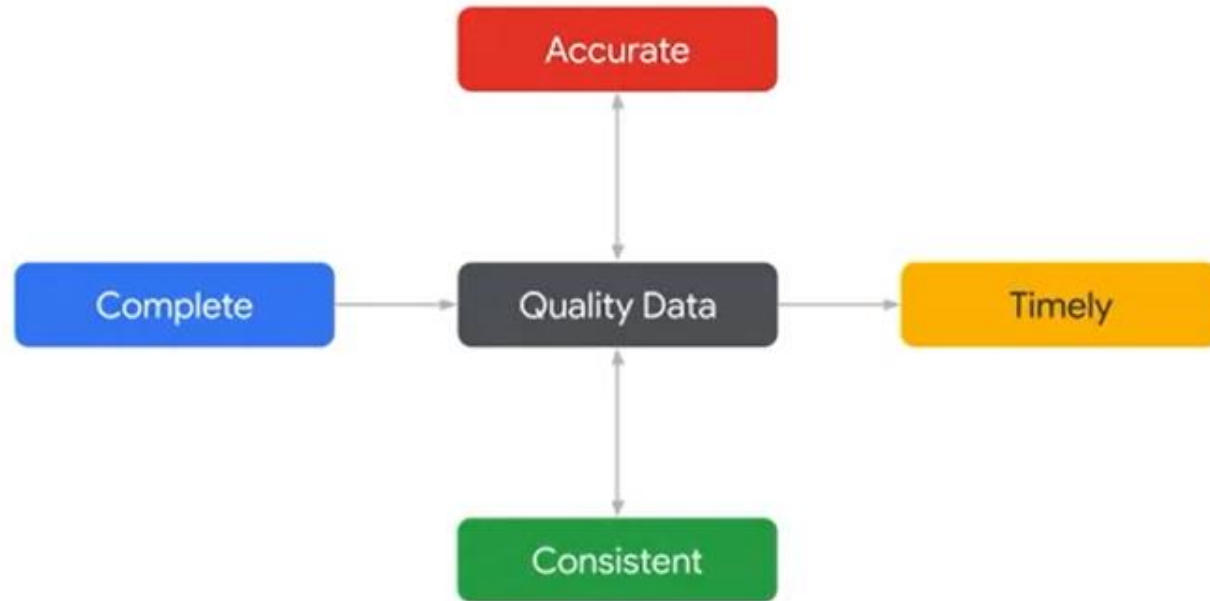
An ML pipeline recap

Staging/pre-production/production environments



Experimentation/development/test environments

Attributes related to the data quality



- 1 Accuracy of Data
- 2 Consistency of Data
- 3 Timeliness of Data
- 4 Completeness of Data

Ways to improve data quality



1

Resolve Missing
Values



2

Convert the
Date feature
column to
Datetime
Format



3

Parse date/time
features



4

Remove
unwanted values



5

Convert
categorical
columns to
"one-hot
encodings"

Convert the Date feature column to a **datetime** **format**

We can convert it to the
datetime with the
`to_datetime()` function in
Pandas.

```
In [19]: # TODO 2a
          df_transport['Date'] = pd.to_datetime(df_transport['Date'],
                                              format= '%m/%d/%Y')

In [20]: # TODO 2b
          df_transport.info() # Date is now converted
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 999 entries, 0 to 998
Data columns (total 7 columns):
Date                999 non-null datetime64[ns]
Zip Code            999 non-null object
Model Year          999 non-null object
Fuel                999 non-null object
Make                999 non-null object
Light_Duty          999 non-null object
Vehicles            999 non-null float64
dtype: datetime64[ns](1), float64(1), object(5)
memory usage: 54.8+ KB
```

Parse date

Let's parse Date into three columns (e.g., year, month, and day.)

```
In [21] df_transport['year'] = df_transport['Date'].dt.year
df_transport['month'] = df_transport['Date'].dt.month
df_transport['day'] = df_transport['Date'].dt.day
#df['hour'] = df['date'].dt.hour - you could use if your
#df['minute'] = df['date'].dt.minute - you could use this if
df_transport.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 999 entries, 0 to 998
Data columns (total 10 columns):
Date                999 non-null datetime64[ns]
Zip Code            999 non-null object
Model Year          999 non-null object
Fuel                999 non-null object
Make                999 non-null object
Light_Duty          999 non-null object
Vehicles            999 non-null float64
Year                999 non-null int64
Month               999 non-null int64
Day                 999 non-null int64
dtypes: datetime64[ns](1), float64(1), int64(3), objects(5)
memory usage: 78.2+ KB
```

Unwanted characters - Model Year

Let's investigate a bit more of our data by using the `.groupby()` function.

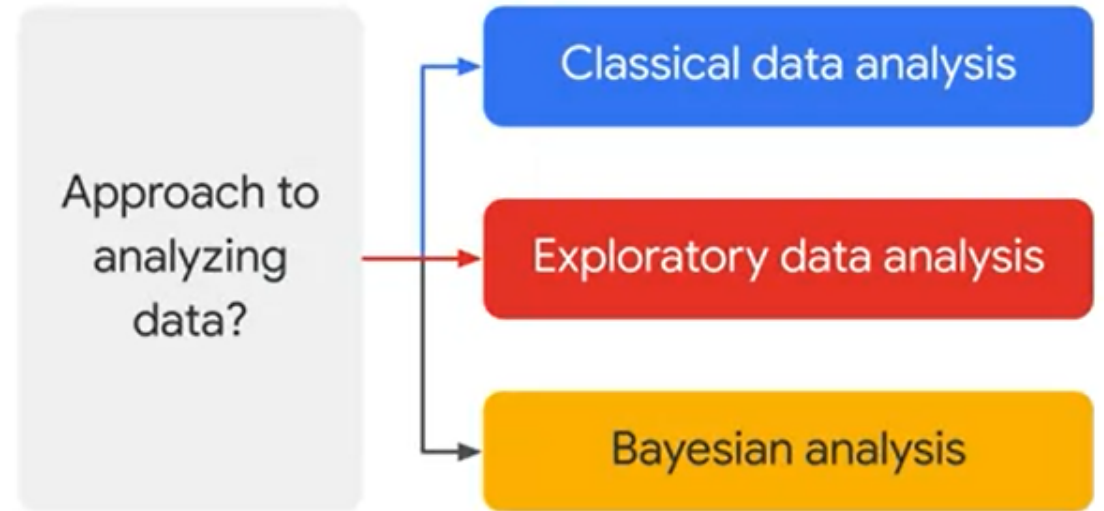
```
In [9]: grouped_data = df_transport.groupby(['Zip Code', 'Model Year', 'Fuel', 'Make', 'Light_Duty', 'Vehicles'])
        df_transport.groupby('Fuel').first() # Get the first entry for each month.
```

Out [9]:

Fuel	Date	Zip Code	Model Year	Make	Light_Duty	Vehicles
Battery Electric	10/1/2018	90000	<2006	OTHER/UNK	No	4.0
Diesel and Diesel Hybrid	10/1/2018	90000	<2006	OTHER/UNK	No	55.0
Flex-Fuel	10/14/2018	90001	2007	Type_A	Yes	78.0
Gasoline	10/1/2018	90000	2006	OTHER/UNK	Yes	1.0
Hybrid Gasoline	10/24/2018	90001	2009	OTHER/UNK	Yes	18.0
Natural Gas	10/25/2018	90001	2009	OTHER/UNK	No	2.0
Other	10/8/2018	90000	<2006	OTHER/UNK	Yes	6.0
Plug-in Hybrid	11/2/2018	90001	2012	OTHER/UNK	Yes	1.0

CDA vs. EDA vs. Bayesian?

What other approaches exist and how does exploratory data analysis differ from these other approaches?



CDA vs. EDA vs. Bayesian?

What other approaches exist and how does exploratory data analysis differ from these other approaches?

Problem => Data => Model => Analysis => Conclusions

Approach to
analyzing
data?

Classical data analysis



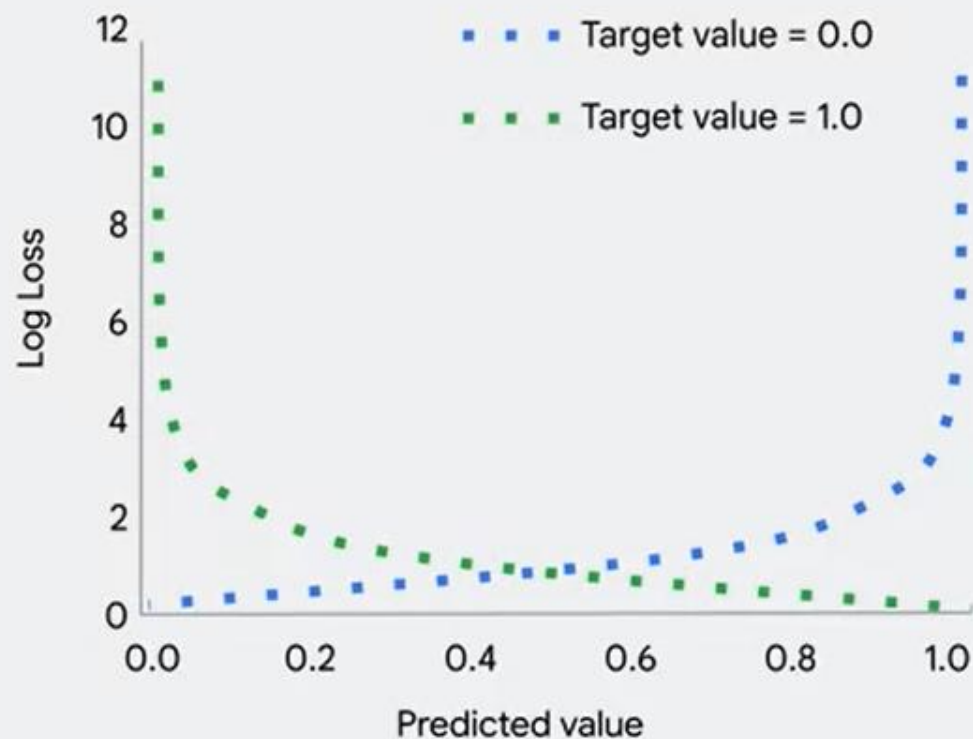
CDA vs. EDA vs. Bayesian?

What other approaches exist and how does exploratory data analysis differ from these other approaches?

Problem => Data => Model => Prior Distribution =>
Analysis => Conclusions

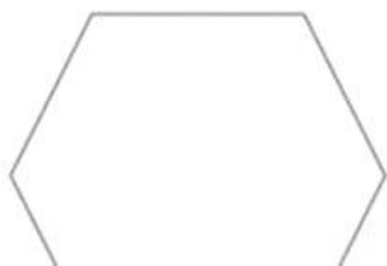
Approach to
analyzing
data?

Bayesian analysis



Typically, use cross-entropy (related to Shannon's information theory) as the error metric

Less emphasis on errors where the output is relatively close to the label.



$$\text{LogLoss} = \sum_{(x,y) \in D} -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

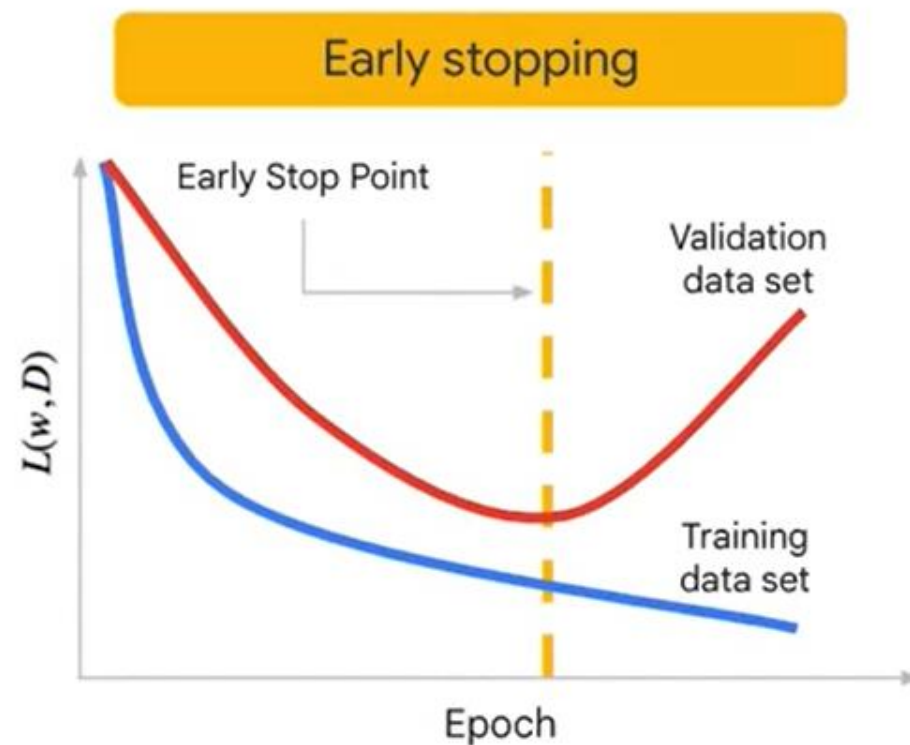
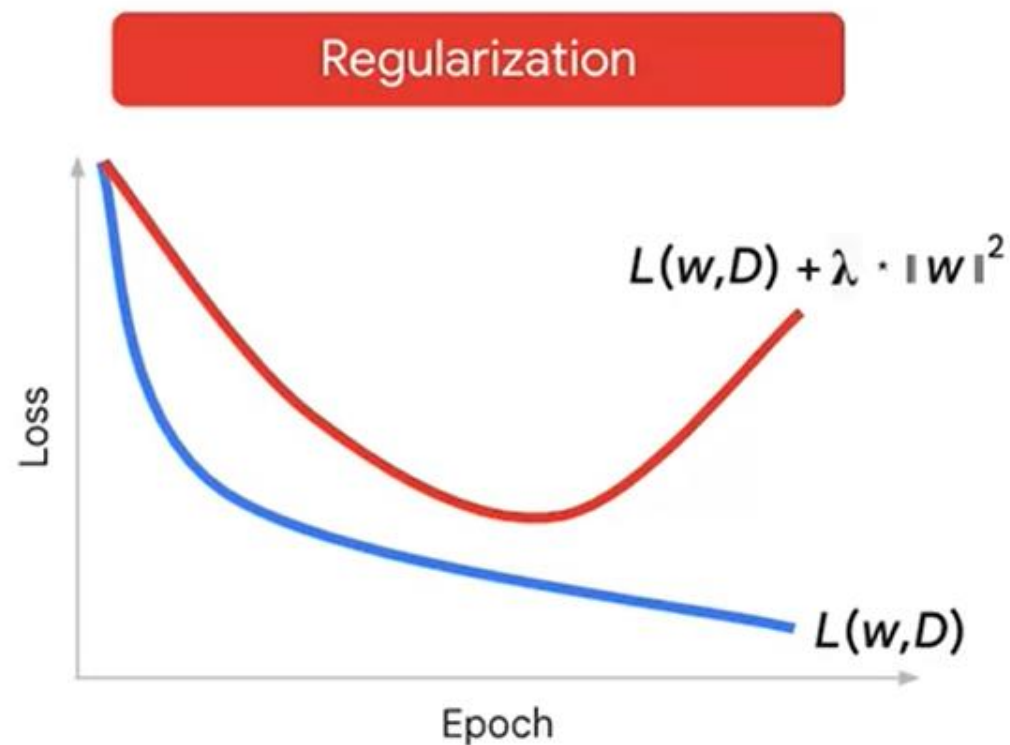
Quiz: Logistic regression regularization

Why is it important to add regularization to logistic regression?

- A. Helps stops weights being driven to \pm infinity.
- B. Helps logits stay away from asymptotes which can halt training
- C. Transforms outputs into a calibrated probability estimate
- D. Both A & B**
- E. Both A & C



Often we do both regularization and early stopping to counteract overfitting



In many real-world problems, the probability is not enough; we need to make a binary decision



Send the mail to
spam folder or not?



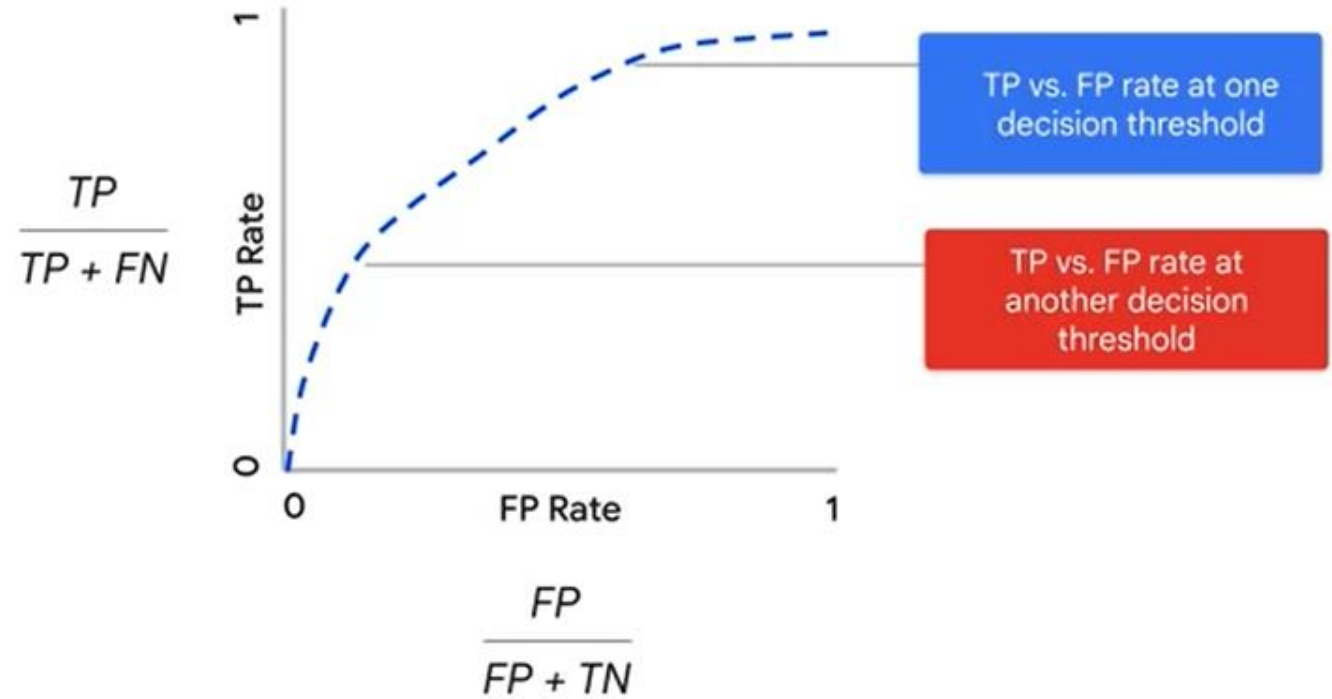
Approve the loan
or not?



Which road should we
route the user through?

Choice of threshold is important and can be tuned

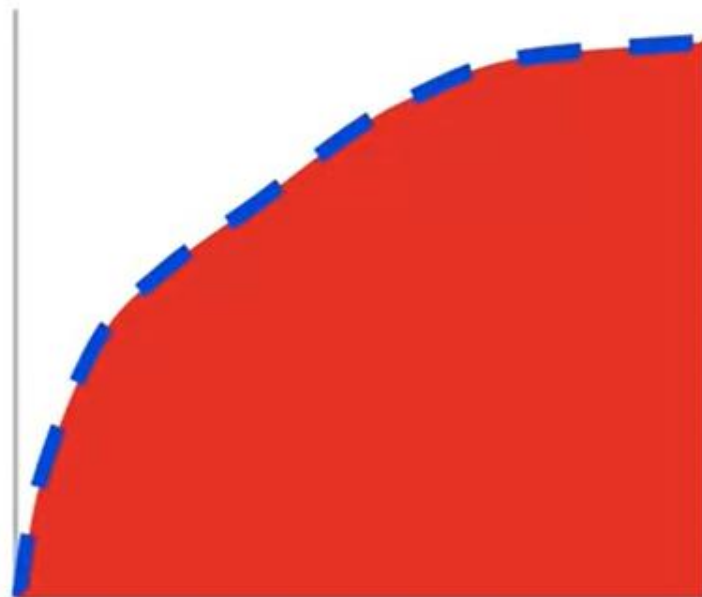
Use the ROC curve to choose the decision threshold based on **decision criteria**



The Area-Under-Curve (AUC) provides an aggregate measure of performance across all possible classification thresholds

AUC helps you choose between models when you don't know what decision threshold is going to be ultimately used.

"If we pick a random positive and a random negative, what's the probability my model scores them in the correct relative order?"



Logistic regression predictions should be unbiased

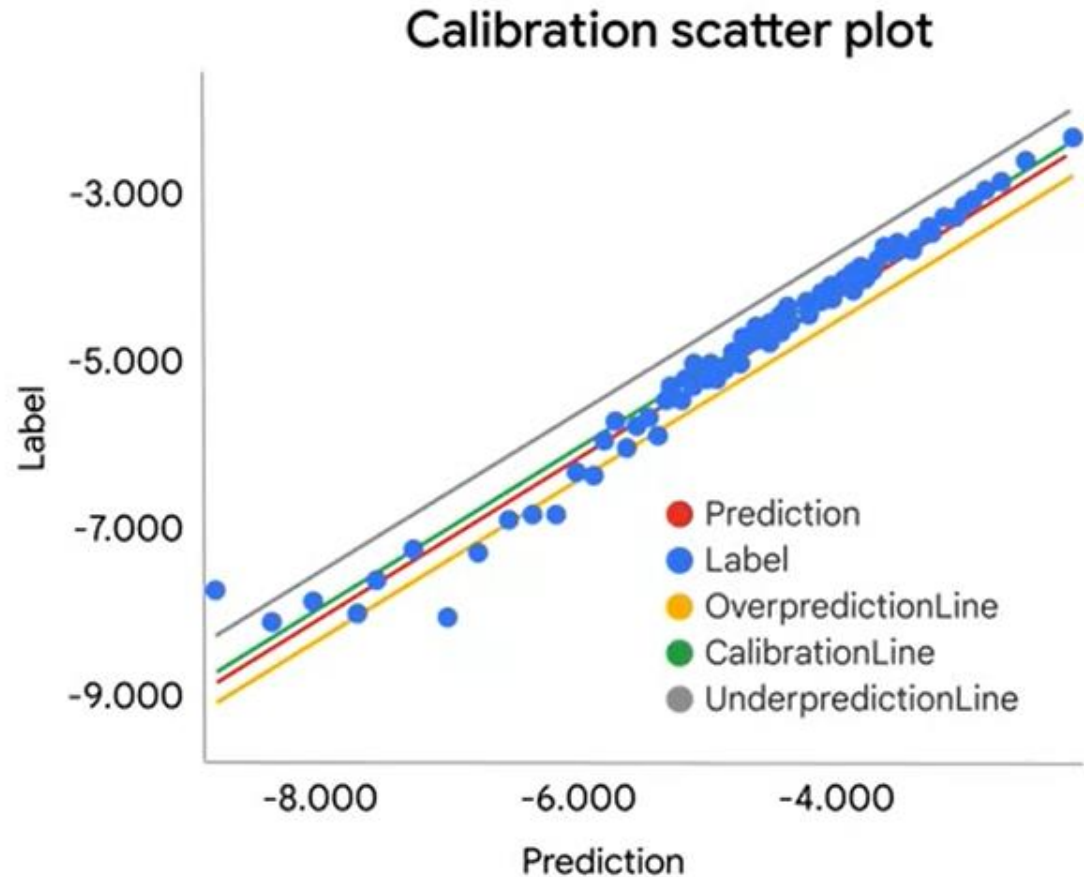
average of predictions == average of observations



Look for bias in slices of data, this can guide improvements.

Use calibration
plots of bucketed
bias to find slices
where your model
performs poorly

Each dot represents many examples in
the same bucketed prediction range



Quiz: Logistic regression

Which of these is important when performing logistic regression?

- A. Adding regularization
- B. Choosing a tuned threshold
- C. Checking for bias
- D. All of the above**



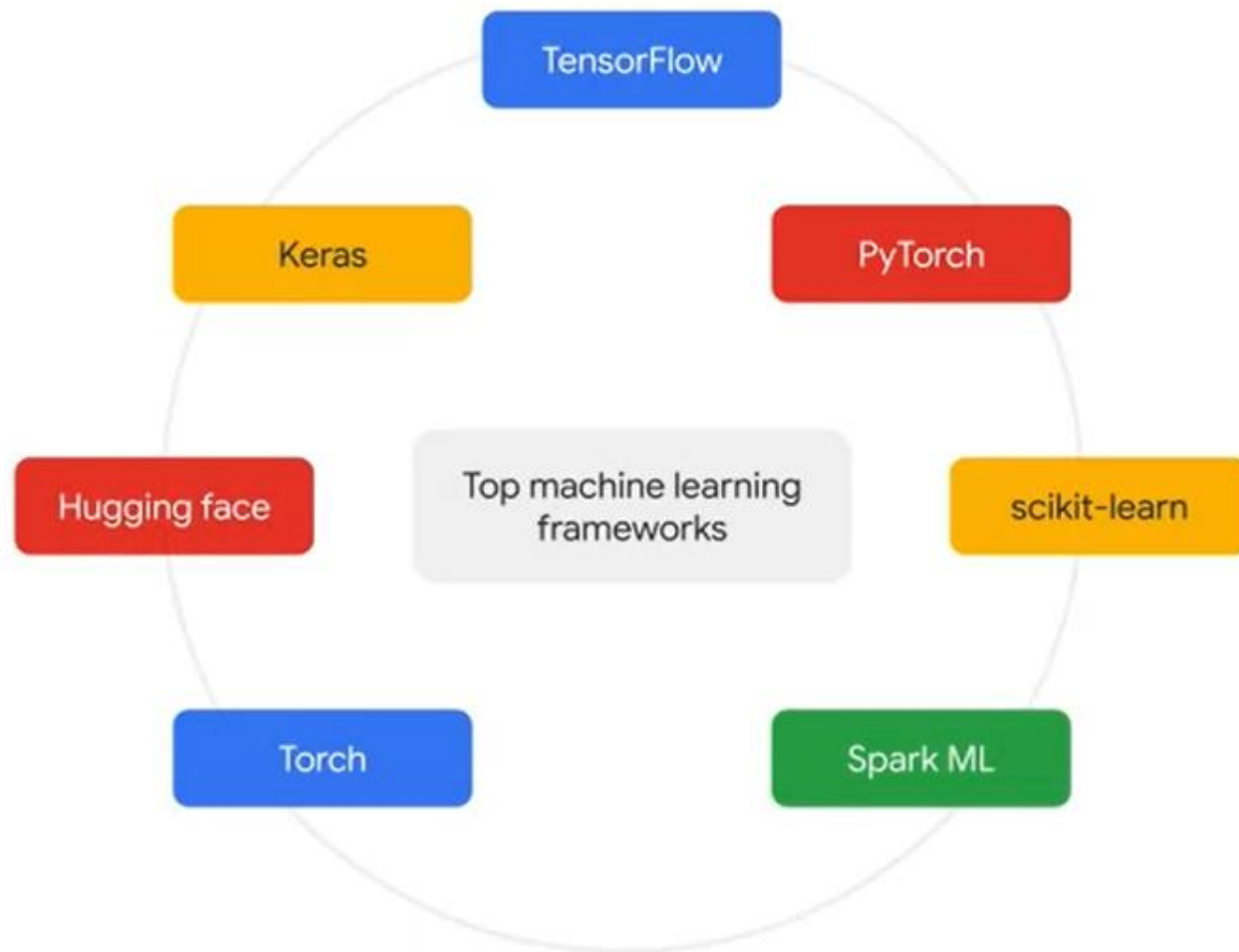


The machine learning pipeline



Pipelines are used to automate data preprocessing and model training for easy model upgrading.





Machine learning versus **statistics**



Machine learning

Lots of data. Keep outliers
and build models for them.



Statistics

"I've got all the data I'll
ever get." Throw away
outliers.

Machine learning versus statistics

	Machine learning	Standard statistics (linear/logistic regressions)
Data preparation?	Doesn't require explicit commands to find patterns in data	Need to know variables and parameters beforehand
Hypothesis	No hypothesis needed	Need hypothesis to test
Type of data?	Multi-dimensional data that can be non-linear in nature	Linear data
Training?	Needs to be "trained"	No training
Goal?	Generally better for predictions	Generally better for inferences/hypothesis testing
Scientific question?	What will happen?	How/why it happened?



Factors

- Data requirement
- Accuracy
- Training time
- Hardware dependency
- Hyperparameter tuning



Deep learning

- Requires large data
- Provides high accuracy
- Takes longer to train
- Requires GPU to train properly
- Can be tuned in various ways



Machine learning

- Can train on lesser data
- Gives lesser accuracy
- Takes less time to train
- Trains on CPU
- Limited tuning capabilities

Summary

01

Machine learning is a subfield of artificial intelligence.

02

The goal is to make computers learn from your data.

03

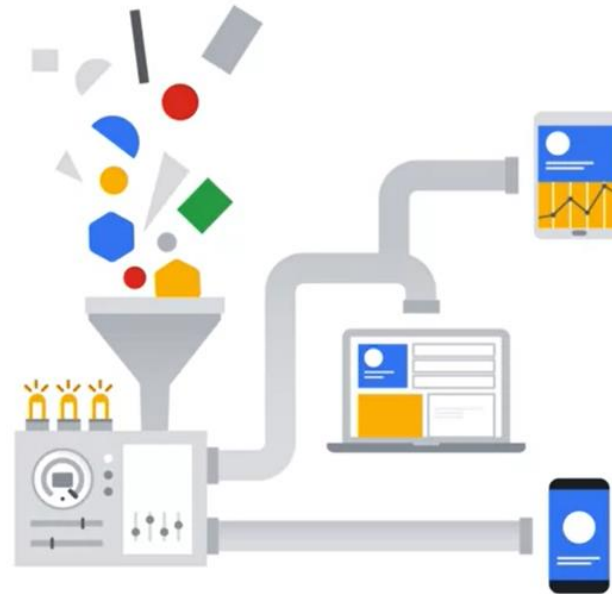
Your code provides an algorithm that adapts.

04

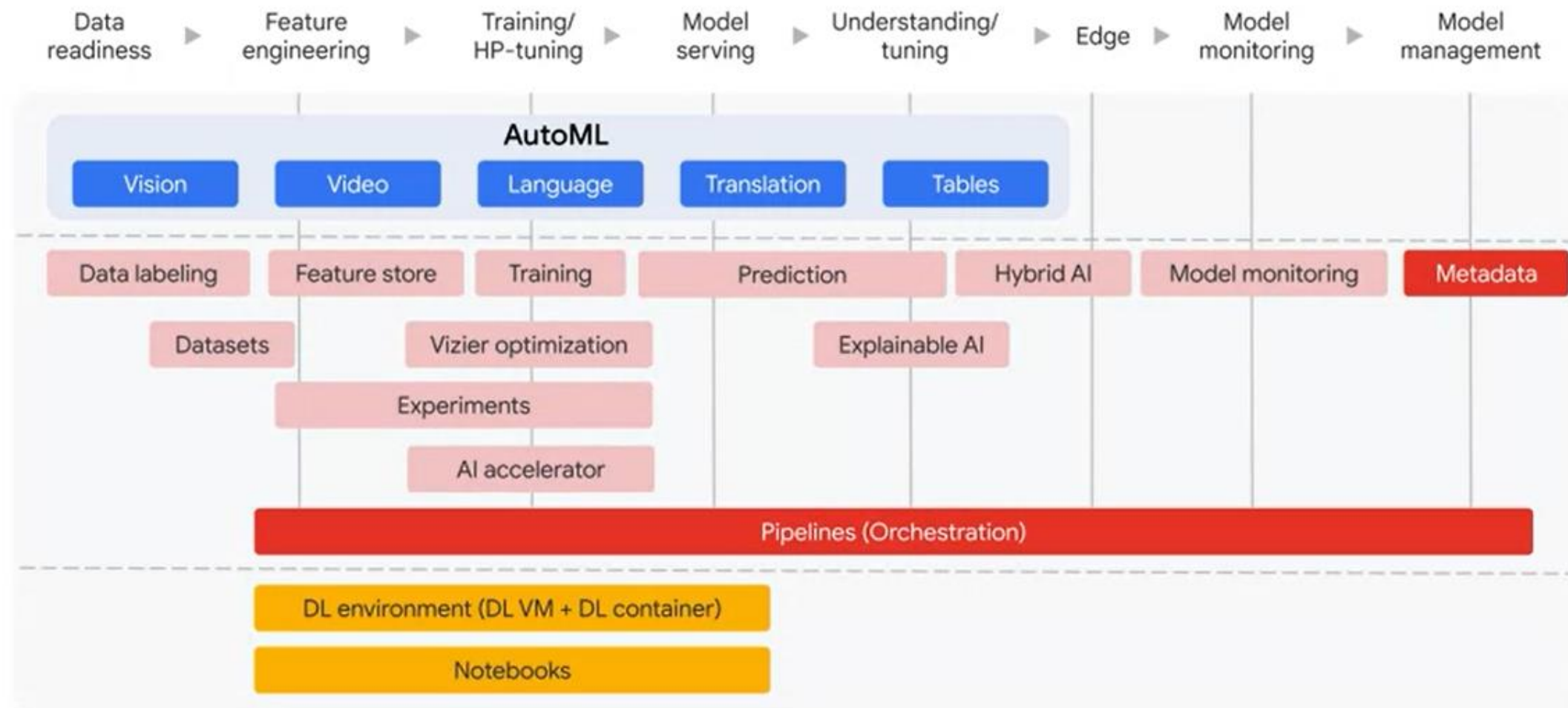
The result (the algorithm and the learned parameters) is your trained model.

What is automated machine learning?

Automate the process
of applying ML to
real-world problems



What's included in Vertex AI?



AutoML regression model

Press **Esc** to exit full screen

Select a data type and objective

First select the type of data your dataset will contain. Then select an objective, which is the outcome that you want to achieve with the trained model. [Learn more about model types](#)

IMAGE

TABULAR

TEXT

VIDEO



☒ Regression/classification

Predict a target column's value.
Supports tables with hundreds of columns and millions of rows.



☐ Forecasting **PREVIEW**

Predict the likelihood of certain events or demand.


```

%%writefile train/model_definition.py
# Here we'll import data processing libraries like Numpy and Tensorflow
import tensorflow as tf
import numpy as np

# Get data

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()

# add empty color dimension
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)

def create_model():
    # The `tf.keras.Sequential` method will sequential groups a linear stack of layers into a tf.keras.Model.
    model = tf.keras.models.Sequential()
    # The `Flatten()` method will flattens the input and it does not affect the batch size.
    model.add(tf.keras.layers.Flatten(input_shape=x_train.shape[1:]))
    # The `Dense()` method is just your regular densely-connected NN layer.
    model.add(tf.keras.layers.Dense(1028))
    # The `Activation()` method applies an activation function to an output.
    model.add(tf.keras.layers.Activation('relu'))
    # The `Dropout()` method applies dropout to the input.
    model.add(tf.keras.layers.Dropout(0.5))
    # The `Dense()` method is just your regular densely-connected NN layer.
    model.add(tf.keras.layers.Dense(512))
    # The `Activation()` method applies an activation function to an output.
    model.add(tf.keras.layers.Activation('relu'))
    # The `Dropout()` method applies dropout to the input.
    model.add(tf.keras.layers.Dropout(0.5))
    # The `Dense()` method is just your regular densely-connected NN layer.
    model.add(tf.keras.layers.Dense(256))
    # The `Activation()` method applies an activation function to an output.
    model.add(tf.keras.layers.Activation('relu'))
    # The `Dropout()` method applies dropout to the input.
    model.add(tf.keras.layers.Dropout(0.5))
    # The `Dense()` method is just your regular densely-connected NN layer.
    model.add(tf.keras.layers.Dense(10))

```


Where Vertex AI fits in the ML workflow

You can use Vertex AI to manage the following stages in the ML workflow:



Create a dataset and upload data.



Train an ML model on your data:

- Train the model
- Evaluate model accuracy
- Tune hyperparameters (custom training only)



Upload and store your model in Vertex AI.



Deploy your trained model to an endpoint for serving predictions.



Send prediction requests to your endpoint.



Specify a prediction traffic split in your endpoint.



Manage your models and endpoints.

Choose a training method

Auto ML

- Create and train a model with minimal technical effort.
- Quickly prototype models or explore datasets before developing in a custom training application.

Custom training

- Create a training application optimized for your targeted outcome.
- Maintain complete control over training application functionality.
 - Target any objective, use any algorithms, develop your own loss functions or metrics, or other customizations.

When to use AutoML and when to use custom training

	AutoML	Custom training
Data science expertise needed	No.	Yes, to develop the training application and also to do some of the data preparation like feature engineering.
Programming ability needed	No, AutoML is codeless.	Yes, to develop the training application
Time to trained model	Lower. Less data preparation is required, and no development is needed.	Higher. More data preparation is required, and training application development is needed.
Limits on machine learning objectives	Yes. You must target one of AutoML's predefined objectives.	No.
Can manually optimize model performance with hyperparameter tuning	No. AutoML does some automated hyperparameter tuning, but you can't modify the values used.	Yes. You can tune the model during each training run for experimentation and comparison.

When to use AutoML and when to use custom training

	AutoML	Custom training
Can control aspects of the training environment	Limited. For image and tabular datasets, you can specify the number of node hours to train for, and whether to allow early stopping of training.	Yes. You can specify aspects of the environment such as Compute Engine machine type, disk size, machine learning framework, and number of nodes.
Limits on data size	Yes. AutoML uses managed datasets; data size limitations vary depending on the type of dataset.	For unmanaged datasets, no. Managed datasets have the same limits as Vertex AI datasets that are used to train AutoML models.

Training an ML model using BigQuery ML

Which framework should they use?

AutoML or BigQuery ML

Vertex AI AutoML tabular data requirements

Required

- Columns: 2 (target and input feature)
- Target column must be categorical (2-500 values) or numerical
- Rows: 1,000

Maximum

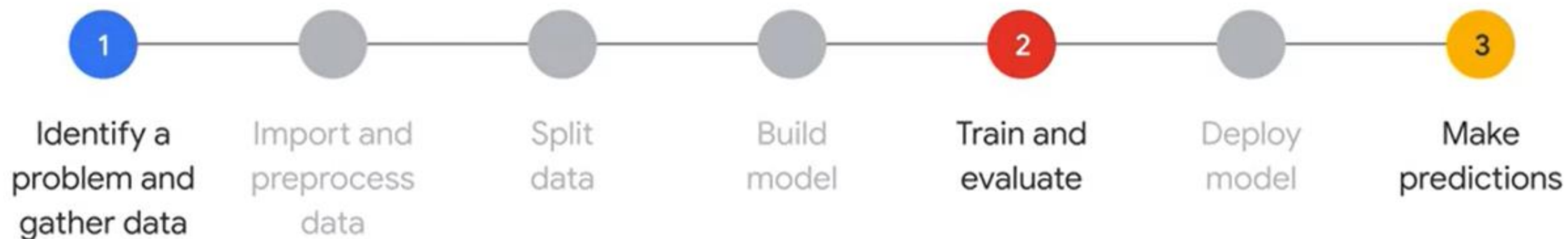
- Dataset size: 100 GB
- Columns: 1,000
- Rows: 100 million

Using BigQuery ML

- Allows you to use SQL to invoke machine learning models on structured data.
- Can provide decision-making guidance through predictive analytics by using its machine learning tool, BigQuery ML.
- Doesn't require exporting data out of BigQuery to create and train a model.



Custom model **with** BigQuery ML



Working with BigQuery ML



01

Dataset

02

Create/ train

```
CREATE MODEL `BigQuery  
ML_tutorial.sample_model`  
OPTIONS(model_type='logistic_reg') AS  
SELECT
```

```
FROM  
  ML.EVALUATE(MODEL `BigQuery  
ML_tutorial.sample_model`,  
    TABLE eval_table)
```

03

Evaluate

04

Predict/ classify

```
FROM  
  ML.PREDICT(MODEL `BigQuery  
ML_tutorial.sample_model`,  
    table game_to_predict) ) AS  
predict
```

Build and train with **CREATE MODEL**

```
CREATE OR REPLACE MODEL  
mydataset.model_linreg
```

```
OPTIONS(  
  input_label_cols=['fare_amount'],  
  model_type='linear_reg') AS
```

```
SELECT  
  fare_amount,  
  pickup_longitude,  
  pickup_latitude,  
  dropoff_longitude,  
  dropoff_latitude,  
  passenger_count
```

```
FROM  
  `nyc-tlc.yellow.trips`
```

Evaluate with ML.EVALUATE

```
SELECT  
  *
```

```
FROM
```

```
  ML.EVALUATE(  
    MODEL mydataset.model_linreg  
  )
```

Use the model
with **ML.PREDICT**

```
SELECT
  *
FROM
  ML.PREDICT(MODEL mydataset.model_linreg,
  (
    SELECT
      fare_amount,
      pickup_longitude,
      pickup_latitude,
      dropoff_longitude,
      dropoff_latitude,
      passenger_count
    FROM
      `nyc-tlc.yellow.trips`
  ))
```

BigQuery ML supported models and features

Classification

- Logistic regression
- DNN classifier (TensorFlow)
- XGBoost
- AutoML Tables
- Wide and Deep NNs^{Preview, GA H1'21}

Other Models

- k-means clustering
- Time series forecasting
- Recommendation: Matrix factorization
- Time series anomaly detection^{Preview Q2'21, GA H2'21}

Regression

- Linear regression
- DNN regressor (TensorFlow)
- XGBoost
- AutoML Tables
- Wide and Deep NNs^{Preview, GA H1'21}

Model ops and explainability

- Import/export TensorFlow models for batch and online prediction
- hyperparameter tuning using Cloud AI Vizier^{Preview H1'21, GA H2'21}
- Model explainability using Cloud AI^{Preview H1'21, GA H2'21}
- Managed Kubernetes and TFX pipelines^{Preview H2'21, GA 2022}
- List models for comparison and online deployment in Cloud AI^{Preview H2'21, GA 2022}
- Model versioning, continuous monitoring^{future}

BigQuery ML hyperparameter tuning

A **hyperparameter** is a model argument whose value is set before the learning process begins.



Less time manually iterating hyperparameters




More time focusing on exploring insights from data

Hyperparameter tuning with BigQuery ML

BigQuery ML supports hyperparameter tuning when training ML models using CREATE MODEL statements.

Hyperparameter tuning is commonly used to improve model performance by searching for the optimal hyperparameters.

Hyperparameter tuning supports the  following model types.

- LINEAR_REG
- LOGISTIC_REG
- KMEANS
- MATRIX_FACTORIZATION
- BOOSTED_TREE_CLASSIFIER
- BOOSTED_TREE_REGRESSOR
- DNN_CLASSIFIER
- DNN_REGRESSOR

Hyperparameter tuning

A Simple DNN

```
CREATE OR REPLACE MODEL `BigQuery  
ML.natality_dnn_hp`  
  OPTIONS (MODEL_TYPE = 'DNN_REGRESSOR',  
    INPUT_LABEL_COLS = ['weight_pounds'] ,  
    DATA_SPLIT_METHOD = 'RANDOM',  
    EARLY_STOP=TRUE,  
    HIDDEN_UNITS = [30,50]  
  ) AS  
SELECT *  
FROM `BigQuery ML.natality`
```

DNN with hyperparameter tuning

```
CREATE OR REPLACE MODEL `BigQuery ML.natality_dnn_hp`  
  OPTIONS (MODEL_TYPE = 'DNN_REGRESSOR',  
    INPUT_LABEL_COLS = ['weight_pounds'] ,  
    DATA_SPLIT_METHOD = 'RANDOM',  
    EARLY_STOP=TRUE,  
    HIDDEN_UNITS = [30,50]  
    --  
    -- HP tuning options  
    NUM_TRIALS = 10,  
    MAX_PARALLEL_TRIALS = 2,  
    DROPOUT=HPARAM_RANGE(0, 0.2),  
    OPTIMIZER = HPARAM_CANDIDATES(['adam', 'adagrad']),  
    LEARN_RATE=HPARAM_CANDIDATES([.001, .01])  
  ) AS  
SELECT *  
FROM `BigQuery ML.natality`
```

Deep Neural Network (DNN) models **without** hyperparameter tuning

```
CREATE or replace MODEL BigQuery
ML.wine_dnn
OPTIONS(MODEL_TYPE='DNN_CLASSIFIER',
        INPUT_LABEL_COLS = ['Quality'])
as select *
from BigQuery ML.ml_wine_quality;
```

Aggregate metrics	
Threshold	0.0000
Precision	0.2888
Recall	0.2558
Accuracy	0.5691
F1 score	0.2572
Log loss	2.8729
ROC AUC	0.5351

Deep Neural Network (DNN) models

```
CREATE or replace MODEL BigQuery ML.wine_dnn
OPTIONS(MODEL_TYPE='DNN_CLASSIFIER',
        INPUT_LABEL_COLS = ['Quality'])
as select *
from BigQuery ML.ml_wine_quality;
```

Aggregate metrics

Threshold	0.0000
Precision	0.2888
Recall	0.2558
Accuracy	0.5691
F1 score	0.2572
Log loss	2.8729
ROC AUC	0.5351

Defaults:

- Activation_FN = 'RELU'
- Auto_Class_Weights = FALSE
- Batch_Size = 32 (or n)
- Dropout = 0.0
- Early_Stop = True
- Hidden_Units = [128*]
- Learn_Rate = 0.01
- Max_Iterations = 20
- Min_Rel_Progress = 0.1 (Early_Stop = True)
- Optimizer = 'ADAM'
- Warm_Start = False
- Data_Split_:
 - Method = 'AUTO_SPLIT'
 - EVAL_FRACTION
 - COL

```
CREATE or replace MODEL
BigQuery ML.wine_dnn_learnrate_005
OPTIONS(MODEL_TYPE='DNN_CLASSIFIER',
        DROPOUT = 0.1,
        EARLY_STOP = TRUE,
        INPUT_LABEL_COLS = ['Quality'],
        LEARN_RATE=0.015,
        MAX_ITERATIONS = 20)
as select *
from BigQuery ML.ml_wine_quality;
```

Aggregate metrics

Threshold	0.0000
Precision	0.2891
Recall	0.2471
Accuracy	0.5884
F1 score	0.2394
Log loss	2.3478
ROC AUC	0.7989

How to build and deploy a recommendation system with BigQuery ML

Common use case

- Prepare your training data in BigQuery
- Train a recommendation system with BigQuery ML
- Use the predicted recommendations in production



Build and train with CREATE MODEL

Training data

	visitorId	itemID	session_duration
0	5468761641774363851-1	GGOEGAAX0031	59
1	7186762836575002506-1	GGOEGAAX0031	5787
2	1827763305655925232-3	GGOEGAAX0031	6423
3	3863873694540855771-4	GGOEGAAX0031	6168
4	2046797444872582027-2	GGOEGAAX0031	6891
5	5885261202339404877-2	GGOEGAAX0031	85256
6	0097216552247524030-5	GGOEGAAX0031	39980
7	860497822069663220-1	GGOEGAAX0031	2486
8	8960897138106393989-1	GGOEGAAX0031	11641
9	9681217015581152227-1	GGOEGAAX0031	6398

```
CREATE OR REPLACE MODEL
```

```
  BigQuery ML.retail_recommender
```

```
OPTIONS(
```

```
  model_type='matrix_factorization',
```

```
  user_col='visitorId',
```

```
  item_col='itemId',
```

```
  rating_col='session_duration',
```

```
  feedback_type='implicit'
```

```
) AS
```

```
SELECT
```

```
  *
```

```
FROM
```

```
  BigQuery ML.aggregate_web_stats
```


Evaluate the model with ML.EVALUATE

```
SELECT
  *

FROM
  ML.EVALUATE(
    MODEL BigQuery ML.retail_recommender
  )
```

CC

Row	mean_average_precision	mean_squared_error	normalized_discounted_cumulative_gain	average_rank
1	0.011463546060150412	1.0999974250389957E-1	18.68240758482596	0.28084605495250364

Introduction to loss functions

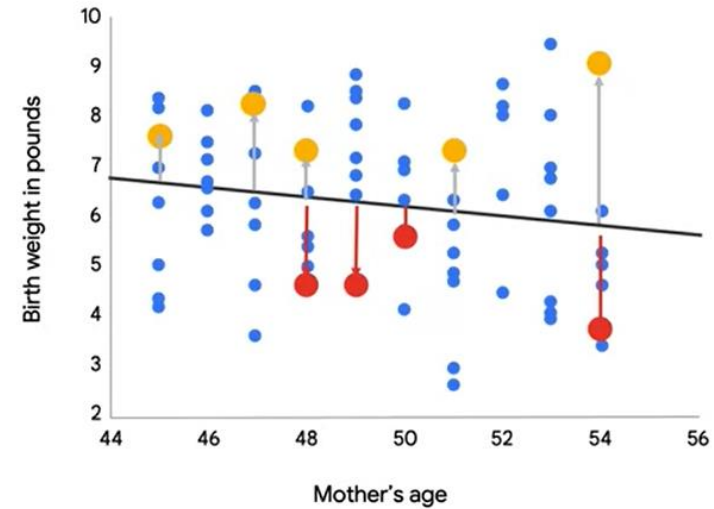
Compose a loss function by calculating errors

Each error makes sense. How about all the errors added together?

Error = actual (true) - predicted value

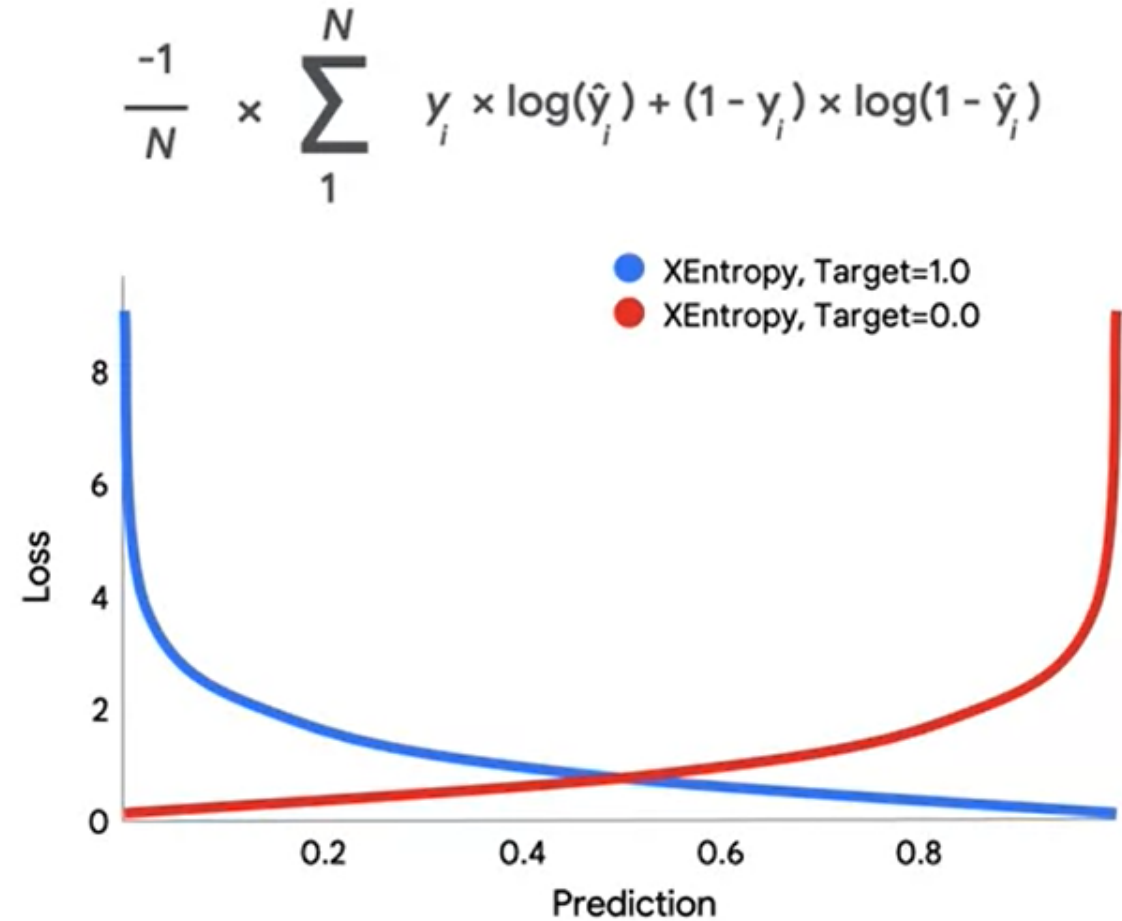
Compute the errors:

+0.70
+1.10
+0.65
-1.20
-1.15
+1.10
+3.09
-2.10



Problem: RMSE doesn't work as well for classification

Bad classifications are penalized
appropriately.



$$\frac{-1}{N} \times \sum_1^N \overbrace{y_i \times \log(\hat{y}_i)}^{\text{Positive term}} + \overbrace{(1 - y_i) \times \log(1 - \hat{y}_i)}^{\text{Negative term}}$$

Computing cross-
entropy **loss**

- Optimization was framed as a search in parameter-space.
- Loss functions were introduced as a way to compare these points.

