

Machine Learning Engineer Learning Path

mardi 13 juin 2023 15:28

Google Cloud Big Data and Machine Learning Fundamentals

Big data and Machine learning on Google Cloud

Google Cloud infrastructure:

based in five major **geographic locations: North America, South America, Europe, Asia, and Australia**

Having **multiple service locations** is important because choosing where to locate applications affects **qualities like availability, durability, and latency**, which measure travel from its source to its destination.

Each of these **locations** is divided into several different **regions** and **zones**.

Regions represent **independent geographic areas**, and are **composed of zones**.

A zone is an area where **Google Cloud resources are deployed**.

If a zone becomes unavailable, the resources won't be available either

Google Cloud lets users **specify the geographical locations to run services and resources**.

In many cases, you can even specify the location on a **zonal, regional, or multi-regional level**

This is useful for **bringing applications closer to users** around the world, and also for **protection** in case there are **issues with an entire region**, say, due to a **natural disaster**.

A few of Google Cloud's services support placing resources in what we call a **multi-region**.

For example, **Cloud Spanner multi-region configurations** allow you to replicate the database's data not just in multiple zones, but in **multiple zones across multiple regions**.

These **additional replicas** enable you to read data with **low latency** from multiple locations **close to or within the regions** in the configuration, like The Netherlands.

103 zones in 34 regions

cloud.google.com/about/locations

3 layers: 1) Networking & security, 2) compute, storage, 3) Big data ML products

Compute:

compute power:

- **Compute Engine:**

is an **IaaS** offering, or **infrastructure as a service**, which provides **compute, storage, and Network** resources **virtually** that are similar to **physical data centers**.

Compute Engine provides **maximum flexibility** for those who prefer to manage server instances themselves

- Google **Kubernetes Engine**, or **GKE**: runs **containerized** applications in a **cloud environment**, as opposed to on an **individual virtual machine**, like Compute Engine. (cc dependencies.)

- **App Engine**,

a **fully managed PaaS** offering, or **platform as a service**:

PaaS offerings **bind code to libraries**.

- **Cloud Functions**

is often referred to as functions as a service.

executes code in response to events

serverless execution environment.

- **Cloud Run**:

a **fully managed** compute platform that enables you to run request or event-driven **stateless** workloads without having to worry about servers.

It automatically scales up and down from zero

Google Photos:

automatic video stabilization

video itself, which is really a large collection of individual images, along with **time series data** on the camera's **position** and **orientation** from the onboard **gyroscope**. It takes an unstable video, stabilizes it to minimize movement.

the hardware on a smartphone is not powerful enough to train sophisticated ML models.

That's why Google trains production machine learning models on a **vast network of data centers**, only to then **deploy smaller, trained versions** of the models to the **mobile hardware**.

Moore's Law, with the required computing power used in the largest AI training runs **doubling every two years**

TPU

introduced by Google **2016** to overcome CPU and GPU.

TPU is **application-specific integrated circuits (ASICs)**.

domain-specific hardware (general-specific : GPU, CPU)

TPUs are **generally faster** than current GPUs and CPUs for **AI applications and machine learning**.

They are also **significantly more energy-efficient**.

Cloud TPUs have been integrated across **Google products**, making this **state-of-the-art hardware and supercomputing technology** available to Google Cloud customers

Storage

Cloud (compute-storage decoupled) vs desktop (compute-storage coupled) computing

fully managed database and storage services: Cloud Storage, Cloud Bigtable, Cloud SQL, Cloud Spanner, Firestore And BigQuery

Goal=reduce the time and effort needed to store data (creating an elastic storage bucket directly in a web interface or through a command line for example on Cloud Storage). Storage depends on data type, and business need.

(un)Structured data = (non)tabular

Unstructured suited to Cloud Storage (also BigQuery),

An object is an immutable piece of data consisting of a file of any format. You store objects in containers called buckets. All buckets are associated with a project, and you can create buckets in any project. Each project, bucket, and object in Google Cloud is a resource in Google Cloud,

Cloud Storage classes (suited for unstructured data):

Standard (best for frequently accessed, or hot data stored for only brief periods of time)

Nearline (infrequently accessed data: data backups, long-tail multimedia content, or data archiving.)

Coldline (low-cost option for storing infrequently accessed data once every 90 days)

Archive (lowest-cost option, used ideally for data archiving, online backup, and disaster recovery. access less than once a year.)

Structured data:

transactional workloads (stem from Online Transaction Processing systems, which are used when fast data inserts and updates are required to build row-based records that impact only a few records.)

and analytical workloads (stem from Online Analytical Processing systems, which are used when entire datasets need to be read. require complex queries, for example, Then determine whether data will be accessed using SQL or not:
 Transactional-SQL-local/regional scalability-Cloud SQL
 Transactional-SQL-global scalability-Cloud Spanner
 Transactional-NoSQL-FireStore
 Analytical-SQL-BigQuery (analyze petabyte-scale datasets)
 Analytical-NoSQL-Cloud BigTable (best for real-time, high-throughput applications that require only millisecond)

Big data:

2002: Google File System, GFS, was designed to handle data sharing and petabyte storage at scale.
 2004: MapReduce
 2008: Dremel: breaking the data into smaller chunks called shards, and then compressing them. then uses a query optimizer to share tasks between the many shards of data.
 BigQuery.
 2010: Colossus, in 2010, which is a cluster-level file system and successor to the Google File System. It is a Platform as a Service (PaaS) that supports querying using ANSI SQL
 2012: Spanner, in 2012, which is a globally available and scalable relational database.
 2015: Pub/Sub, in 2015, which is a service used for streaming analytics and data integration pipelines to ingest and distribute data.
 2015: TensorFlow, also in 2015, which is a free and open source software library for machine learning and artificial intelligence.
 2018: brought the release of the Tensor Processing Unit, or TPU, and AutoML, as a suite of machine learning products.
 2021: Vertex AI, a unified ML platform
 Unified and stable platform: Cloud Storage Dataproc Cloud Bigtable BigQuery Dataflow Firestore Pub/Sub Looker Cloud Spanner AutoML, and Vertex AI,
 Product categories along the data-to-AI workflow:
 ingestion & process: to digest both real-time and batch data (Pub/Sub, Dataflow (streaming data processing), Dataproc, Cloud Data Fusion)
 storage: (Cloud Storage, Cloud SQL (relational), Cloud Spanner (relational), Cloud Bigtable (NoSQL), and Firestore (NoSQL))
 analytics: (BigQuery: a fully managed data warehouse that can be used to analyze data through SQL commands; Looker, and Looker Studio)
 ML: ML development platform and the AI solutions. (Vertex AI which includes the products and technologies: AutoML, Vertex AI Workbench, and TensorFlow)
 AI solutions are built on the ML development platform and include state-of-the-art products to meet both horizontal and vertical market needs: Document AI, Cloud Healthcare Data Engine

Example:

A unicorn is a privately held startup business valued at over US\$1 billion.
 For data latency: Dataflow (streaming data processing) + BigQuery (real-time business insights)
 For geospatial: Dataflow (a streaming event data pipeline).
 driver locations ping Pub/Sub every 30 seconds, and
 Dataflow would process the data (was able to automatically manage the number of workers processing the pipeline to meet demand)
 The pipeline would aggregate the supply pings from the drivers against the booking requests.
 This would connect to Gojek's notification system to alert drivers where they should go
 Conclusion: actively monitor requests to ensure that drivers are in the areas with the highest demand. This brings faster bookings for riders and more work for the drivers.

Data Engineering for streaming data

real-time data solution:

Ingest streaming data using Pub/Sub (data ingestion is where large amounts of streaming data are received)
 Process the data with Dataflow, and
 Visualize the results with Looker and Looker Studio
 (In between data processing with Dataflow and visualization with Looker or Looker Studio, the data is normally saved and analyzed in a data warehouse such as BigQuery)
 design streaming pipelines with Apache Beam

streaming vs batch processing (media streaming vs downloading):

Batch processing is when the processing and analysis happens on a set of stored data: Payroll and billing systems
 Streaming data is a flow of data records generated by various data sources: fraud detection or intrusion detection. data is analyzed in near real-time

4Vs: data engineers and data scientists four major challenges = variety, volume, velocity, and veracity:

data could come in from a variety of different sources and in various formats
 Veracity: data quality (Due to several data types and sources, big data often has many data dimensions)

IoT devices challenge:

data can be streamed from many different methods and devices
 it can be hard to distribute event messages (notification) to the right subscribers
 data can arrive quickly and at high volumes.

ensuring services are reliable, secure, and perform as expected

Pub/Sub:

a tool to handle distributed message-oriented architectures at scale

(Publisher/Subscriber, or publish messages to subscribers).

distributed messaging service

Pub/Sub's APIs are open, the service is global by default, and it offers end-to-end encryption

end-to-end encryption:

Pub/Sub reads, stores, broadcasts to any subscribers of this data topic that new messages are available

A central element of Pub/Sub is the topic.

Designing streaming pipelines with Apache Beam:

streaming input sources + pipe data into a data warehouse (dataflow)

"Process" : extract, transform, and load data, or ETL.

pipeline design phase questions:

Will the pipeline code be compatible with both batch and streaming data? Or need to be refactored?

Will the pipeline code software development kit, or SDK, being used have all the transformations, mid-flight aggregations and windowing and be able to handle late data?

Are there existing templates or solutions that should be referenced?

pipeline design: Apache Beam: an open source, unified programming model to define and **execute data processing pipelines**, including ETL, batch, and stream processing unified, which means it uses a single programming model for both batch and streaming data

It's portable, which means it can work on multiple execution environments, like Dataflow and Apache Spark

extensible, which means it allows you to write and share your own connectors and transformation libraries.

Apache Beam provides pipeline templates (Java, Python, or Go)

Implementing streaming pipelines on Cloud Dataflow:

Questions:

How much maintenance overhead is involved?

Is the infrastructure reliable?

How is the pipeline scaling handled?

How can the pipeline be monitored?

Is the pipeline locked in to a specific service provider?

Dataflow is a fully managed service for executing Apache Beam pipelines within the Google Cloud

Dataflow is serverless and NoOps

NoOps: doesn't require management from an operations team, because maintenance, monitoring, and scaling are automated.

Serverless computing is a cloud computing execution model

Dataflow tasks :

optimizing a pipeline model's execution graph

schedules out distributed work

scaler

auto-heals any worker faults

rebalances efforts to most efficiently use its workers.

execution engine to process and implement data processing pipelines

you don't need to monitor all of the compute and storage resources that Dataflow manages,

Dataflow templates:

streaming templates (for processing continuous, or real-time, data): Pub/Sub to BigQuery, Pub/Sub to Cloud Storage, Datastream to BigQuery, Pub/Sub to N

batch templates (for processing bulk data, or batch load data.): BigQuery to Cloud Storage, Bigtable to Cloud Storage, Cloud Storage to BigQuery, Cloud Spar

utility templates (activities related to bulk compression, deletion, and conversion)

Visualization with Looker:

Looker: semantic modeling layer on top of databases using Looker Modeling Language, or LookML

LookML defines logic and permissions independent from a specific database or a SQL language,

The Looker platform is 100% web-based, which makes it easy to integrate into existing workflows and share with multiple teams at an organization

Looker API, which can be used to embed Looker reports in other applications.

Looker features:

Dashboards: schedule its delivery through storage services like: Google Drive, Slack, Dropbox.

Visualization with Data Studio:

Looker Studio: integrated into BigQuery: doesn't require support from an administrator to establish a data connection, which is a requirement with Looker

integrated into Google Analytics, Google Cloud billing dashboard

to create a Looker Studio dashboard:

choose a template (a pre-built template or a blank report)

link the dashboard to a data source.

Lab introduction: Creating a streaming data pipeline for a Real-Time dashboard with Dataflow:

gcloud is the command-line tool for Google Cloud.

pre-installed on Cloud Shell and supports tab-completion

list the active account name: gcloud auth list

list the project ID: gcloud config list project

to create dataset: Google Cloud Shell or the Google Cloud Console.

[Pub/Sub](#) is an asynchronous global messaging service.

decoupling senders and receivers, it allows for secure and highly available communication between independently written applications

delivers low-latency, durable messaging

publisher applications and subscriber applications connect with one another through the use of a shared string called a **topic**

Google maintains a few public Pub/Sub streaming data topics

[BigQuery](#) is a serverless data warehouse

Tables in BigQuery are organized into datasets.

to create a new BigQuery dataset:

command-line tool (**Cloud Shell**):

Create dataset: bq --location=us-west1 mk taxirides

Create Table: bq --location=us-west1 mk \

--time_partitioning_field timestamp \

--schema ride_id:string,point_idx:integer,latitude:float,longitude:float,\

timestamp:timestamp,meter_reading:float,meter_increment:float,ride_status:string,\

passenger_count:integer -t taxirides.realtime

BigQuery Console UI:

Create a Cloud Storage bucket (to provide working space for your Dataflow pipeline)

[Cloud Storage](#) allows world-wide storage and retrieval of any amount of data at any time

Applications: serving website content, storing data for archival and disaster recovery, or distributing large data objects to users via direct download.

Set up a Dataflow Pipeline

set up a streaming data pipeline to read sensor data from Pub/Sub, compute the maximum temperature within a time window, and write this out to B

Restart the connection to the Dataflow API.

Create a new streaming pipeline

Analyze the taxi data using BigQuery:

```
SELECT * FROM taxirides.realtime LIMIT 10
```

Perform aggregations on the stream for reporting

Stop the Dataflow Job (to free up resources for your project)

Create a real-time dashboard

Create a time series dashboard

Summary:

used Pub/Sub to collect streaming data messages from taxis and feed it through your Dataflow pipeline into BigQuery

Pub/Sub, which can be used to ingest a large volume of IoT data from diverse resources in various formats.

Dataflow, a serverless, NoOps service, to process the data. 'Process' here refers to ETL (extract, transform, and load).

Big data with BigQuery:

BigQuery's two main services, storage and analytics

BigQuery is a fully managed data warehouse

A data warehouse is a large store, containing terabytes and petabytes of data gathered from a wide range of sources within an organization, that's used to g

A **data lake** is just a pool of **raw, unorganized, and unclassified** data, which has **no specified purpose**. A data warehouse on the other hand, contains structu

used for **advanced querying**

Being **fully managed** means that BigQuery **takes care of the underlying infrastructure**, so you can focus on using SQL queries to answer business questions, scalability, and security.

BigQuery Storage: to store petabytes of data. **1 petabyte** is equivalent to **11,000 movies at 4k** quality.

BigQuery analytics: **machine learning** (using SQL), **geospatial analysis**, and **business intelligence**,

BigQuery is a fully managed serverless solution, meaning that you don't need to worry about provisioning any resources or managing servers in the backend to answer your organization's questions in the frontend.

By **encryption at rest**, we mean encryption used to protect data that is stored on a **disk**, including solid-state drives, or backup media.

data warehouse solution architecture:

input data can be either real-time or batch data

streaming data, which can be either structured or unstructured, high speed, and large volume, Pub/Sub is needed to digest the data.

If it's batch data, it can be directly uploaded to Cloud Storage.

both pipelines (real-time or batch data) lead to Dataflow to process the data (ETL)

(real-time+Pub/Sub)(batch data+Cloud Storage)+Dataflow (ETL)+BigQuery(analytics, AI, and ML)+(business intelligence(BI) tools)(AI/ML tools)

The job of the analytics engine of BigQuery at the end of a data pipeline is to ingest all the processed data after ETL, store and analyze it, and possibly c

visualization and machine learning.

business analyst, BI developers: If you prefer to work in spreadsheets, you can query both small or large BigQuery datasets directly from **Google Sheet** operations like pivot tables.

data scientist or machine learning engineer: you can directly call the data from BigQuery through **AutoML** or **Workbench** (parts of Vertex AI)

BigQuery is like a common staging area for data analytics workloads

Storage and analytics

BigQuery :**fully managed storage facility to load and store datasets**, and also a **fast SQL-based analytical engine**

The two services are connected by **Google's high-speed internal network**.

BigQuery can ingest datasets from : **internal** data(data saved directly in BigQuery), **external** data, **multi-cloud** data (data stored in **multiple cloud services**, s

datasets.

After the data is stored in BigQuery, it's fully managed by BigQuery and is **automatically replicated, backed up, and set up to autoscale**

BigQuery also offers the option to **query external data sources**—like data stored in other **Google Cloud storage** services like **Cloud Storage**, or in other **Googl**

Spanner or Cloud SQL

That means a raw CSV file in Cloud Storage or a Google Sheet can be used to write a query **without being ingested by BigQuery first**

inconsistency might result from **saving and processing data separately**. To avoid that risk, consider using **Dataflow** to build a **streaming data pipeline** into l

patterns to load data into BigQuery:

batch load, where source data is loaded into a BigQuery table in a single batch operation. (**one-time** operation or it can be **automated to occur on a sc**

A batch load operation can create a **new** table or **append** data into an existing table.

streaming, where **smaller batches of data are streamed continuously** so that the data is available for **querying in near-real time**

generated data, where SQL statements are used to insert rows into an existing table or to write the results of a query to a table.

BigQuery is optimized for running analytic queries over large datasets.

It can perform queries on **terabytes of data in seconds and on petabytes in minutes.**

BigQuery analytics features:

Ad hoc analysis using Standard SQL, the **BigQuery SQL** dialect.

Geospatial analytics using geography data types and Standard SQL geography functions.

Building **machine learning models** using **BigQuery ML**.

Building rich, interactive **business intelligence dashboards** using **BigQuery BI Engine**.

Query types:

BigQuery runs **interactive** queries, which means that the **queries are executed as needed**

batch queries, where **each query is queued** on your behalf and the query starts **when idle resources are available**, usually within a few minutes.

BigQuery demo - San Francisco bike share

- o If no **hyphen** in the project name, no need to **back ticks** around the project name
- o hold down the command key or the **Windows** key it'll highlight all the data sets in your query, so you can click on it
- o Click on Query Table. Click on field names from Schema to add them into Query editor (no need to write names)
- o More>Format
- o Run
- SELECT, FROM, AS, GROUP BY, ORDER BY, WHERE, DESC, LIMIT
- Explore In Data Studio

data definition language (DDL): I want a creation statement inside of the actual code itself

Create dataset

CREATE OR REPLACE **TABLE** <dataset name>.<table name> AS

CREATE OR REPLACE **VIEW** <dataset name>.<view name> AS : If the original public dataset updates, the derived table will update

Introduction to BigQuery ML:

Traditional:

export data from datastore into an IDE (integrated development environment) such as **Jupyter Notebook** or **Google Colab**
transform the data and perform all your feature engineering steps before you can feed it into a training model.
build the model in TensorFlow, or a similar library, and train it **locally** on your computer or on a **virtual machine**.
 To **improve the model performance**, you also need to go back and forth to **get more data and create new features**.

BigQuery:

- **Create a model** with a SQL statement
 - Write a **SQL prediction query** and invoke ml.Predict
 - evaluating the model,
- Hyperparameters** are the settings applied to a model **before the training starts**, like the learning rate
 Choosing which type of ML model depends on your **business goal and the datasets**.

Supervised models are task-driven and identify a goal.

Goal: classification: logistic regression

goal :predict a number (regression): linear regression

unsupervised models are data-driven and identify a pattern

goal: identify patterns or clusters: cluster analysis

We recommend that you start with these options (logistic/linear regression), and use the results to **benchmark** to compare against more complex networks (XGBoost, AutoML Tables, wide and deep DNNs), which may take more time and computing resources to train and deploy machine learning operations (**ML Ops**): BigQuery ML supports features to **deploy, monitor, and manage the ML production**

Importing TensorFlow models for batch prediction

Exporting models from BigQuery ML for online prediction

hyperparameter tuning using Vertex AI Vizier

ML development: upload data, engineer feature, train model, evaluate result

Using BigQuery ML to predict customer lifetime value (LTV):

LTV: to estimate how much revenue or profit you can expect from a customer given their history and customers with similar patterns.

a **record or row** in the dataset is called an **example, an observation, or an instance**.

A **label** is a correct answer, and you know it's correct because it comes from historical data.

Depending on what you want to predict, a label can be either a **numeric** variable, which requires a **linear regression** model, or a **categorical** variable, w model.

Those columns are called **features**, or at least potential features.

Understanding the quality of the data in each column and working with teams to **get more features or more history** is often the hardest part of any ML project
 feature engineering: **combine or transform feature** columns

BigQuery ML **automatically does one-hot encoding** for categorical values.

BigQuery ML **automatically splits the dataset into training data and evaluation data**

BigQuery ML project phases:

phase 1: you **extract, transform, and load** data into BigQuery, if it isn't there already.

If you're already using other Google products, like YouTube for example, look out for easy **connectors** to get that data into BigQuery before you build y

You can **enrich your existing data warehouse with other data sources** by using **SQL joins**.

phase 2: you **select** and **preprocess** features.

You can use **SQL** to create the training dataset for the model to learn from

phase 3: you **create the model inside BigQuery**.

This is done by using the **"CREATE MODEL"** command.

Give it a **name**, specify the **model type**, and pass in a SQL query with your training dataset.

phase 4: after your model is trained, you can execute an **ML.EVALUATE** query to evaluate the performance of the trained model on your evaluation dataset.

phase 5: use it to **make predictions**.

invoke the **ml.PREDICT** command on your newly trained model to return with predictions and the model's confidence in those predictions.

BigQuery ML key commands:

CREATE MODEL: create a model. Models have **OPTIONS** (model type), which you can specify.

CREATE OR REPLACE MODEL: to overwrite an existing model

ML.WEIGHTS: inspect what a model learned

That value indicates **how important the feature is for predicting the result**, or label.

ML.EVALUATE: To evaluate the model's performance

ML.PREDICT: to make batch predictions

BigQuery ML commands for supervised models:

Labels: you need a field in your training dataset titled **LABEL**, or you need to specify which field or fields are your labels using the **input_label_cols** in y

Feature: features are the data columns that are part of your **SELECT** statement after your **CREATE MODEL** statement.

After a model is trained, you can use the **ML.FEATURE_INFO** command to get **statistics and metrics** about that column for additional analysis.

model object: created in BigQuery that **resides in your BigQuery dataset**. You train many different models, which will all be objects stored under your **tables and views**. Model objects can display information for **when it was last updated** or **how many training runs** it completed

Model types (regression/classification): Creating a new model is as easy as writing **CREATE MODEL**, choosing a type, and passing in a training dataset.

Training progress: While the model is running, and even after it's complete, you can view training progress with **ML.TRAINING_INFO**.

inspect weights (**ML.WEIGHTS**): to see what the model learned about the importance of each feature as it relates to the label you're predicting.

ML.EVALUATE: how well the model performed against its evaluation dataset.

ML.PREDICT: getting predictions (through referencing your model name and prediction dataset)

Predicting Visitor Purchases with a Classification Model with BigQuery ML:

BQML: a feature in BigQuery where data analysts can **create, train, evaluate, and predict** with machine learning models with **minimal coding**

Create a BigQuery dataset to store models

the web analytics dataset has nested and repeated fields like **ARRAYS** which **need to be broken apart into separate rows** in your dataset. This is accomplish

Machine learning options on google cloud:

Smart Reply: Gmail automatically suggests three responses to a received message. it uses artificial intelligence (natural language processing) to predict how you r

The goal is to enable every company to be an AI company by **reducing the challenges of AI model creation** to only the steps that require **human judgment or cre**

Options to build ML models:

- a. **BigQuery ML**: uses SQL queries to create and execute machine learning models in BigQuery
- b. **pre-built APIs**: lets you **leverage machine learning models that have already been built and trained by Google**, so you don't have to build your own machine learning models or have enough training data or sufficient machine learning expertise in-house.
- c. **AutoML**: is a **no-code solution**, so you can **build your own machine learning models** on **Vertex AI** through a **point-and-click interface**.
- d. **custom training**: through which you can **code your very own machine learning environment, the training, and the deployment**, which gives you flexibility in your machine learning pipeline.

Differences between options:

Data type: BigQuery ML only supports tabular data while the other three support tabular, image, text, and video

Training data size: Pre-built APIs do not require any training data, while BigQuery ML and custom training require a large amount of data

Machine learning and coding expertise: Pre-Built APIs and AutoML are user friendly with low requirements, while Custom training has the highest requirement for you to understand SQL.

Flexibility to tune the hyperparameters: At the moment, you can't tune the hyperparameters with Pre-built APIs or AutoML, however, you can experiment with BigQueryML and custom training.

Time to train the model: Pre-built APIs require no time to train a model because they directly use pre-built models from Google. The time to train a model depends on the specific project.

Normally, custom training takes the longest time because it builds the ML model from scratch, unlike AutoML and BigQuery ML.

Which option:

- **BigQuery ML**: data engineers, data scientists, and data analysts are familiar with SQL and already have your data in BigQuery: **BigQuery ML** lets you develop machine learning models.
- **pre-built APIs**: business users or developers with little ML experience: pre-built APIs (vision, video, and natural language)
- **AutoML**: If your developers and data scientists want to **build custom models** with your **own training data** while spending **minimal time coding**, then AutoML provides a code-less solution to enable you to focus on business problems instead of the underlying model architecture and ML provisioning.
- **custom training**: If your ML engineers and data scientists want **full control of ML workflow**, **Vertex AI** custom training lets you train and serve custom machine learning models.

Pre-built APIs:

Good Machine Learning models require **lots of high-quality training data**. You should aim for **hundreds of thousands** of records to train a custom model. If you don't have that much data, pre-built APIs are a great place to start.

Pre-built APIs are offered as services.

They save the time and effort of **building, curating, and training** a new dataset so you can just jump right ahead to predictions

API examples:

The Speech-to-Text API: converts audio to text for data processing.

The Cloud Natural Language API: recognizes **parts of speech** called entities and sentiment.

The Cloud Translation API: converts text from one language to another.

The Text-to-Speech API: converts text into high quality voice audio.

The Vision API: works with and recognizes **content** in static images.

cloud.google.com/vision

The Video Intelligence API: recognizes **motion and action** in video.

You can actually **experiment** with each of the ML APIs in a **browser**.

When you're ready to **build a production model**, you'll need to **pass a JSON object request to the API and parse what it returns**.

AutoML:

training and deploying ML models can be extremely time consuming, because to achieve the best result you need to

repeatedly add new data and features,

try different models, and

tune parameters.

AutoML was first announced in January of 2018, the goal was to **automate machine learning pipelines to save data scientists from manual work**

For AutoML, two technologies are vital:

- **transfer learning**: is a powerful technique that lets people with **smaller datasets, or less computational power**, achieve state-of-the-art results by taking advantage of models that have been trained on **similar, larger data sets**.
Because the model learns via transfer learning, **it doesn't have to learn from scratch**, so it can generally reach higher accuracy with much less data than models that don't use transfer learning.
- **neural architecture search**: to find the **optimal model** for the relevant project.

AutoML platform actually trains and evaluates **multiple models** and compares them to each other

This neural architecture search produces an **ensemble of ML models** and chooses the best one.

It is a no-code solution: minimal effort and requires little machine learning expertise

AutoML provides a tool to **quickly prototype models** and explore new datasets before investing in development.

How AutoML works:

For each data type (image, tabular, text, and video), AutoML solves different types of problems, called **objectives**.

upload your data (from Cloud Storage, BigQuery, or even your local machine) into AutoML.

inform AutoML of the problems you want to solve.

Some problems may sound similar to those mentioned in pre-built APIs. However the major difference is that pre-built APIs use **pre-built machine learning models**, while AutoML uses **custom-built models**.

In AutoML, you use your **own data** to train the machine learning model.

Objectives:

- For image data:
 - You can use a **classification model** to analyze image data and return a list of **content categories** that apply to the image.
 - You can also use an **object detection model** to analyze your image data and return **annotations** that consist of a **label** and **bounding box** for each object in an image.
- For tabular data:
 - You can use a **regression model** to analyze tabular data and return a numeric value.
 - You can use a **classification model** to analyze tabular data and return a list of categories.
 - And a **forecasting model** can use multiple rows of **time-dependent** tabular data from the past to predict a series of numeric values in the future.
- For text data:
 - You can use a **classification model** to analyze text data and return a list of categories that apply to the text found in the data.
 - An **entity extraction model** can be used to inspect text data for known entities referenced in the data and label those entities in the text (e.g., **hashtag**, but created by machine).
 - A **sentiment analysis model** can be used to inspect text data and identify the prevailing emotional opinion within it, especially to determine if it is positive, negative, or neutral.
- For video data:
 - You can use a **classification model** to analyze video data and return a list of **categorized shots and segments**.

- You can use an **object tracking model** to analyze video data and return a list of **shots and segments** where these objects were detected
- And an **action recognition model** can be used to analyze video data and return a **list of categorized actions** with the moments the act

Custom training

Using Vertex AI Workbench

Workbench is a **single development environment for the entire data science workflow**, from exploring, to training, and then deploying a machine learning **environment** you want your ML training code to use: a **pre-built container** (includes dependencies and libraries) or a **custom container (empty: no libraries)**

Vertex AI

Some traditional challenges

determining how to handle **large quantities of data**,
Determining the **right machine learning model** to train the data,
harnessing the required amount of **computing power**.

Production challenges:

scalability,
monitoring,
continuous integration and continuous delivery or deployment

ease-of-use challenges.

Many tools on the market require advanced coding skills,
which can take a data scientist's focus away from model configuration.
no unified workflow.

Google's solution to the above challenges: Vertex AI: a **unified platform** that brings all the components of the **machine learning ecosystem and workflow** to Vertex AI, a tool that **combines the functionality of AutoML**, which is codeless, and **custom training**, which is code-based, **to solve production and ease-of-use challenges**

unified platform: one digital experience to **create, deploy, and manage models over time, and at scale**:

1. **data readiness** stage: users can **upload** data from wherever it's stored— **Cloud Storage, BigQuery, or a local machine**.
2. during the **feature readiness** stage: users can **create features**, which are the **processed data** that will be put into the model, and then share them
3. **Training and Hyperparameter tuning**. This means that when the data is ready, users can **experiment** with different models and **adjust hyperparameters**
4. **deployment and model monitoring**: users can set up the pipeline to **transform the model into production** by **automatically monitoring** and performing

Vertex AI allows users to **build** machine learning models with either **AutoML**, or **Custom Training**

AutoML is easy to use and lets data scientists spend more time **turning business problems into ML solutions**, while custom training enables data scientists to **build custom models** in the **Vertex AI development environment and process**.

Vertex AI properties:

seamless: Vertex AI provides a **smooth** user experience from **uploading and preparing data all the way to model training and production**.
Scalable: The Machine Learning Operations (**MLOps**) provided by Vertex AI helps to **monitor and manage** the ML **production** and therefore **scale the models** **automatically**.
sustainable. All of the artifacts and features created using Vertex AI can be **reused and shared**.
speedy: Vertex AI produces models that have **80% fewer lines of code** than competitors.

AI Solutions:

Google Cloud's artificial intelligence solution portfolio (three layers):

the AI foundation: Google Cloud infrastructure (compute, storage, networking) and data (BQ, Dataflow, Looker).

the AI development platform: AutoML and custom training, which are offered through Vertex AI, and pre-built APIs and BigQuery ML.

AI solutions:

horizontal solutions:

usually apply to any industry that would like to solve the same problem.

Document AI and CCAI

Document AI uses **computer vision** and **optical character recognition**, along with **natural language processing**, to create pretrained models for processing documents

The goal is to **increase the speed and accuracy** of document processing to help organizations make better decisions faster, while **reducing costs** of **Contact Center AI**, or CCAI. The goal of CCAI is to improve **customer service** in contact centers through the use of artificial intelligence.

It can help **automate simple interactions**, assist human agents, **unlock caller insights**, and **provide information to answer customer questions**

vertical/industry solutions:

relevant to specific industries.

Retail Product Discovery, which gives retailers the ability to provide **Google-quality search and recommendations on their own digital products** to **increase conversions** and reduce search abandonment

Healthcare Data Engine, which generates **healthcare insights and analytics** with one **end-to-end solution**

Lending DocAI, which aims to transform the **home loan** experience for borrowers and lenders by **automating mortgage document processing** and **reducing risk**
cloud.google.com/solutions/ai

Summary

The machine learning **Workflow** with **Vertex AI**:

Introduction:

basic differences between machine learning and traditional programming:

Traditional: Data, plus rules—otherwise known as algorithms—lead to answers: **Data+rules(algorithms)=answers**

computer can only follow the algorithms that a human has set up

ML: the algorithms are too complex to figure out

we feed a machine a **large amount of data, along with answers** that we would expect a model to conclude from that data

For machine learning to be successful:

lots of storage, like what's available with **Cloud Storage**,
the ability to make **fast calculations**, like with **cloud computing**.

key stages to this learning process:

- **data preparation**: data uploading and feature engineering

Data types:

real-time streaming data or batch data,
structured (numbers and text normally saved in tables), or unstructured (images and videos).

- **model training**: A model needs a tremendous amount of **iterative** training. This is when **training and evaluation** form a cycle to train a model, then evaluate it, and then train it some more.
- **model serving**: move an ML model into production. the machine learning model is **deployed, monitored, and managed**.

ML workflow isn't linear, it's iterative:

during model training, you may need to **return** to dig into the raw data and **generate more useful features** to feed the model.

When **monitoring** the model during **model serving**, you might find **data drifting**, or the accuracy of your prediction might suddenly drop. You might need to **adjust the model parameters**.

Fortunately, these steps can be **automated with machine learning operations, or MLOps**.

Vertex AI provides many features to support the ML workflow:

- **Feature Store**: provides a **centralized repository for organizing, storing, and serving** features to feed to training models,
- **Vizier**: helps you **tune hyperparameters** in complex machine learning models,
- **Explainable AI**: helps **interpret training performance and model behaviors**,
- **Pipelines**: help you **automate and monitor the ML production**

Data preparation:

AutoML workflow

upload data, feature engineering

When you upload a dataset in the **Vertex AI user interface**, you'll need to provide a **meaningful name** for the data and then select the **data type** (image, tab tagged—or labeled

A label can be **manually** added, or it can be added by using **Google's paid label service via the Vertex console**.

These **human labellers** will manually generate accurate labels for you.

Data can be uploaded from a local source, BigQuery, or Cloud Storage.

After your data is uploaded to AutoML, the next step is preparing the data for model training with feature engineering.

feature

a factor that contributes to the prediction.

an independent variable in statistics

a column in a table.

Preparing features can be both challenging and tedious. To help, Vertex AI has a function called **Feature Store**.

Feature Store: **centralized repository to organize, store, and serve** machine learning features.

Feature Store **aggregates all the different features from different sources** and updates them to make them available from a central repository

Vertex AI automates the feature aggregation to scale the process.

benefits of Vertex AI Feature Store:

features are **shareable** for training or serving tasks.

Features are **managed and served from a central repository**, which helps **maintain consistency** across your organization.

features are **reusable**. This helps save time and reduces duplicative efforts, especially for high-value features.

features are **scalable**. Features automatically scale to provide **low-latency serving**, so you can focus on **developing the logic to create the feature deployment**.

features are **easy to use**.

Model training:

model training, and model evaluation. This process might be iterative.

Artificial intelligence (AI): **anything related to computers mimicking human intelligence**.

Machine learning is a subset of AI that mainly refers to supervised and unsupervised learning.

deep learning, or deep neural networks: a subset of ML that **adds layers in between input data and output** results to make a machine learn at more depth

Supervised learning is **task-driven and identifies a goal**.

classification: predicts a categorical variable

regression model: predicts a continuous number,

Unsupervised learning, however, is **data-driven and identifies a pattern**:

clustering: groups together data points with similar characteristics and assigns them to "clusters,"

association: identifies underlying relationships

dimensionality reduction: reduces the number of dimensions, or features, in a dataset to **improve the efficiency** of a model.

with AutoML and pre-built APIs you **don't need to specify a machine learning model**.

you'll **define your objective**, such as text translation or image detection. Then on the **backend**, **Google will select the best model to meet your business**

With the other two options, BigQuery ML and custom training, you'll **need to specify which model** you want to train your data on and assign something

AutoML, you **don't need to worry about adjusting these hyperparameter knobs** because the tuning happens automatically in the **back end**.

This is largely done by a **neural architecture search**, which finds the best fit model by comparing the performance against thousands of other models

Model evaluation:

Vertex AI provides extensive evaluation metrics:

confusion matrix (recall and precision): specific performance measurement for machine learning classification problems.

false positive/negative : Type 1/2 Error

Precision and recall are often a trade-off:

If the goal is to **catch (as many potential)(all) spam emails as possible (FN->0)**, Gmail may want to prioritize **recall**.

In contrast, if the goal is to **only catch the messages that are definitely spam (FP->0)** without blocking other emails, Gmail may want to prioritize **precision**

based on feature importance:

is displayed through a **bar chart** to illustrate **how each feature contributes to a prediction**.

feature importance is one example of Vertex AI's comprehensive machine learning functionality called **Explainable AI**

Explainable AI is a set of tools and frameworks to help **understand and interpret predictions** made by machine learning models.

Model deployment and monitoring:

model serving: **model deployment, model monitoring**

model management exists throughout this whole workflow to **manage the underlying machine learning infrastructure**

This lets data scientists focus on **what to do**, rather than **how to do it**.

MLOps combines **machine learning development with operations** and applies similar principles from **DevOps** to machine learning models, which includes

MLOps aims (both data and code are constantly evolving in machine learning):

to solve **production challenges related to machine learning**.

building an **integrated machine learning system**

operating it in **production**

This means adopting a process to enable **continuous integration, continuous training, and continuous delivery**.

three options to deploy a machine learning model:

to deploy to an **endpoint**: is best when **immediate results with low latency** are needed, such as **making instant recommendations** based on a user's behavior online.

A model must be **deployed to an endpoint** before that model can be used to serve real-time predictions

to deploy using **batch prediction**:

This option is best when **no immediate response is required**, and **accumulated data** should be processed with a single request.

For example, **sending out new ads** every other week based on the user's recent purchasing behavior and what's currently popular on the market to deploy using **offline prediction**:

This option is best when the model should be deployed in a **specific environment off the cloud**.

The backbone of MLOps on Vertex AI is a tool called **Vertex AI Pipelines**:

It **automates, monitors, and governs** machine learning systems by **orchestrating the workflow** in a **serverless** manner.

With **Vertex AI Workbench**, which is a **notebook tool**, you can **define your own pipeline**

Lab introduction: Predicting loan risk with AutoML:

AutoML requires **at least 1,000 data points** in a dataset.

Vertex AI: Predicting Loan Risk with AutoML:

[Vertex AI](#), the **unified AI platform** on Google Cloud to **train and deploy** a ML model:

Vertex AI offers two options on one platform to build a ML model:

a codeless solution with **AutoML**

a code-based solution with **Custom Training** using Vertex **Workbench**

There are three options to import data in Vertex AI:

- Upload a local file from your computer.
- Select files from Cloud Storage.
- Select data from BigQuery.

For **Budget**, which represents **the number of node hours for training**, enter **1**.

Training your AutoML model for 1 compute hour is **typically a good start for understanding whether there is a relationship between the features and**

The **confidence threshold** determines how a ML model counts the positive cases.

A **higher threshold increases the precision, but decreases recall**. A lower threshold decreases the precision, but increases recall.

You can improve the percentage by **adding more examples (more data), engineering new features, and changing the training method**, etc.

These **feature importance** values could be used to help you improve your model and have more confidence in its predictions:

You might decide to **remove the least important features** next time you train a model or

to **combine two of the more significant features** into a [feature cross](#) to see if this improves model performance.

Feature crosses are mostly used with **linear models** and are rarely used with neural networks.

Now that you have a trained model, the next step is to **create an endpoint in Vertex**.

A model resource in Vertex can have **multiple endpoints** associated with it, and you can **split traffic between endpoints**.

To allow the pipeline to authenticate, and be authorized to **call the endpoint to get the predictions**, you will need to provide your **Bearer Token**.

To use the trained model (get predictions), you will need to **create some environment variables**.

The smproxy application is used to communicate with the backend.

Lab recap: Predicting loan risk with AutoML:

how high (TP, TN) or low (FP, FN) they need to be really depends on the **business goals** you're looking to achieve

to improve the performance of a model:

using a more accurate data source,

using a larger dataset,

choosing a different type of ML model,

tuning the hyperparameters.

Course Summary:

cloud.google.com/training/data-engineering-and-analytics

cloud.google.com/training/machinelearning-ai

cloud.google.com/certifications

How Google Does Machine Learning

Introduction to course and series:

Course series preview

Roles:

machine learning scientists,

machine learning engineers

data scientists,

data engineers and

data analysts

Tools:

Vertex AI platform,

BigQuery ML,

TensorFlow

Keras

machine-learning products and concepts:

Analytics Hub,

Dataplex,

Data Catalog,

predictions

model monitoring,

data quality improvement

exploratory data analysis

what it means to be AI-first :

Introduction:

- build a data strategy around ML,
- identify and solve ML problems,
- infuse your applications with ML.
- What kinds of problems can ML solve?

What is ML?

ML

Machine learning is a way to use standard algorithms to **derive repeated predictive insights from data** and **make repeated decisions**

So machine learning is about **making many predictive decisions** from data

It's about scaling up business intelligence and decision-making

backward-looking use of data: looking at historical data to create reports and dashboards.

AI vs ML: AI is a discipline, ML is a specific way of solving AI problems

The ML model is a **mathematical function**

ML has two stages: **training and inference (prediction)**.

each of the layers of the Neural networks are a simple mathematical function. The entire model therefore consists of a **function of a function of a function** and so on.

Training **deep neural networks**, neural networks with lots of layers, takes a lot of computing power

Google has more than **10,000** plus deep learning models

In practice, you have to build **many ML models to solve the problem**

Avoid the trap of thinking of building **monolithic "one model solves the whole problem"** solutions

Google Photos:

This is the Google product where you can **upload photos from your camera to the cloud**. You don't need to tag it. The **ML software tags images** for you so that **you can then find images**.

Google Translate: lets you point a phone camera at a street sign, and it translates the sign for you:

- One model to find the sign,
- another model to read the sign through OCR, optical character recognition,
- a third model to detect the language,
- a fourth model to translate the sign,
- a fifth model to superimpose translated text,
- perhaps even a sixth model to select the font to use and so on

Gmail's Smart Reply:

This is arguably **the most sophisticated** ML model in production today.

It's a **sequence-to-sequence** model.

What problems can it solve?

ML scales better than the Hand-coded rules, because it's all automated

RankBrain, a deep neural network for search ranking

the system could continually improve itself based on what users actually preferred.

Replacing **heuristic rules** by ML, that's what ML is about

What kinds of problems can you solve with ML?

Anything for which you're writing rules today.

Google is an **AI-first** company: Google thinks of ML as the way to **scale, to automate, to personalise**

Rule-based vs model-based application

- You don't think about **coding up rules**, you think about **training models based on data**.
- You don't think about **fixing bug reports by adding new rules**, you think in terms of **continuously training the model as you get new data**.
- And when you think of **applying rules to inputs**, you think in terms of **deploying models at scale to make predictions**.

Machine learning is about collecting the appropriate data, and then finding the right balance of good learning, trusting the examples

Activity intro: Framing a machine learning problem

Cloud ML use cases:

- Manufacturing
- Retail
- healthcare and life sciences
- travel and hospitality
- financial services
- Energy, feedstock and utilities.

Activity solutions: Framing a machine learning problem:

ML problem:

- what is being predicted?
- what data do we need?

software problem:

- What is the API of the service?
- what does it take?
- who's going to use this service?
- How are they doing it today?

data problem:

- what kind of data do we need to collect?
- what data do we need to analyze?
- what is our reaction? How do we react to a prediction?

Infuse your apps with ML:

An easy way to add ML to your apps is to take advantage of pre-trained models.

Aucnet:

the new machine learning system can **detect the model number of the car** at high accuracy

It can also show the estimated price range for each model

and recognize what part of the car is being photographed

With this system, the car dealers just drag and drop a bunch of unclassified photos and check if the model and parts are classified by the system correctly.

Aucnet built a **custom** image model on Google Cloud platform using TensorFlow

you don't have to do that. There are a variety of domains where Google exposes ML services trained with their own data (Pre-trained (vs custom) ML models):

speech API: transcribe speech

Vision API
 Translation API
 Natural Language API
 Jobs API
 Video Intelligence API

Ocado

is the world's largest **online-only grocery** based in the UK:
 They were able to get **sentiment** entities and **pausing** syntax.
 The computational technology helps Ocado **parse** through the body of e-mails, **tagging** and **routing** to help contact center reps determine the **priority and context**
 Ocado use **parsed results from the NLP API** to **route** customer e-mails.
 They don't want to send you an e-mail. They want to **talk to you interactively** to get their questions or concerns answered.
 we'll be spending more on **conversation interfaces** than even on **mobile apps**

Dialogflow

high level conversational agent tool

Giphy

uses the Vision API to **find the text in memes** using **optical character recognition**.
 The social media company used the Vision API to reject inappropriate uploads.

Uniqlo

designed a shopping chatbot using Dialogflow.

Build a data strategy around ML:

Google Maps

Where the roads are,
 the traffic on each road, bridge closures,
 Traffic and bridge closures are a little more difficult in that you have to work with a bunch of smaller government entities the algorithm itself,
 routing algorithms between A and B subject to a set of constraints.
 You're in a subway station called Roppongi, and Maps tells you that you're on floor number two
 Personalization of the map service is possible only with machine learning.
 Machine learning is about scaling beyond handwritten rules.
 asking for **user feedback to keep improving the model**

Given the choice between **more data and more complex models**, spend your energy collecting **more data**

That means collecting **not just more quantity, but also more variety**

An ML strategy is first and foremost a **data strategy**.

several reasons why you want to go through manual data analysis to machine learning:

if you're doing manual data analysis, you probably **have the data already**.

Collecting data is often the **longest and hardest** part of an ML project and the one most likely to fail.

Manual analysis helps you **fail fast and try new ideas in a more agile way**, so don't skip the analysis step.

to build a good ML model, you have to **know your data**.

Since that's the first step, go through the process of doing manual data analysis, don't **jump straight** to ML.

ML is a journey towards **automation and scale**.

training-serving skew:

This is where you had a certain system (**batch processing system**) for processing **historical** data so that you could **train** it

Then you have a different system that needs to **use the ML model during prediction**.

The system that serves these predictions is probably written in something that your **production engineering team** writes and maintains. Perhaps it's written in **Java** using **Web Frameworks**.

training-serving skew: The problem is that unless the model sees the exact same data in **serving** as was used to do **training**, the model predictions are going to be off

One way to reduce the chances of this is to take **the same code that was used to process historical data during training** and **reuse** it during **predictions**, but for that to happen, your **data pipelines have to process both batch and stream**.

This is a key insight behind **cloud dataflow**

a way to offer **data pipelines in Python, Java** or even visually with Cloud Dataprep.

It's **open-sourced as Apache Beam**, where the B stands for **batch** and -eam stands for **stream**.

A single system to do both batch and stream

in machine learning, it's helpful to use **the same system in both training and prediction**

During training, the key performance aspect you care about is **scaling to a lot of data (distributed training)**

During prediction, though, the key performance aspect is **speed of response**, high **QPS**.

This is a key insight behind **TensorFlow**.

the magic of ML comes with **quantity not complexity**.

If you're building many ML models and planning for many more that you may never build, you want to have an environment where you **fail fast**. The idea is that **if you're failing fast, you get the ability to iterate**.

Quiz:

ML training phase:

Evaluating the models

Data management

Create the models

~~Connecting Neural Networks~~

to replace user input by machine learning?

~~Neural networks~~

~~All options are correct~~

~~labeled data~~

Pre-trained models.

best practices for Data preparation

Avoid target leakage

Provide a time signal

Avoid training-serving skew

How Google Does ML:

machine learning,

we mean the process by which one computer writes a computer program (rule) to accomplish a task. which the best program is by only looking at a set of examples.

normal software engineering,

We have a human who analyzes the problem, writes a bunch of code, and then this code becomes a program that can translate inputs to outputs.

Fully end-to-end ML effort allocation:

defining the **KPI**,
 what you should even be trying to accomplish,
 collecting data,
 building the infrastructure,
optimizing the ML algorithm itself (the most),
 integrating with the rest of the pre-existing systems at your organization.

The secret sauce

Top 10 ML pitfalls:

ML requires just **as much software infrastructure**

you thought training your own ML algorithm would be faster than writing the software.

And the reason is that to make a great ML system, beyond just the algorithm, you're going to need lots of things around the algorithm. like a whole software stack to serve, to make sure that it's robust, and it's scalable, and has great uptime.

But then, if you try to use an algorithm, you put in additional complexities around **data collection, training**, and all of that just gets a little bit more complicated.

No data collected yet

If there's not someone in your organization who's **regularly reviewing that data** or generating reports or new insights, if that data isn't generating value already, likely, the effort to maintain it isn't being put in.

keep **humans in the loop**:

they're reviewing the data,
 handling cases the ML didn't handle very well,
 curating its training inputs.

You launched a product whose initial value prop was its ML algorithm instead of some other feature.

ML system, and it just happens to **optimize for the wrong thing**

if you forget to measure if your ML algorithm is actually **improving** things in the real world?

you confuse the ease of use and the value add of somebody else's **pre trained** ML algorithm with **building your own**

ML algorithms are going to be **retrained many times**

And in fact, all the algorithms we have to address these are **very highly tuned from decades of academic research**. And you should almost always take one off the shelf already made, or already kind of defined, instead of trying to do your own research, as that is very expensive.

ML and business processes:

business processes: any set of activities a company must do directly or indirectly to serve customers.

almost every one of them has a feedback loop.

General feedback loop:

Input-process-outputs-insight generation-tuning-process

The path to ML

Input-process(individual contributor,delegation,digitization)-outputs-insight generation(big data and analytics)-tuning(ML)-process

individual contributor

A task or business process that's in the individual contributor phase is performed by a single person.

The task is not parallelized or scaled at all and is usually very informal.

delegation

multiple people who are all performing the same task in parallel

there's some repeatability in the task

Digitization

we take the **core repeatable** part of the task or a business process, and we **automate** it with computers.

We want to give our users a higher quality of service

it involves so much upfront investment,

Big Data and analytics

we're going to use a lot of data to **build operational and user insights**.

ML:

Here we use all the data from the previous step.

We'll automatically start to improve these computer processes.

A closer look at the path to ML

individual contributor

Dangers of Skipping:

Inability to scale

incorrect assumptions that are hard to change later.

Dangers of Linging

Imagine that you've got one person who's very skilled at their job but then leaves the company or retires.

All that organizational knowledges leaves with them. It's a problem when no one else can perform that process.

Also in this phase you can't scale up the process to meet a sudden increase in demand.

delegation

Dangers of Skipping:

you're never forced to formalize the business process and to define success.

Human responses have an inherent diversity

great ML systems will need humans in the loop

If your ML system is very important, it should be reviewed by humans

Dangers of Linging

you're paying a very high marginal cost to serve each user

The more voices you have in your organization, automation is less possible.

This creates a kind of **organizational lock-in** because you have too many **stakeholders**

Digitization

Dangers of Skipping:

you'll need all the infrastructure of this step to be able to serve your ML at scale

you might start to untangle an IT project, which we may call software, with an ML project. If either one of them fails, the whole project fails.

Dangers of Linging

the other members of your industry are collecting data and tuning their offers and operations from these new insights

Big Data and analytics

Dangers of Skipping:

- you won't be able to train your ML algorithm because your data isn't clean and organized
- you can't build a measure of success

Dangers of Linger

- limiting the **complexity** of problems you can solve and the **speed** at which you can solve them

End of phases deep dive

Almost every single one has a team of people reviewing the algorithms, reviewing their responses and doing random sub-samples and it generates a lot of value for the organization, for customers and for end users.

facets that differentiate deep learning networks in multilayer networks?

- More complex ways of connecting layers
- Cambrian explosion of computing power to train
- Automatic feature extraction

ML development with Vertex AI

Introduction

To build an ML for production

- identifying a goal,
- acquiring, exploring, and preparing data,
- building,
- training,
- evaluating the model.
- Deployment (**web client** that can **request** predictions from the model)
- monitored
- maintained.

the proof of concept, or experimentation phase: the process of determining whether the model is **ready for production**

Moving from experimentation to production

ML development during the experimentation phase:

- framing the problem
 - you identify your use case,**
 - questions typically asked are what you're trying to do? What are the minimum requirements of your business application?
- preparing the data,
 - you might use a subset of a larger data set.**
 - You would also perform **EDA** or exploratory data analysis and seek to improve data quality
 - You'd also consider combining features to create a new feature (**Feature engineering**)
- experimenting,
 - you experiment with **different models** to compare performance metrics.
- evaluating the model
 - recall, precision, an F1 score, or cross-entropy**

ML practitioners train models using

- different architectures,
 - CNNs**, or convolutional neural networks are used for image classification, object detection, and recommender systems.
 - RNNs** or recurrent neural networks are used for sequence modeling, next word prediction, translating sounds to words and human language translation.
 - Sorting and clustering architectures are used for anomaly detection, and pattern recognition.
 - GANS**, or generative adversarial networks are used for anomaly detection, pattern recognition, cybersecurity, self-driving cars, and reinforced learning.
- different input datasets,
 - numerical data sets,
 - bivariate data sets,
 - multivariate data sets,
 - categorical data sets,
 - correlation data sets
- different hyperparameters
 - learning rate,
 - number of layers,
 - num_estimators,
 - max_depth
- different hardware
 - CPUs,
 - GPUs,
 - TPUs

Moving from experimentation to production requires:

- packaging,
 - As you package your code for production, you need to **build and install a Python package** out of your predictive model in Python.
- deploying,
 - A trained module can be deployed in various ways, such as on a **user's mobile device**, or as a **web service in a container running on a cluster**.
 - Deploying the model also requires you to **set up endpoints** which allow your app to serve predictions.
 - a machine learning model can have a **web app front end**. a **request** is sent by a **web client** using a **REST API**, a prediction made by the model is returned.
- monitoring your model
 - model stale:** the underlying data distribution may have shifted over time
 - the model may have been **misconfigured** in its production deployment
 - Then, a **model retraining deployment pipeline** can be triggered
 - Monitoring measures key model performance metrics, and includes
 - model drift,

model performance,
model outliers
data quality

to build a custom model:

you need to know an **ML framework** such as **TensorFlow and Keras**.
You need to know how to **upload** the model to **Google storage** using code,
how to **host** a model on an **AI platform**.
You also need to know how to create a **service account** to **access** the model,
how to **wrap the app in a Docker container**,
how to **push the Docker container to the Google Cloud registry (GCR)**.
You then need to know how to **deploy** the cloud registry container to **App Engine** to serve predictions.
you need to enable **monitoring** of the app to assure **model stability**.

You could build above components **separately or in a pipeline**. But a **unified platform** could offer a more efficient way to achieve your objective or goal

What is there to unify:

Dataset: We **create** datasets by **ingesting** data, **analyzing** the data and **cleaning it up (ETL, ELT)**.

Model:

model **training**. This includes experimentation, hypothesis testing, and hyperparameter tuning.
The trained model is **versioned** and **rebuilt** when there's new data on a schedule, or when the code changes (**ML Ops**).
The model is **evaluated** and compared to existing model versions.
The model is **deployed** and used for **online and batch** predictions.

Vertex AI provides unified definitions and implementations (unified set of APIs for the ML lifecycle) of **four** concepts.

A **data set** can be structured or unstructured, it has **managed metadata**, including **annotations**, and can be stored anywhere on Google Cloud.

A **training pipeline** is a **series of containerized steps** that can be used to train an ML model using a data set, the containerization helps with **generalization, reproducibility, and auditability**.

A **model** is an ML model with **metadata** that was built with a **training pipeline, or directly loaded** only if it's in a compatible format.

And an **endpoint** can be invoked by users for online predictions and explanations.

Vertex AI to manage the following stages in the ML workflow:

Create a data set
upload data.
Train an ML model on your data, train the model,
evaluate model accuracy,
tune hyperparameters and 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 **prediction traffic split** in your endpoint,
manage your models and endpoints.

choose a training method:

AutoML

AutoML lets you create and train a model with **minimal technical effort**.
Even if you want the flexibility of a custom training application, you can use AutoML to **quickly prototype** models, and **explore new datasets** before investing in development.

Custom training

lets you create a training application **optimized for your targeted outcome**.
You have **complete control over training** application functionality.
You can target any **objective**, use any **algorithm**, develop your own **loss functions** or **metrics** or do any other customization

Vertex AI offers

fast experimentation,
accelerated deployment
simplified model management

Components of Vertex AI

Vertex AI dashboard:

Datasets:

After you load data into Vertex AI, whether it's from cloud storage or BigQuery, it's **managed** by Vertex AI.
This means **it can be linked to a model**.

Features:

Vertex AI Feature Store is a **fully managed repository** where you can **ingest, serve, and share** ML feature values within your organization.
Vertex AI Feature Store **manages all of the underlying infrastructure** for you.
For example, it provides **storage and compute resources** for you and can easily scan as needed

labeling tasks

let you request **human labeling** for a dataset that you plan to use to train the custom machine learning model.

Workbench

is a **Jupyter notebook based development environment** for the entire **data science workflow**.
Vertex AI Workbench lets you **access data, process data in a Dataproc cluster, train a model, share** your results, and more.
All without leaving the Jupyter lab interface.

Pipelines

helps you to **automate, monitor, and govern your ML systems** by orchestrating your ML workflow in a **serverless** manner, and storing your **workflow's artifacts** using Vertex ML **metadata**.
By storing the artifacts of your ML workflow in Vertex ML metadata, you can analyze the **lineage** of your workflow's artifacts.
For example, an ML model's lineage may include the **training data, hyperparameters, and code** that were used to create the model.
The key takeaways are that, pipelines allow you to automate, monitor, and **experiment with interdependent parts of an ML workflow**.
ML pipelines are **portable, scalable, and based on containers** and each individual part of your pipeline workflow for example creating a dataset or training a model, **is defined by code**.
This code is referred to as a **component**.
Each instance of a component is called a **step**

Training

You can train models on Vertex AI, using **AutoML**, or use **custom training** if you need the wider range of **customization** options available in AI platform training.

In custom training you can select many different machine types to power your training jobs, enable **distributed training**, use **hyperparameter tuning**, and **accelerate with GPUs**.

Experiments

includes Vertex **Vizier**, which is an **optimization service** that helps you **tune hyperparameters** in complex machine-learning models.

You can **run different studies and compare them using TensorBoard**.

Models

built from the **dataset or unmanaged data sources**.

Many different types of machine learning models are available in Vertex AI, depending on your **use case** and **level of experience with machine learning**.

Managing and deploying models manually can involve writing an **application or framework** to load the model and serve the inferences.

The application might also need to handle pre or post-processing steps and the incoming traffic.

Vertex AI's model resources help to **manage your model on Google Cloud** including **deploying, generating predictions and hyperparameter tuning**.

Vertex AI models can handle both AutoML models and custom trained models.

Endpoints:

Vertex AI lets you **deploy a trained model to an endpoint** for **serving predictions**.

Models can be deployed in Vertex AI, **whether or not the model was trained on Vertex AI**.

Batch prediction

intakes a **group of prediction requests** and outputs the results to a specified location.

Use batch prediction when you **don't require an immediate response**, and want to **process accumulated data with a single request**.

Metadata

stores **artifact** and **metadata for pipelines run using Vertex AI pipelines**.

Each pipeline run produces metadata and ML artifacts, such as the **training, test, and evaluation data** used to create the model.

The **hyperparameters** used during model training, and the **code** that was used to train the model.

It also includes the metadata recorded from the training and evaluation process, such as the model's **accuracy** and artifacts that descend from this model, such as the results of **batch predictions**.

Lab intro: Using an image dataset to train an AutoML model

create an image classification dataset

import images,

train an AutoML classification model,

deploy a model to an end point

send a prediction.

Lab demo: Using an image dataset to train an AutoML model

be sure that we have the Vertex AI API enabled

create our first managed data set

Using an Image Dataset to Train an AutoML Model

Enable the APIs

In the **Google Cloud Console**, on the **Navigation menu**, click **Vertex AI > Dashboard**.

Click **Enable all recommended API**.

Previously, models trained with **AutoML** and **custom models** were **accessible via separate services**.

The new offering **combines both into a single API**, along with other new products.

You can also **migrate existing projects to Vertex AI**.

Vertex AI includes many different products to support **end-to-end ML workflows**.

Dataset:

These input images are stored in a **public Cloud Storage bucket**.

This publicly accessible bucket also contains a **CSV** file you use for data **import**.

This file has **two columns**: the first column lists an image's **URI in Cloud Storage**, and the second column contains the image's **label**.

Create an image classification dataset and import data

on the **Vertex AI** page, in the navigation pane, click **Dashboard**

In the central pane, click **Create dataset**.

Optional: Specify a name for this dataset.

For **Select a data type and objective**, on the **Image tab**, select **Image classification (Single-label)**.

For **Region**, select **us-central1**.

To create the **empty dataset**, click **Create**.

The **Data import** page opens.

Select **Select import files from Cloud Storage**, and specify the **Cloud Storage URI of the CSV** file with the image location and label data.

When import process is complete, the next page shows all of the images, both labeled and unlabeled, identified for your dataset.

Review imported images

After your dataset is **created** and data is **imported**, use the **Cloud Console** to **review the training images** and begin model training.

After the dataset is imported, the **Browse** tab opens.

You can also access this tab by selecting **Datasets** from the side menu, and then selecting the **annotation set** (set of single-label image annotations) associated with your new dataset.

Train an AutoML image classification model

On the **Browse tab**, you can choose **Train new model** to begin training.

You can also start training by selecting **Models** from the side menu, then selecting **Create**.

1. On the **Vertex AI** page, in the navigation pane, click **Model Registry**.
2. To open the **Train new model** page, click **Create**.
3. Under **Training method**, select the target **Dataset** and **Annotation set** if they are not automatically selected.
4. Select **AutoML**, and then click **Continue**.
5. Optional: Under **Model details**, type a **Model name**.

6. Click **Continue**.
7. Leave the **Explainability** section as default and click **Continue**.
8. Under **Compute and pricing**, for **Budget**, enter **8** maximum node hours.
9. Click **Start training**.

Training takes about **2 hours**. When the model finishes training, it is displayed with a **green checkmark status icon**.

Deploy a model to an endpoint and send a prediction

After your AutoML image classification model training is complete, **use the Google Cloud Console to create an endpoint and deploy your model to the endpoint**.

After your model is deployed to this new endpoint, send an image to the model for label prediction.

Access your trained model to deploy it to a **new or existing endpoint** from the **Models** page.

1. On the **Vertex AI** page, in the navigation pane, click **Model Registry**.
2. Select your trained AutoML model and then click on **Version ID**.
The **Evaluate** tab opens, where you can view model performance metrics.
3. On the **Deploy & Test** tab, click **Deploy to endpoint**.
The **Endpoint options** page opens.
4. Under **Define your endpoint**, select **Create new endpoint**, and for **Endpoint name**, type **hello_automl_image**.
5. Click **Continue**.
6. Under **Model settings**, accept the **Traffic split** of **100%**, and set the **Number of compute nodes** to **1**.
7. Click **Deploy**.

Send a prediction to your model

After the endpoint creation process finishes, you can **send a single image annotation (prediction) request in the Cloud Console**.

In the **Test your model** section of the same **Deploy & test** tab you used to create an endpoint in the previous subtask, click **Upload image**, choose a local image for prediction, and view its predicted label.

Training an AutoML Video Classification Model

Task 4. Deploy a model to make batch predictions

- i. On the **Batch Predict** tab, click **Create Batch Prediction**.
- ii. Provide a batch prediction name.
- iii. For the **Source path**, use **automl-video-demo-data/hmdb_split1_predict.jsonl**
- iv. For the **Destination path** to your bucket, click **Browse**.
- v. Click **Create new bucket**, type **Project_ID**.
- vi. Click **Create**.
- vii. Click **Create new folder**, type **predict_results**.
- viii. Click **Create**, and select the destination path.
- ix. Click **Create**.
- x. You can navigate to **Cloud Storage** to find your bucket name. Results are added to the **predict_results** folder.

View results

When the job is complete, your prediction is displayed on the **Batch predictions** tab.

- i. Click on the prediction in the **Batch prediction** view.
Details of the batch prediction job appear.
- ii. Click the **Export location** link to view the results in your storage bucket.
- iii. To see your results in the UI, click **View results**.

A video appears. From the dropdown menu at the top of the page, you can select other videos you want to see the results for.

Understanding the results

In the results for your video annotation, Vertex AI provides three types of information:

- Labels for the video: This information is on the **Segment** tab below the video on the results page.
- Labels for shots within the video: This information is on the **Shot** tab below the video on the results page.
- Labels for each 1-second interval within the video: This information is on the **Interval** tab below the video on the results page.

If the prediction fails, the results in the list show a red icon on the **Recent Predictions** list.

If only one video in the prediction attempt failed, the results show a green icon in the **Recent Predictions** list. On the results page for that prediction, you can view the results for the videos that Vertex AI has annotated.

Tools to interact with Vertex AI

You can deploy models to the Cloud and **manage your datasets, models, endpoints and jobs** on the Cloud Console.

This option gives you a **user interface** for working with your machine-learning resources.

As part of Google Cloud, your **Vertex AI resources** are connected to useful tools like **Cloud Logging and Cloud Monitoring**.

Tools:

- **client library**: for some languages to help you **make calls to the Vertex AI API**.
The client libraries provide an **optimized developer experience** by using each supported language's **natural conventions and styles**.
Alternatively, you can use the **Google API Client Libraries** to access the Vertex AI API by using other languages such as [Indistinct].
When using the Google API Client Libraries, **you build representations of the resources and objects used by the API**.
This is easier and requires less code than **working directly with HTTP requests**.
For example, Cloud client libraries include **Python, Node.js and Java**.
- The **Vertex AI REST API**: provides **RESTful services for managing jobs, models and endpoints**, and for making predictions with **hosted models on Google Cloud**
- **Deep Learning VM Images**: is a set of virtual machine images **optimized for data science and machine-learning tasks**.
All images come with key ML frameworks and tools preinstalled.
You can use them out of the box on instances with GPUs to accelerate your data processing tasks.
Deep Learning VM Images are available to support many combinations of **framework and processor**.
There are currently images supporting **TensorFlow Enterprise, TensorFlow, PyTorch** and generic high-performance computing with versions for both **CPU-only and GPU-enabled workflows**.
- **Deep Learning Containers** are a set of **Docker containers** with key data science frameworks, libraries and tools preinstalled.

These containers provide you with **performance-optimized consistent environments** that can help you **prototype and implement workflows** quickly.

Quiz: Machine Learning Development with Vertex AI
managed dataset in Vertex AI?

Data loaded into Vertex AI - whether it be from **Google Cloud Storage or BigQuery**. This means, for example, that it can be **linked to a model**.

Machine Learning Development with Vertex Notebooks

Machine Learning Development with Vertex Notebooks

Vertex AI Workbench provides **two Jupyter notebook-based options** for your **ML workflow**.

i. managed notebooks

Managed notebooks instances are **Google-managed environments** with **integrations and features** that help you set up and work in an **end-to-end notebook-based production environment**.
are usually a good choice if you want to use a notebook for **data exploration, analysis or modeling, or as part of an end-to-end data science workflow**.

Managed notebooks instances lets you perform **workflow-oriented tasks without leaving the JupyterLab interface**.

They also have many **integrations and features** for implementing your data science workflow.

a. **Control your hardware and framework** from JupyterLab.

In a managed notebooks instance, your JupyterLab interface is where you determine **what compute resources** -- for example, how many **VCPUs or GPUs** and how much **RAM** -- your code will run on and what **framework** you want to run the code in.

This means you can write your code first and then choose how to run it **without having to leave JupyterLab or restart your instance**.

This makes it easy to **scale your hardware down** for **quick tests** of your code and then **scale it back up** when you need to run your code on more data.

b. **Custom containers**.

Your managed notebooks instance includes many common data science frameworks to choose from such as **TensorFlow** and **PyTorch**, (PySpark, R) but you can also add **custom Docker container images** to your **instance**. Your custom containers are available to use **directly** from the JupyterLab interface alongside the pre-installed frameworks.

Training: Dockerfile, Cloud Build, Container Registry, Vertex Training

c. **Access to data**. Managed notebooks lets you access your data **without leaving the JupyterLab interface**.

In JupyterLab's left sidebar, use the **Cloud Storage extension** to browse data and other files that you have access to.

Also in the left sidebar, use the **BigQuery extension** to browse tables that you have access to, write queries, preview results and load data into your notebook.

d. **Dataproc integration**.

You can **process data quickly** by running a notebook on a **Dataproc cluster**.

After your **cluster is set up**, you can run a notebook file on it **without leaving the JupyterLab interface**.

Automated shutdown for idle instances. To help **manage costs**, you can set your managed notebooks instance to **shut down after being idle for a specific time period**.

ii. **user-managed notebooks**:

Deep Learning VM Images instances:

User-managed notebooks are **Deep Learning VM Images instances** that are **heavily customizable** and are ideal if you need a **lot of control over your environment**.

you select your **machine type and the framework** for your instance when you create it.

You **can change your instance's machine type after creation**, although this requires a **restart** of your instance.

You **can't easily change the framework on your instance**, but you can still make manual modifications like updating software and package versions.

Additionally, because user-managed notebooks instances are exposed as **Compute Engine instances**, you can customize them in the same way that you can customize Compute Engine instances.

Health status monitoring.

User-managed notebooks instances provide several methods for monitoring the health of your notebooks, including a **built-in diagnostic tool**.

For example, the diagnostic tool verifies the status of **core services**, including **Docker and Jupyter**.

It checks whether the **disk space for boot and data disks is used beyond an 85 percent threshold**, and collects instance logs on network information, Docker, Jupyter and proxy service status

Networking and security.

For users who have specific networking and security needs, user-managed notebooks can be the best option.

You can use **VPC Service Controls** to set up a user-managed notebooks instance within a service parameter and implement other built-in networking and security features.

You can also configure user-managed notebooks instances manually to satisfy some specific networking and security needs.

Both options are **pre-packaged with JupyterLab** and have a **pre-installed suite of Deep Learning packages**, including support for the TensorFlow and PyTorch frameworks.

Both options support **GPU accelerators** and the ability to **sync with a GitHub repository**.

And both options are **protected by Google Cloud authentication and authorization**.

Vertex AI Model Builder SDK: Training and Making Predictions on an AutoML Mode

to train and make predictions:

Vertex AI Python client library

gcloud command-line tool

online Cloud Console.

Set up your environment

Enable the Notebooks API

1. In the Google Cloud Console, on the **Navigation menu**, click **APIs & Services > Library**.
2. Search for **Notebooks API** and press enter. Click on the Notebooks API result.
3. If the API is not enabled, you'll see the **Enable** button. Click **Enable** to enable the API.

Enable the Vertex AI API

In the Google Cloud Console, on the **Navigation menu**, click **Vertex AI > Dashboard**, and then click **Enable Vertex AI API**.

Launch a Vertex AI Notebooks instance

- i. In the Google Cloud Console, on the **Navigation Menu**, click **Vertex AI > Workbench**. Select **User-Managed Notebooks**.
- ii. On the Notebook instances page, click **New Notebook > TensorFlow Enterprise > TensorFlow Enterprise 2.6 (with LTS) > Without GPUs**.
- iii. In the **New notebook** instance dialog, confirm the name of the deep learning VM, if you don't want to change the region and zone, leave all settings as they are and then click **Create**. The new VM will take 2-3 minutes to start.
- iv. Click **Open JupyterLab**.
A JupyterLab window will open in a new tab.
- v. You will see "Build recommended" pop up, click **Build**. If you see the build failed, ignore it.

Clone a course repo within your Vertex AI Notebooks instance

To clone the training-data-analyst notebook in your JupyterLab instance:

1. In JupyterLab, to open a new terminal, click the **Terminal** icon.
2. At the command-line prompt, run the following command: **git clone URL**
3. To confirm that you have cloned the repository, double-click on the training-data-analyst directory and ensure that you can see its contents.

Best practices for implementing machine learning on Vertex AI

Best practices for machine learning development

- o best practices for **preparing and storing your data**

tabular/structured data:

store all data in **BigQuery**,

you can also store **intermediate processed data** in **BigQuery**.

For **maximum speed**, it's better to store **materialized data** instead of using **views or sub-queries** for training data.

Use Vertex AI **Feature Store** with structured data.

When you're training a model with **structured data**, irrespective of way you're **training that model**, follow these steps.

- a. Search Vertex AI Feature Store
 - i. to determine whether **existing features satisfy your requirements**.
Open Vertex AI Feature Store and search to see whether a **feature already exists** that relates to your use case or covers the signal that you're interested in passing to the model.
 - ii. If Vertex AI Feature Store contains features that you want to use, **fetch those features** for your training labels using Vertex AI **Feature Store's batch serving** capability.
- b. Create a new feature.
 - i. If Vertex AI Feature Store doesn't have the features you need, create a new feature using data from your **data lake (Cloud storage bucket or BigQuery)**.
 - ii. **Fetch raw data from your data lake** and write your **scripts** to perform the necessary **feature processing and engineering**.
- c. **Join** the feature values you **fetch from Vertex AI Feature Store** and the **new feature values** that you created from the data lake.
Merging those feature values produces the training dataset.
- d. Set up a **periodic job** to compute **updated values of the new feature**.
When you determine that a feature is **useful** when you want to put it into **production**, set up a **regularly scheduled job** with the required cadence to compute updated values of that feature and **ingest** it into Vertex AI Feature Store.
By **adding your new feature to Vertex AI Feature Store**, you automatically have a solution to **online serving** of the features for online prediction use cases, and you can **share** your features with **others in the organization** that may get value from it for their own ML models.

image, video, audio and unstructured

Store data in **large container formats in Cloud Storage**.

This applies to **sharded TFRecord** files if you're using **TensorFlow** or **AVRO** files if you're using any other framework.

Combine many individual images, videos or audio clips **into large files**.

This would **improve your read and write throughput to Cloud Storage**.

Aim for files of at least **100 megabytes** and between 110,000 shards.

To enable **data management**, use **Cloud Storage buckets and directories** to group the shards.

Use **Vertex Data Labeling** for unstructured data.

You might need humans to provide labels to your data, especially with unstructured data.

You can hire your own labels and use Good Cloud's software for managing their work, or you can use Google's **in-house** labels for the task.

Avoid storing data in **block storage** like **network file systems or in virtual machine hard disks**.

Those tools are **harder to manage than Cloud Storage or BigQuery** and often present challenges in shooting performance.

Similarly, avoid reading data directly from **databases like Cloud SQL**.

Instead, store data in **BigQuery in Cloud Storage**

best practices for training a model with Vertex AI:

Training a model within the **Notebooks** instance may be sufficient for **small datasets or for subsets of a larger dataset**.

It may be helpful to use the **training service** for **larger datasets** or for **distributed training**.

Using the Vertex Training service is also recommended to **productionized training**, even on small datasets if the training is carried out on a **schedule or in response to the arrival of additional data**

Vertex AI Training provides a set of **prebuilt algorithms** that allows users to **bring their custom codes to train models**.

This is a **fully managed training service** for users needing **greater flexibility and customization**, or for users running training on premises or another cloud environment.

The workflow presented here shows

training with prebuilt containers using Vertex Training.

The **training application** with your model is **packaged** and then **pushed** to a **Cloud Storage bucket**.

You can then **pull** the package from the bucket and run the **job** on Vertex Training using **prebuilt containers**, which is **simpler** than creating your own **custom Docker image container** for training

best practices for Explainable AI

Vertex Explainable AI is an integral part of the ML implementation process, offering **feature attributions** to provide insights into why models generate predictions.

By **detailing the importance of each features** that the model uses as input to make a prediction, Vertex Explainable AI helps you **better understand your model's behavior and build trust in your models**.

Vertex Explainable AI supports custom-trained models based on tabular and image data.

hyperparameter tuning with Vertex Training

To maximize your model's predictive accuracy, use hyperparameter tuning.

This is an **automated model enhancer** provided by Vertex Training that takes advantage of the **processing infrastructure of Google Cloud** to test different hyperparameter configurations when training your model.

Hyperparameter tuning **removes the need to manually adjust hyperparameters** over the course of numerous training runs to arrive at the **optimal values**

best practices for using Workbench Notebooks

Use Workbench Notebooks to **evaluate and understand your models**.

In addition to built-in common libraries like scikit-learn, Notebooks offers **What-if-Tool, WIT**, and **Language Interpretability Tool, LIT**.

WIT lets you **interactively analyze your models** for **bias** using multiple techniques, and LIT enables you to **understand natural language processing model behavior through a visual, interactive and extensible tool**.

best practices for using **Vertex AI TensorBoard**

When developing models, use Vertex AI TensorBoard **to find and compare specific experiments**.

For example, based on hyperparameters, Vertex AI TensorBoard is an **enterprise-ready managed service**.

It provides a **cost effective, secure** solution that lets **data scientists and ML researchers** collaborate easily by making it seamless to track, compare and share their experiments.

Vertex AI TensorBoard lets you **track experiment metrics, such as loss and accuracy, over time, visualize a model graph, project embeddings to a lower dimensional space** and much more.

Data preprocessing best practices

best practices for data preprocessing

BigQuery: tabular data,

Dataflow: unstructured data

TensorFlow Extended: managed datasets (data linked to your models)

If you're using tabular data, use BigQuery for data **processing and transformation** steps.

When you're working with ML, use **BigQuery ML** in BigQuery.

After your data is **preprocessed** for ML, you may want to consider using a **managed dataset** in Vertex AI.

Managed datasets enable you to create a **clear link between your data and custom-trained models** and provide **descriptive statistics and automatic or manual splitting into train, test, and validation sets**.

Managed datasets **aren't required**.

You may choose **not** to use them if you want **more control over splitting** your data in your training code or if **lineage** between your data and model isn't critical to your application.

With **large volumes of unstructured** data, consider using **Dataflow**, which uses the **Apache beam** programming model.

You can use Dataflow to convert the unstructured data into **binary data formats like TFRecord**, which can improve performance of data **ingestion** during the training process.

If you need to perform transformations that are **not expressible in Cloud SQL**, or are for **streaming**, you can use a **combination of Dataflow and the pandas library**

If you're using TensorFlow for model development, use **TensorFlow Extended** to **prepare your data from training**.

TensorFlow Transform is the TensorFlow component that enables **defining the executing the preprocessing function to transform your data**.

Best practices for machine learning environment setup

o **Workbench Notebooks**

Use notebooks for **experimentation and development** including **writing code, starting jobs, running queries and checking status**.

It's a common practice to **customize Google Cloud properties** like **network, Cloud Identity, Access Management and software for a container associated with a notebook**

Create notebooks instance for **each team member**.

If a team member is involved in multiple projects, especially projects that have **different dependencies**, we recommend using multiple notebooks instances and treating each notebooks instance as a **virtual workspace**

And use Vertex **SDK for Python**

o **security**

Help secure **PII** (personally identifiable information) in notebooks.

take a look at the **notebooks security blueprint protecting PII data guide**, which provides guidance about applying data governance and security policies to help protect your notebooks.

See also the accompanying deployable blueprint in GitHub

o **Data & model**

Store prepared data and your model in the same project.

You can store prepared data in the **Google Cloud project** where your model is stored.

This will give your AI project **access to all of the datasets required** for modeling and will help to ensure that there are no breaks in reproducibility.

However, different parts of your organization might store their data in different projects, and ML models may need to rely on raw data from different projects.

o **Optimize performance and costs.**

responsible AI development

inclusive machine learning

ML fairness (biases that ML can amplify), explainability, privacy and security,

equality of opportunity: (there is an equal chance of a machine learning system correctly classifying an outcome **regardless of sensitive attributes**)

getting the best results out of a machine learning system requires that you truly understand your data

Facets: open-source visualization tool

Human biases lead to biases in ML models

just because something is based on data doesn't automatically make it neutral

Biases:

interaction bias:

Like this recent game where **people were asked** to draw shoes for the computer, **most people drew ones** like this so as more people interacted with the game the computer didn't even recognize these.

Latent bias:

For example, if you were training a computer on what a physicist looks like and you're using pictures of **past** physicists, your algorithm will end up with a latent bias skewing towards men.

selection bias:

Say you're training a model to recognize faces, whether you grab images from the internet or your own photo library, are you making sure to select photos that **represent everyone**?

Examples:

from tackling **offensive or clearly misleading information from appearing at the top of your search results page**, to adding a **feedback tool on the search bar** so people can **flag hateful or inappropriate auto complete suggestions**.

Biases in data

Unconscious biases exist in our data

human biases that exist in **data**: because data found in the world has existing biases with properties like gender, race and sexual orientation.

reporting bias

Subjects only choose to reveal certain aspects about themselves or their opinions.

selection bias

subjects that get into our samples only represent a **privileged** type of user.

human biases in data collection and labeling procedures:

Confirmation bias

only looking for data which may confirm our hypotheses.

Automation bias

appear when the data we use is only the data we can easily automate.

Google has announced seven AI principles (not theoretical concepts but concrete standards)

The challenge we run into when creating a system that is **fair and inclusive** to all is that ML models learn from **existing data collected from the real world**, so an accurate model may **learn** or even **amplify problematic preexisting biases** in the data based on **race, gender, religion** or other characteristics.

a quick checklist for situations where you should watch out for bias-related issues.

Does your use case or product specifically use any of the following data, **biometrics, race, skin color, religion, sexual orientation, socioeconomic status, income, country, location, health, language or dialect**?

Does your use case or product use data that is likely to be **highly correlated with any of these personal characteristics**? For example, **zip code or other geospatial data** is often correlated with **socioeconomic status and income**.

Image and video data can **reveal information about race, gender and age**.

Could your use case or product negatively individuals economic or other important life opportunities?

tools to diagnose **fairness issues in your data, in your labels and in the effects of predictions**:

What-If Tool

access from within **TensorBoard**

visualize inference results,

edit a data point

see how your model performs,

explore the **effects of a single feature**,

arrange examples for similarity,

view confusion matrices

test algorithmic fairness constraints.

Evaluating metrics with inclusion for your ML system

A confusion matrix helps in understanding inclusion and how to introduce inclusion across different types of groups in your data

This is only for classification problems

$FPR = FP / (FP + TN)$

$FNR = FN / (FN + TP)$

$TPR = TP / (TP + FN)$

$Precision = TP / (TP + FP)$

FP = Type I error

FN = Type II error

missing a lot of stuff in exchange for **high precision** or of the limited about of stuff the ML classifies **it's all correct**.

For email spam: FP should be minimised

For blurring identity for privacy: FN should be minimised

Criterion value: thr

equality of opportunity

To **evaluate inclusion** as you're developing and testing your machine learning model the true positive rate is identical between groups

How to find errors in your dataset using Facets

So there's two parts to Facets, **Overview** and **Dive**

Common data issues that can hamper machine-learning

unexpected feature values,

features with high percentages of missing values,

features with unbalanced distributions,

features distribution skew between datasets.

Summary

Summary

Machine learning at Google is about providing a **unified plaorm** for **managed datasets, a feature store, a way to build, train, and deploy** machine learning models **without writing a single line of code**, providing the **ability to label data**, create **Workbench notebooks using frameworks such as TensorFlow, SciKit Learn, Pytorch, R, and others**. Veex AI Plaorm also includes the ability to **train custom models, build component pipelines, and peorm both online and batch predictions**. This course reviews the ve phases of conveying a candidate use case to be driven by machine learning, and considers why it is impoant to not skip the phases. We end with a **recognition of the biases** that machine learning can amplify and how to recognize them.

Module 1: What It Means to be AI-First

Machine learning is a way to use **standard algorithms** to **derive predictive insights** from data and make **repeated decisions**.

AI is a discipline that has to do with the **theory**

and **methods** to build machines that think and

act like humans. ML is a **toolset** that you can use machine learning to solve certain kinds of AI Problem

Given the choice between **more data and more complex** models, spend your energy collecting more data. That means, collecting **not just more quantity, but also more variety**

Module 2: How Google Does Machine Learning

Avoid these ten ML pitfalls:

1. You thought training your own ML algorithm would be faster than writing the software
2. You don't collect enough data
3. You haven't looked at the data but assume it's ready to use
4. You forgot to put and keep humans in the loop
5. Your product launch focused on the ML algorithm
6. You optimized your ML algorithm for the wrong thing
7. You don't know if your ML is improving things in the real world
8. You didn't use pre-trained ML algorithm
9. You only trained your ML algorithm once
10. You designed your own perception or NLP algorithm

Module 3: Machine Learning Development with Veex AI

At a high level, machine learning development addresses **framing the problem, preparing the data, experimenting, and evaluating** the model.

You **build and compare many different models to determine which works best**. For example, random forests, support vector machines, and logistic regression are just three models you could use.

Moving from **experimentation to production** requires **packaging, deploying, and monitoring** your model.

Veex AI provides unified definitions/implementations of four concepts:

1. A dataset can be **structured or unstructured**. It has **managed metadata** including **annotations**, and can be stored anywhere on **Google Cloud**.
2. A **training pipeline** is a series of **containerized steps** that can be used to train an ML model using a dataset. The **containerization** helps with **generalization, reproducibility, and auditability**.
3. A model is an ML model with metadata that was built with a Training Pipeline or directly loaded (only if it is in a compatible format).
4. An endpoint can be invoked by users for online predictions and explanations. It can have one or more models, and one or more versions of those models, with disambiguation carried out based on the request.

Tools to **interact with Veex AI** include **client libraries, VM images, REST API, and containers**.

Module 4: Machine Learning Development with Veex Notebooks

Veex AI Workbench provides two Jupyter notebook-based options for your data science workflow: **managed notebooks and user-managed notebooks**.

Managed notebooks instances are

Google-managed environments with integrations and features that help you set up and work in an **end-to-end notebook-based production environment**. Managed notebooks let you access your data **without leaving the JupyterLab interface**.

User-managed notebooks are **Deep Learning VM Images instances** that are **heavily customizable** and are ideal if you need a lot of control over your environment. User-managed notebooks can be a good choice for users who require extensive customization or who need a lot of control over their environment.

Module 5: Best Practices for Implementing Machine Learning on Veex AI

Google has recommended best practices for:

- Machine learning development
- Data preprocessing
- Machine learning environment setup
- Model deployment and serving
- Model monitoring
- Veex AI Pipeline
- Aifact organization

Machine learning development best practices

Preparing and storing data:

- Regardless of your data's origin, extract data from the source systems and convert the format and storage (separate from the operational source) optimized for ML training.
- **Structured data**:
 - Store tabular data in **BigQuery**
 - Use Veex AI **Feature Store** with structured data
- **Unstructured data**:
 - Store image, video, audio, and unstructured data in **Cloud Storage**
 - Use Veex **Data Labeling** to provide labels

Training a model:

- For **small** datasets, train a model within

the **Notebooks** instance.

- For **large** datasets, distributed training, or scheduled training, use the **Veex training service**.
- Veex AI Training provides a set of pre-built algorithms that allows users to bring their custom code to train models.

Best practices for **Explainable AI**:

- Offers **feature attributions** to provide insights into why models generate predictions.
- Details the importance of each feature that a model uses as input to make a prediction.
- Supports custom-trained models based on tabular and image data.

Best practices for using **Workbench** Notebooks:

- Use Notebooks to evaluate and understand your models. In addition to built-in common libraries like scikit-learn, Notebooks offers **What-if Tool (WIT)** and **Language Interpretability Tool (LIT)**.

Best practices for using Veex AI

TensorBoard:

- Veex AI TensorBoard service lets you **track experiment metrics** such as **loss and accuracy over time**, visualize a model graph, project embeddings to a lower dimensional space, and much more.

Data preprocessing best practices

- Use **BigQuery** to process **tabular** data and use **Dataflow** to process **unstructured** data.

ML environment setup best practices

- Use Notebooks for **experimentation and development**.
- Create a Notebooks instance for **each team member**.
- Help **secure PII** in Notebooks.
- **Store prepared data and your model in the same project**.
- Optimize performance and cost.
- Use Veex **SDK** for **Python**.

Module 6: Responsible AI development

Unconscious biases exist in data. They can exist in both **collecting and labeling data**. These biases will then be reflected in your ML, and affect the entire pipeline.

A checklist for bias-related issues:

- Biometrics • Religion • Income • Health
- Race • Sexual orientation • Country • Language
- Skin color • Socioeconomic status • Location • Dialect

Use the **What-If tool** to help you diagnose **fairness** issues in your data, in your labels, and in the effects of predictions.

A **confusion matrix** helps in understanding **inclusion** and how to introduce inclusion across different kinds of groups across your data.

Equality of opportunity is an approach that strives to **give individuals an equal chance of the desired outcome**. Incorporating this approach into your machine learning system gives you a way to scrutinize your model in order to discover possible areas of concerns. Once you identify opportunities for improvements, you can now make the necessary adjustments to strike a **balance** **between accuracy and non-discrimination**—which, in turn, could make your machine learning model **more inclusive**.

The **Facets** tool can help you make machine learning **more inclusive**.

Launching into Machine Learning automatically gives you a **quick understanding of the distribution of values across the features** of their datasets.

- **Facets** provides an **easy-to-customize, intuitive** interface for exploring the **relationships between the data and the different features of a dataset**.

Those sources can be **streaming** in real time or **batch**.

For example, you may extract data from a Customer Relationship Management system, or **CRM**, to analyze customer behavior.

This data may be **structured** where it is in a given format such as **CSV, a text, JSON, or XML** format.

Or, you may have unstructured source data where you may have **images** of your customers or **text comments** from your chat sessions with your customers.

Or, you may have to extract **streaming** data from your company's transportation vehicles that are equipped with sensors that **transmit data in real time**.

Other examples of unstructured data may include **books and journals, documents, metadata, health records, audio and video**.

data analysis

you analyze the data you've extracted.

For example, you can use **Exploratory Data Analysis, or EDA**, which involves using **graphics** and **basic sample statistics** to get a feeling for **what information might be obtainable** from your dataset.

`Sns.jointplot()`: Bivariate plot+univariate plot in the margins

You look at various aspects of the data such as **outliers or anomalies, trends, and data distributions**, all while attempting to identify those features that can aid in increasing the predictive power of your machine-learning model

Lab:

Df.info(): # of nulls, column name, data type
 Category (string) feature data type is 'object'
 Print(df, 5): prints first and last 5 rows
 Df.describe(): gives summary statistics. For numeric features
 Df.groupby('col name').first(): First entry after grouping
 Df.rename(columns = {old1:new1, old2:new2, ...}): rename the column names

Data preparation

includes data **transformation**, which is the process of **changing or converting the format, structure, or values** of data you've extracted into another format or structure.

There are many ways to **prepare or transform data** from machine-learning model.

For example, you may need to perform **data cleansing** where you need to remove **superfluous and repeated** records from log data.

Or you may need to alter data types where a data feature was **mistyped** and you need to convert it.

Or you may need to **convert categorical data to numerical** data. Most ML models require categorical data to be in numerical format, but some models work with **either numeric or categorical** features while others can **handle mixed** type features.

Df_dummies = Pd.get_dummies(df, drop_first=True) : categorical to OHE

Df = pd.concat([df, Df_dummies], axis = 1) Concat old and new features

Df = df.drop([], axis = 1) Remove old categorical features

To determining the **data quality levels**, organizations typically perform **data asset inventories** in which the **relative accuracy, uniqueness, and validity** of data is measured.

attributes related to data quality.

Data accuracy

relates to whether the data value was stored or an object or the correct values.

Data consistency:

To be correct, data values must be the **right** value and must be **represented in a consistent and unambiguous form**.

Timeliness

can be measured as the **time between when information is expected and when it is readily available for use**.

Data completeness

relates to whether all the intended data being produced in the data set is complete. Or, is any of the data missing?

ways to improve data quality (untidy/messy data).

resolve missing values,

Missing values can **skew** your data

.IsNull().sum()

.IsNull().sum().values.sum() : All missing

.nunique() : # of unique values per column

Df.columns.tolist

Df = Df.apply(lambda x:x.fillna(x.value_counts().index[0])) : 'apply' applies function to the rows or columns of a df

convert date time features to a **date/time format** if it is not in the correct format already,

Df[date] = pd.to_datetime(Df[date], format=)

Df[date].dt.year

Df[date].dt.month

Df[date].dt.day

If the data type of the data column is 'object', use above to convert it to datetime64

parse the data/time features to get **temporal features** that allow you to create more insight in to your data.

We should also consider **parsing the date feature** in to three distinct feature columns: **year, month, and day**.

This would allow you to look at the **seasonality** of your data, to spot **trends**, and to also perform **time series related predictions**.

You can **remove unwanted values** from a feature column.

convert **categorical** feature columns to **one-hot encodings**.

Another data quality issue is unwanted screen characters in a column.

Now the intent of the **less than sign** is valid, the researcher wants to show models less than 2006. But we **cannot leave this less than sign in our feature column**.

There are many ways to deal with this. We could create **year buckets**

Improving data quality can be done **before and after data exploration**.

We can **explore and clean data iteratively**, as you will see the lab. The process does not have to be a **sequential process**.

Data is said to be **messy or untidy** if it is **missing** attribute values, contains **noise or outliers**, has **duplicates**, **wrong data**, upper-lower case column names and is essentially **not ready for ingestion** by a machine-learning algorithm.

Improving Data Quality

Launch Vertex AI Notebooks instance

- In the Google Cloud Console, on the **Navigation Menu**, click **Vertex AI > Workbench**. Select **User-Managed Notebooks**.
- On the Notebook instances page, click **New Notebook > TensorFlow Enterprise > TensorFlow Enterprise 2.6 (with LTS) > Without GPUs**.
- In the **New notebook** instance dialog, confirm the name of the deep learning VM, if you don't want to change the region and zone, leave all settings as they are and then click **Create**. The new VM will take 2-3 minutes to start.
- Click **Open JupyterLab**.
A JupyterLab window will open in a new tab.
- You will see "Build recommended" pop up, click **Build**. If you see the build failed, ignore it.

Clone course repo within your Vertex AI Notebooks instance

- In JupyterLab, to open a new terminal, click the **Terminal** icon.
- At the command-line prompt, run the following command:
- To confirm that you have cloned the repository, double-click on the training-data-analyst directory and ensure that you can see its contents.

The files for all the Jupyter notebook-based labs throughout this course are available in this directory.

Improve data quality

- i. In the notebook interface, navigate to **training-data-analyst > courses > machine_learning > deepdive2 > launching_into_ml > labs**, and open **improve_data_quality.ipynb**.
- ii. In the notebook interface, click **Edit > Clear All Outputs**.
- iii. Carefully read through the notebook instructions and fill in lines marked with #TODO where you need to complete the code as needed.

What is exploratory data analysis

EDA is an approach to analyzing data sets to **summarize their main characteristics**, often with **visual methods**.

A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task.

Exploratory data analysis is a loosely defined term that involves using **graphics and basic sample statistics** such as mean and median or standard deviation to get a feeling for what information might be obtainable from your data set.

EDA is a set of techniques that allows analysts to quickly look at data for trends, outliers and patterns.

The eventual goal of EDA is to obtain theories that can later be tested in the modeling step. Exploratory data analysis is an approach for data analysis that employs a variety of techniques, mostly graphical, **to maximize insight into a data set, uncover underlying structure, extract important variables, detect outliers and anomalies, test underlying assumptions, develop parsimonious models and determine optimal factor settings**.

The three popular data analysis approaches

Classical data analysis:

the data collection is followed by the **imposition of a model, normality, linearity**, for example, and the analysis, **estimation and testing** that follows are focused on the **parameters of that model**.

exploratory data analysis

the data collection is **not followed by a model imposition**.

Rather, it is followed immediately by analysis with a goal of inferring **what model would be appropriate**.

Unlike the classical approach, the exploratory data analysis approach does not impose deterministic or probabilistic models on the data.

On the contrary, the EDA approach allows the data to suggest admissible models that best fit the data.

Bayesian data analysis:

the analyst attempts to answer research questions about **unknown parameters** using **probability** statements based on **prior data**.

They may bring their own **domain knowledge and/or expertise** to the analysis as **new information is obtained**, so that's the purpose of Bayesian analysis, is to **determine posterior probabilities based on prior probabilities and new information**.

Posterior probabilities is a the probability an event will happen after **all evidence or background information** has been taken into account.

Prior probability is the probability an event will happen **before you've taken adding new evidence** into account.

EDA techniques are generally **graphical**.

They include **scatterplots, boxplots, histograms, regplot**, et cetera.

In the real world, **data analysts freely mix elements of all of the above three approaches and other approaches**, as well.

How is EDA used in machine learning

For exploratory data analysis, the focus is on the **data, its structure, outliers and models suggested by the data**

EDA type:

Univariate analysis

It **doesn't** deal with **causes or relationships**, unlike **regression**, and its major purpose is to **describe**.

It takes the data, it **summarizes** that data, and it **finds patterns** in the data.

types

categorical

numerical EDA :Pandas' crosstab function

visual EDA :Seaborn's countplot function.

Continuous

numerical EDA :Pandas' **describe** function,

visual EDA : **boxplots, distribution** plots and kernel density estimation plots, or **KDE** plots in Python, using Matplotlib or using Seaborn.

Bivariate analysis:

to find out if there is a relationship between two sets of values

Types:

Category to category:

One of the most powerful features of Seaborn is the ability to easily build **conditional plots**.

This lets us see **what the data looks like when segmented by one or more variables**.

The easiest way to do this is through the **factor plot** method, which is used to draw a categorical plot up to a facet grid.

Category to continuous:

Seaborn's **jointplot** function draws a plot of two variables with bivariate and univariate graphs.

Continuous to category:

Seaborn's factorplot map method can map a factorplot onto a KDE, distribution or boxplot chart.

`Sns.factorplot().map(sns.kde)`

Data analysis and visualization

the purpose of an EDA is to **find insights** which will serve for **data cleaning, preparation, or transformation**, which will ultimately be used in a machine learning algorithm.

Histogram:

A histogram displays the **shape and spread of continuous sampled data**.

`Series.hist(bins=50)`

Scatter plot

`Plt.scatter()`

heatmap function to show correlations (**multivariate graphical analysis**)

uses a system of **color coding** to represent different values

`Sns.heatmap(Df.corr())`

Exploratory Data Analysis Using Python and BigQuery

Machine Learning in Practice

supervised learning

Rows: examples

Columns: features

Regression: label is continuous

Classification : label is discrete

If data is **not labeled**: **clustering algorithms** to discover interesting properties of the data

Linear regression

decision boundary

Line in higher dimension is hyperplane

Regression: MSE

Classification : cross-entropy

A categorical feature can be embedded into a continuous space.

Both of these problem types, **regression and classification**, can be thought of as **prediction problems** in contrast, to **unsupervised problems**, which are like **description problems**

very common **source of structured data** for machine learning is your **data warehouse**

Logistic regression

In coin-flip if we use linear regression with the **standard mean square error** or loss function, our predictions could end up being **outside the range of zero and one**.

Simple tricks, like **capping the predictions at zero or one**, would introduce **bias**, so we need something else.

Converting this from linear regression to logistic regression can solve this dilemma.

The input into the sigmoid, normally the output of linear regression, is called the **logit**

we are performing a **nonlinear transformation on our linear model**.

Unlike mean squared error, the sigmoid never guesses 1.0 or 0.0 probability. This means that in gradient descent's constant drive to get the loss closer and closer to zero, it will drive the weights closer and closer to plus or minus infinity in the **absence of regularization** which can lead to problems

the output of a sigmoid: It is a **calibrated probability estimate**.

Beyond just the range, the sigmoid function is the **cumulative distribution function of the logistic probability distribution**, whose quantile function is the inverse of the logic which models the long odds. Therefore, mathematically, **the opposite of a sigmoid can be considered probabilities**.

In this way, we can think of calibration as the fact the outputs are real-world values like probabilities. This is in contrast to uncalibrated outputs, like an embedding vector, which is internally informative, but the values have no real correlation.

Lots of output activation functions, in fact, an infinite number, could give you a number between zero and one, but **only this sigmoid is proven to be a calibrated estimate** of the training data set probability of occurrence.

Using this fact about the sigmoid-activation function, we can **cast binary-classification problems into probabilistic problems**.

This paired with a **threshold** can provide a lot **more predictive power** than just a simple binary answer **regularization is important in logistic regression** because driving loss to zero is **difficult and dangerous**.

Due to the equation of the sigmoid, the function **asymptotes to zero when the logit is negative infinity** and to one when the logit is positive infinity. To get the logits to negative or positive infinity, the manager of the weights is increased and increased, leading to **numerical-stability problems, overflows and underflows**.

Also near the asymptotes, as you can see from the graph, the sigmoid function becomes **flatter and flatter**. This means that the **derivative is getting closer and closer to zero**.

Saturation: Since we used the **derivative and back propagation to update the weights**, it is important for the **gradient not to become zero or else training will stop**.

when all activations end up in these plateaus, which leads to a **vanishing-gradient problem** and makes training difficult.

If you use **unregularized logistic regression**, this will lead to **absolute overfitting**, as the model tries to drive **loss to zero** on all **examples** and never gets there, the **weights** for each indicator feature will be driven to **positive infinity or negative infinity**.

Adding regularization to logistic regression helps keep the model simpler by having smaller parameter weights.

This **penalty term added to the loss function** makes sure that **cross entropy** through gradient descent doesn't keep pushing the weights from closer to closer to plus or minus infinity and causing numerical issues. Also with now **smaller logits**, we can now stay in the **less-flat portions of the sigmoid function**, making our **gradients less closer to zero** and thus allowing weight updates and training to continue.

regularization does not transform the outputs into calibrated probability estimate

The great thing about logistic regression is that it already outputs the **calibrated probability estimate**, since the sigmoid function is **accumulated distribution function of the logistic-probability distribution**. This allows us to **actually predict probabilities** instead of just binary answers

To counteract overfitting, we often do both **regularization and early stopping**.

For regularization, **model complexity increases with large weights**, and so as we tune and start to get larger and larger weights for rarer and rarer scenarios, we end up increasing the loss, so we stop.

L2 regularization will keep the **weight values smaller**, and **L1** regularization will keep the models **sparser** by dropping poor features.

To find the **optimal L1 and L2 parameter choices** during **hyper parameter tuning**, you are searching for the point in the **validation-loss function** where you obtain the lowest value.

any **less regularization increases your variants, starts overfitting and hurts generalization**, and any **more regularization increases your bias, starts underfitting and hurts your generalization**.

Early stopping stops training when overfitting begins.

As you train your model, **you should evaluate your model on your validation data set every so many steps, epochs, minutes, et cetera**.

As training continues, **both the training error and the validation error should be decreasing**, but at some point the **validation error might begin to actually increase**.

It is at this point that the **models begin to memorize the training data set and lose its ability to generalize to the validation data set**, and most importantly, to the **new data** that we will eventually want to use this model for.

Using early stopping would stop the model at this point and then **back up and use the weights from the previous step before it hit validation-error-inflection point**.

early stopping is an approximate equivalent of L2 regularization and is often used in its place because it is **computationally cheaper**.

Even though **L2 regularization and early stopping** seem a bit redundant, for real-world systems, you may not quite choose the **optimal hyper parameters**, and thus **early stopping can help fix that choice for you**

A **simple threshold of a binary-classification problem** would be all probabilities less than or equal to 50 percent should be a **no**, and all probabilities greater than 50 percent should be a **yes**.

However, for studying real-world problems, we may want a **different split, like 60/40, 20/80, 99/1**, et cetera, depending on how we want our **balance of our type one and type two errors**, or in other words, our balance of false positives and false negatives.

In ROC, As we **lower the threshold**, we are likely to have **more false positives**, but we'll also **increase the number of true positives** we find.

there's an efficient **sorting-based algorithm** to do this

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

AUC is that it's **scale invariant and classification-threshold invariant**

People sometimes also use AUC (TPR vs FPR) for the **precision-recall curve**, or more recently, **precision-recall-gain curves**, which just use different combinations of the four prediction outcomes as metrics along the axes

However, treating this only as an **aggregate measure** (AUC) can **mask some effects**.

For example, a small improvement in AUC might come by doing a better job of ranking very unlikely negatives as even still yet more unlikely, which is fine, but potentially not materially beneficial.

When we evaluate our logistic-regression models, we need to make sure predictions aren't biased.

there should be an **overall shift in either the positive or negative direction**.

A simple way to **check the prediction bias** is to compare the **average value predictions made by the model over a data set to the average value of the labels** in that data set. If they are not relatively close, then you might have a problem.

even **zero bias alone does not mean everything in your system is perfect**, (but it is a great sanity check)

If you have bias, you could have an **incomplete feature set**, a **buggy pipeline**, a **biased training sample**, et cetera.

calibration scatter plot: log-log scale

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

we're comparing the **bucketized log odds predicted** to the **bucketized log odds observed**

This can happen when parts of the data space is not well represented or because of noise or because of overly strong regularization.

You can bucket by literally **breaking up the target predictions**, or we can bucket by **quantiles**.

Why do we need to bucket prediction to make calibration plots when predicting probabilities? For any given event, the true label is either zero or one. But our prediction values will always be a probabilistic guess, we're always off, but if we group enough examples together, we'd like to see that **on average the sum of the true zeroes and ones is about the same as the mean probability we're predicting**.

important in performing logistic regression:

adding penalty terms to the objective function + early stopping (regularization),

tuned threshold

predictions are unbiased

average of the predictions is very close to the average of observations.

look at slices of data (calibration plot)

Training AutoML Models Using Vertex AI

ML vs DL

All machine learning starts with a business requirement, academic requirement or problem that you are trying to solve.

wrangle

we explored previously when we introduced an untidy dataset and walked you through **making it tidy enough to feed into a machine learning model**.

how to resolve missing values,

convert the date feature column to a **date time format**,

rename a feature column,

remove a value from a feature column,

create one hot coding features

understand temporal feature conversions.

Pipeline:

business understanding

Data wrangling

Data visualisation

Data preprocessing

Model training

Model validation

Deployment (to production)

a point-and-click solution

The data scientist has domain knowledge, but has limited experience putting a machine learning model into production

data analyst knows SQL, but has no ML knowledge

software developer knows Java, no ML knowledge

difference between machine learning and statistics:

In machine learning, you have and want **lots of data**. You'll want to use your dataset for **training, testing and validating**, so you'll have to **split it up**.

In statistics, you **don't need to split** your data.

You take what you are given and you use it.

Also in machine learning, you keep the outliers or those data points that stray outside of all the other data points, and then you build models for them.

You'll want to use those outliers to train your model with so that you **get a more holistic picture of your data**.

Statistics on the other hand is about keeping the data that you have and **getting the best results** out of the data that you have, so it is not uncommon **to toss the outliers out**.

Also, in statistics, we examine the **relationship between variables**.

In machine learning, we want to **predict** the Y given X.

A major difference

data preparation

Machine learning **doesn't require explicit commands to find patterns in data**.

In standard statistics, we need to know **variables and parameters** beforehand

Hypothesis

hypothesis

Also, while you need a hypothesis or theory to test in statistics, there's no **hypothesis testing** required in machine learning.

Type of data

In statistics, your data is **linear**, while in machine learning your data can be **multi-dimensional or non-linear** in nature.

Training

In machine learning, you need to train, meaning the model or algorithm needs to **learn the patterns in the data** and it learns those patterns through training.

Goal

Machine learning is better for **making inferences or predictions**, while statistics is generally better for **testing inferences and hypotheses**.

Scientific question:

The question for statistics is **how and why** something happens, for example, the **relationship or correlation** between the variables. In machine learning, the goal is **what** will happen if I give the algorithm data it has not seen before? Will it make a strong prediction?

ML and DL: Within the **subset of machine learning** methods, **deep learning** is usually implemented as a form of **supervised** learning

Data requirement

Deep learning requires **large datasets**, while machine learning allows you to train on **smaller** datasets.

Accuracy

Because deep learning uses large datasets to glean patterns, it provides **higher accuracy** than other methods

Training time

deep learning **takes longer to train a model**

Hardware dependency

you can train your **ML model** on a **CPU**, while you'll need a **GPU** to train a **deep learning** model given the large data size

hyperparameter tuning

more control over tuning of the hyperparameters with deep learning than with other forms of machine learning.

weight is a parameter. Number of layers is a hyperparameter

machine learning is a subfield of artificial intelligence

The goal of ML is to make computers **learn from the data** that you give them

Instead of writing code that describes the action the computer should take, your code provides an algorithm that **adapts based on examples of intended behavior**

The resulting program consisting of the **algorithm** and associated **learned parameters** is called a **trained model**.

What is automated machine learning?

The process of applying machine learning to real-world problems is time consuming.

Automated Machine Learning workflow = pipeline

traditional components of a machine learning pipeline

You need to get the data ready.

Perform feature engineering.

Train and tune your model.

Serve your model.

Understand it.

Present it to any Edge devices.

Monitor your model and

manage it.

Vertex AI automates the following components in the machine learning pipeline:

data readiness, feature engineering, training and hyperparameter tuning, model serving, explainability and interpretability and the ability to deploy to Edge devices.

+ Vertex AI's Vizier optimization for hyperparameter tuning, managed data sets, feature store

AutoML regression model

XYZ Company has just **defined a business use case**, established the success criteria and wants to deliver an ML model to **production**.

data scientist has **domain knowledge** of the problem and **machine learning experience**, but they have **little experience bringing a model to production**

point and click solution: with Vertex AI, the team at XYZ Company can load data, generate statistics on the data and build and train their model, without writing a single line of code.

Vertex AI, which is a **dashboard with features and services** that allow them to perform various tasks in the machine learning pipeline.

AutoML supports the following data types; image, tabular, text and video.

AutoML tables will **automatically define your problem and model to build**, based on the **data type of your target column**, so if your target column contains **numerical** data, AutoML tables will build a **regression** model. If your target column is **categorical** data, AutoML tables will detect the number of classes and determine if you need to build a **binary or multiclass** model.

Vertex AI is a unified platform

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, and

manage your models and endpoints.

If you want to code, you still can by building a custom solution (Notebooks, Pipelines)

AutoML lets you create and train a model with **minimal technical effort**.

Even if you want the flexibility of a custom training application, you can use AutoML to **quickly prototype** models and explore new datasets **before investing in development**.

Custom training lets you create a training application **optimized for your targeted outcome**. You have **complete control over training application functionality**. You can target any **objective**, use any **algorithm**, develop your own **loss functions or metrics**, or do any other **customization**.

AutoML or Custom training

No data science expertise is required to use AutoML. Custom training requires data science expertise to develop the training application, and also to do some of the data preparation, like feature engineering. AutoML is codeless, so no programming ability is needed. Custom training requires programming experience to develop the training application.

When it comes to training a model, **AutoML saves time** because it requires **less data preparation and no development**. Custom training requires more data preparation and time to develop the training application.

One of the major differentiators is your machine learning **objectives**. With AutoML, you must target one of the predefined objectives, such as regression classification and forecasting, which are supervised learning tasks. There are also predefined objectives for image, text and video. You can use custom training for a variety of objectives.

In terms of optimizing model performance with hyperparameter tuning, AutoML does some automated hyperparameter tuning, but you can't modify the values used. With custom training, you can tune the model during each training run for experimentation and comparison.

If you need more control over aspects of your **training environment**, such as specifying **compute engine type, disk size, machine learning framework, or the number of nodes used for hyperparameter tuning**, then custom training is the best option.

Both AutoML and custom training have the **same limits on managed datasets**, and data size limitations vary, depending on the type of datasets. However, for **unmanaged datasets**, where the data comes from **Google cloud storage or BigQuery** and is **not uploaded as a CSV file**, there is **no limit on data size**, as these are considered unmanaged datasets.

Training an AutoML Classification Model - Structured Data

Previously, models trained with AutoML and custom models were accessible via separate services.

The new offering combines both into a single API, along with other new products.

Vertex AI includes many different products to support **end-to-end ML workflows**

Evaluate AutoML models

Evaluating AutoML models begins with understanding how AutoML Tables uses your dataset.

Your **dataset** will be split into training, validation, and testing sets.

training set

This is the data your model sees during training.

It's used to learn the **parameters** of the model, namely the **weights of the connections between nodes of the neural network**.

validation set

sometimes called the "dev" set, is also used during the **training process**.

After the model learning framework incorporates training data during each iteration of the training process, it uses the model's performance on the validation set to **tune the model's hyperparameters** which are variables that specify the model's structure.

If you try to use the training set to tune the hyperparameters, it's quite likely the model will end up overly focused on your training data and have a **hard time generalizing** to examples that don't exactly match it.

Using a **somewhat novel dataset to fine tune** your model structure, means your model will generalize better.

test set

is **not involved in the training process** at all.

Once the model has completed its training entirely, AutoML Tables uses the test set as an entirely new challenge for your model.

The performance of your model on the test set is intended to give you a pretty good idea of how your model will perform on real-world data.

There's **no perfect answer on how to evaluate your model**.

Evaluation metrics should be considered **in context with your problem type and what you want to achieve with your model**.

MAE

because it uses absolute values, MAE **doesn't consider the relationship's direction** nor indicate **underperformance or overperformance**.

MAPE (Mean absolute percentage error)

RMSE

is **more sensitive to outliers than MAE**. So if you're concerned about large errors, then RMSE can be a more useful metric to evaluate.

RMSLE

The root mean squared **logarithmic** error metric

natural logarithm of the predicted and actual values **plus one**

RMSLE **penalizes under-prediction more heavily than over-prediction**.

R squared

is the **square of the Pearson correlation coefficient** between the observed and predicted values. coefficient of determination

0 <= <=1

Model feature attributions

how much each feature **impacts** a model

Log loss

this is the **cross-entropy between the model predictions and the target values**.

This ranges from zero to infinity, where a **lower value indicates a higher-quality model**.

Mathematically, log loss is the negative average of the log of the corrected predicted probabilities for each instance.

F1

is a useful metric if you're looking for a balance between precision and recall and there's an **uneven class distribution**.

Precision and recall

how well your model is **capturing information and what it's leaving out**

In addition to evaluating models via the **Web UI**, evaluations can be made via **REST** and the **command-line** and using **Python**.

Endpoints are **machine learning models made available for online prediction requests**.

Endpoints are useful for **timely predictions** for many users; for example, in response to an application request.

You can also request **batch predictions** if you **don't need immediate results**.

Batch prediction

is useful for **making many prediction requests at once**.

Batch prediction is **asynchronous**, meaning that the model will **wait until it processes all of the prediction requests before returning a CSV file** or a **BigQuery Table** with prediction values.

Online prediction,

deploy your model to make it available for prediction requests using a **REST API**.

Online prediction is **synchronous** or **real time**, meaning that it will **quickly return a prediction**, but **only accepts one prediction request per API call**.

Online prediction is useful if your **model is part of an application** and **parts of your system are dependent on a quick prediction turnaround**.

BigQuery Machine Learning: Develop ML Models Where Your Data Lives

Vertex AI AutoML data requirements

maximum data set size is 100 gigabytes

1,000 < row < 100M.

target column must be categorical (2-500) or numerical.

2 < column < 1000

Google's BigQuery is **more than just a data warehouse**.

It can provide decision-making guidance through **predictive analytics** by using its machine-learning tool, **BigQuery ML**.

you can **create and train a model without ever exporting data out of BigQuery**

Google BigQuery ML is a **set of SQL extensions** to support machine learning.

to do **custom modeling** without BigQuery ML, there is an increased complexity and multiple tools are required.

Custom modeling also reduces speed, moving and formatting large amounts of data for **Python based ML frameworks takes longer than model training in BigQuery**.

BigQuery ML **speeds up the time to production**, makes **development work much easier**, and **automates** a number of the steps in the **ML workflow**.

BigQuery ML will **import and preprocess data, split data, build the model and deploy the model**.

The team sees that with BigQuery ML all they really have to do is **have the data in BigQuery**, identify a use case, write a short piece of **SQL code**, and **they're ready to deliver their model**.

BigQuery ML is a middle ground between using **pretrained models** and building your **own TensorFlow model with Vertex AI platform**.

Exploratory data analysis or **EDA in a Jupyter Notebook, prototyping and scaling out to a managed service** is a time-consuming process.

From a **logistics** viewpoint, it is also challenging to **manage security and permissions** when using many different tools in the machine learning process.

BigQuery ML:

write a SQL query to **extract** their training data from BigQuery.

create a model specifying model type.

evaluate the model and verify that it meets requirements,

predict using the model on data extracted from BigQuery.

Advantages:

Since the team already has data that lives in BigQuery, they are able to execute their initiatives **without needing to move any data**,

and since one team member is **very familiar with SQL**, development will be **faster**.

And finally, BigQuery ML will **automate** common machine learning **tasks** and perform **hyperparameter tuning**.

BigQuery Machine Learning supported models

Boosted decision trees have better performance than decision trees on extensive datasets

For creating a **recommendation system**, use **Matrix factorization**.

BigQuery ML for time series is popular for **estimating future demands** such as, retail sales, or manufacturing production forecast **automatically detects and corrects for anomalies, seasonality and holiday effects**.

You can use **AutoML Tables** for any **regression classification and time series forecasting** problems, it will **automatically search through various models and find the best one for you**.

Using BigQuery ML to Predict Penguin Weight

This lab introduces **data analysts** to BigQuery ML using SQL queries

The goal is to **democratize** machine learning by enabling **SQL practitioners** to build models using their existing tools and to increase development speed by **eliminating the need for data movement**.

Create your dataset

The first step is to create a BigQuery dataset to **store your ML model**. To create your dataset:

- i. In the Cloud Console, on the **Navigation menu**, click **BigQuery**.
- ii. In the **Explorer** panel, click the **View actions** icon (**three vertical dots**) next to your project ID, and then click **Create dataset**.
- iii. On the Create dataset page:
 - For **Dataset ID**, type **bqml_tutorial**
 - (Optional) For **Data location**, select **us (multiple regions in United States)**.
Currently, the public datasets are stored in the US multi-region [location](#). For simplicity, you should place your dataset in the same location.
- iv. Leave the remaining settings as their defaults, and click **Create Dataset**.

Create and train a linear regression model

```
#standardSQL
```

```
CREATE OR REPLACE MODEL `bqml_tutorial.penguins_model`
```

```
OPTIONS
```

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

```
SELECT
```

```
*
```

```
FROM
```

```
`bigquery-public-data.ml_datasets.penguins`
```

```
WHERE
  body_mass_g IS NOT NULL
```

[CREATE MODEL](#) clause is used to create and train the model
FROM project.dataset.table

To run: **Compose new query > Query editor > Run**

Because the query uses a CREATE MODEL statement to create a table, you do not see query results.

- You can ignore the warning about NULL values for input data.
- Get training statistics

To see the results of the model training, you can use the [ML.TRAINING_INFO](#) function, or you can view the statistics in the Cloud Console

A machine learning algorithm builds a model by **examining many examples and attempting to find a model that minimizes loss**. This process is called *empirical risk minimization*.

Loss is the penalty for a bad prediction: a number indicating how bad the model's prediction was on a single example.

A "[normal equation](#)" optimization strategy is automatically used (optimize_strategy option) for this training (linear regression), so **only one iteration is required to converge to the final model**

see the [CREATE MODEL statement for generalized linear models](#).

see the [BigQuery ML syntax reference for ML.TRAINING_INFO](#)

evaluate the model
#standardSQL

```
SELECT
  *

FROM

ML.EVALUATE(MODEL `bqml_tutorial.penguins_model`,

(
  SELECT
    *
  FROM
    `bigquery-public-data.ml_datasets.penguins`
  WHERE
    body_mass_g IS NOT NULL))
```

You can also call ML.EVALUATE **without providing the input data**. ML.EVALUATE uses the **automatically reserved evaluation dataset**:

When the query is complete, click the **Results** tab below the query text area.

R² score

0 indicates that the model explains none of the **variability of the response** data around the mean.

1 indicates that the model explains all the variability of the response data around the mean.

predict
#standardSQL

```
SELECT
  *

FROM

ML.PREDICT(MODEL `bqml_tutorial.penguins_model`,

(
  SELECT
    *

  FROM

    `bigquery-public-data.ml_datasets.penguins`

  WHERE
    body_mass_g IS NOT NULL

    AND island = "Biscoe"))
```

ML.PREDICT function creates the column predicted_<label_column_name>

XAI

To understand why your model is generating these prediction results, you can use the

[ML.EXPLAIN_PREDICT](#) function

extended version of ML.PREDICT

returns prediction results with additional columns that explain those results

Shapley values and explainable AI in BigQuery ML, see [BigQuery ML explainable AI overview](#).

#standardSQL

```
SELECT
  *

FROM

ML.EXPLAIN_PREDICT(MODEL `bqml_tutorial.penguins_model`,

(
  SELECT
    *
```

```
FROM
  `bigquery-public-data.ml_datasets.penguins`
WHERE
  body_mass_g IS NOT NULL
  AND island = "Biscoe"),
STRUCT(3 as top_k_features))
```

For linear regression models, **Shapley values** are used to generate **feature attribution values** per feature in the model. These attributions are sorted by the absolute value of the attribution in **descending** order.

[ML_EXPLAIN_PREDICT syntax documentation](#)

Globally explain

In order to use `ML.GLOBAL_EXPLAIN`, the model must be **retrained** with the option `ENABLE_GLOBAL_EXPLAIN=TRUE`

```
#standardSQL
```

```
SELECT
  *
```


```
FROM
  ML.GLOBAL_EXPLAIN(MODEL `bqml_tutorial.penguins_model`)
```

Clean up

To avoid **incurring charges** to your Google Cloud account for the **resources** used in this tutorial, either **delete the project** that contains the resources, or **keep the project and delete the individual resources**.

Deleting your dataset

Deleting your project removes all datasets and all tables in the project. If you prefer to **reuse** the project, you can delete the dataset you created in this tutorial:

- i. If necessary, open the BigQuery page in the Cloud Console.
- ii. In the **Explorer** panel, click **View actions** (
 - iii. 
 -) next to your dataset.
- iv. Click **Delete**.
- v. In the Delete dataset dialog box, to confirm the delete command, type **delete** and then click **Delete**.

Deleting your project

To delete the project:

1. In the Cloud Console, on the **Navigation menu**, click **IAM & Admin > Manage Resources**.

Note: If prompted, Click **LEAVE** for unsaved work.
- vi. In the project list, select the project that you want to delete, and then click **Delete**.
- vii. In the dialog, type the project ID, and then click **Shut down** to delete the project.

BigQuery ML hyperparameter tuning

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

vertex vizier

Hyperparameter tuning supports the following model types: linear regression, logistic regression, K means, matrix factorization, boosted tree classifier, boosted tree regressor, DNN classifier and DNN regressor.

a simple deep neural network, or DNN, has three standard hyperparameter options:

data split method set to random,
early stop set to true,
hidden units set to 30, 50.

DNN +HP

num trials equals 10, max parallel trials equals two, dropout HParam range zero, 0.2, optimizer equals HParam candidates, Adam, Adagrad, learn rate HParam candidates 0.01, .01

Check the overview of all 20 trials using the `ML.TRIAL_INFO` function

Using the BigQuery ML Hyperparameter Tuning to Improve Model Performance

The first task is to create a BigQuery dataset to store your training data and ML model.

for **Data location**, select **United States (US)**. Currently, the **public datasets** are stored in the **US multiregional location**. For simplicity, place your dataset in the same [location](#).

Create your training input table

materialize the training input table with 100k rows.

```
CREATE TABLE `bqml_tutorial.taxi_tip_input` AS
```

```
SELECT
  * EXCEPT(tip_amount), tip_amount AS label
```

```
FROM
  `bigquery-public-data.new_york_taxi_trips.tlc_yellow_trips_2018`
```

```
WHERE
  tip_amount IS NOT NULL
```

```
LIMIT 100000
```

Create your model

```
CREATE MODEL `bqml_tutorial.hp_taxi_tip_model`
```

```
OPTIONS
  (model_type='linear_reg',
```

```
num_trials=20,
max_parallel_trials=2) AS
```

```
SELECT
*
```

```
FROM
`bqml_tutorial.taxi_tip_input`
```

The LINEAR_REG model has **two tunable hyperparameters**: `l1_reg` and `l2_reg`.

The previous query uses the **default search space**.

You can also specify the search space explicitly:

```
OPTIONS
(
  ...
  l1_reg=hparam_range(0, 20),
  l2_reg=hparam_candidates([0, 0.1, 1, 10]))
```

- o `hparam_tuning_algorithm`: "VIZIER_DEFAULT"
- o `hparam_tuning_objectives`: ["r2_score"]

`max_parallel_trials` is set to 2 to **accelerate the tuning process**.
the two concurrent trials cannot benefit from each other's training results.

Trial info

To see the **overview of all trials**, including their **hyperparameters, objectives, status, and the optimal trial**, use the [ML.TRIAL_INFO](#) function, and view the result in the Cloud Console after running the SQL:

```
SELECT *
FROM
  ML.TRIAL_INFO(MODEL `bqml_tutorial.hp_taxi_tip_model`)
```

You can run this SQL query **as soon as one trial is done**. If the tuning is stopped in the middle, all already-completed trials will remain available to use.

Evaluate

```
SELECT * FROM ML.EVALUATE(MODEL `bqml_tutorial.hp_taxi_tip_model`)
```

Check the [Data Split](#) section to see the difference between `ML.TRIAL_INFO` objectives and `ML.EVALUATE` evaluation metrics.

Predict

```
SELECT * FROM ML.PREDICT(MODEL `bqml_tutorial.hp_taxi_tip_model`, (
  SELECT * FROM `bqml_tutorial.taxi_tip_input` LIMIT 10))
```

he prediction is made against the **optimal trial by default**.

You can select another trial by specifying the `trial_id` parameter.

```
SELECT
*
FROM
  ML.PREDICT(MODEL `bqml_tutorial.hp_taxi_tip_model`,
    (
      SELECT
      *
      FROM
        `bqml_tutorial.taxi_tip_input`
      LIMIT
        10),
    STRUCT(3 AS trial_id))
```

How to build and deploy a recommendation system with BigQuery ML

Recommendations systems are machine learning systems that help users discover new products and services.

Recommendation systems are all about **personalization**, and have a number of benefits in terms of user **engagement, upselling and cross-selling**.

to improve **conversions, increase clickthrough rates and build customer satisfaction, loyalty and brand affinity**
prepare training data in BigQuery, train a recommendation system with BigQuery ML, and use the predicted recommendations in production.

"implicit feedback": the amount of time a user spends looking at a product

explicit feedback: stars

model type = matrix-factorization

model is trained differently for explicit and implicit feedback.

Now when you run this query, you might get an error if you run it **on demand** in BigQuery. This is because matrix factoring can be **very computationally expensive**, so to protect the user from a potentially high bill, you can only train matrix factoring models with **flat-rate prices**. And the easiest and most flexible way to configure this is with the **flex slots**.

The **average rank**, also known as the **mean percentile rank**, is perhaps the most-used metric for **implicit matrix factoring**
with **0.5** being a **random probability**, and **0** being a **perfect prediction**

To predict: `ML.RECOMMEND`

In production

One way is to export recommendations for ad redirection campaigns with Google Analytics.

First, by focusing on specific products at once, you can create a new column for Likelihood of Purchase, based on your expected recommendations, something like taking the top five items for each user, then scaling the time spent viewer between 0 and 1 for each user, as a proxy for likelihood to purchase.

You can then re-import your predictions **into Google Analytics to create new campaigns** for those products.

Another way is by connecting the intended recommendations with your **Customer Relationship Management system**, or CRM.

By doing so, you can create **targeted email campaigns to deliver relevant products directly to your inbox**.

Optimization

ML models are **mathematical functions with parameters and hyperparameters**.

A parameter is a **real valued variable that changes during model training**.

A hyperparameter is a setting that we **set before training** and which doesn't change afterwards.

$Y=mx+b$: This same concept of a relationship defined by a **fixed ratio change between labels and features** can be extended to arbitrarily high dimensionality (in feature or label)

n-dimensional generalization of a line, which is called a hyperplane,
decision boundary

property of **extending to unseen examples** is called **generalization**

Introducing the course dataset

scatterplotting

too much data is **computationally infeasible**,
scatterplots with lots of data become **visually hard to interpret**

binning

graph represents groups of data, specifically **quantiles**.
No sample. no risk of it getting a nonrepresentative sample
repeatable and parallelizable

least-squares regression: **analytically** determining the **best possible weights** for linear models.

Once you start using really **big data** sets, the **computation required to analytically solve this problem becomes impractical**

Solution: gradient descent

Introduction to loss functions

analytical methods for finding the best set of model parameters **don't scale** (for big data)

Loss function quantifies the quality of predictions for a group of data points from our training set

sum of sign errors cancel each other out

often used is what is called the **Mean Squared Error**

That can make the **MSE kind of hard to interpret**.

So we often take the square root of the MSE instead to get units that we can understand

Although RSME works fine for linear regression problems, it **doesn't work as a loss function for classification**.

The domain on the X axis . The range on the Y axis

One of the most commonly used loss functions for classification is called **cross entropy or log loss**

unlike RSME, cross entropy penalizes bad predictions very strongly

Gradient descent

we framed optimization as a **search in parameter-space** and then introduced the loss functions as a way to **compare these points**.

Gradient descent refers to the process of walking down the surface formed by using our loss function on all the points in parameter-space.

In actuality, we'll only know loss values at the points and parameter-space where we've **evaluated our loss function**.

the problem of finding the bottom can be decomposed into two different and important questions.

Which direction should I head?

And how far away should I step?

You can think of a loss surface as a **topographic or contour map**.

Every line represents a specific depth

The closer the lines are together, the steeper the surface is at that point.

STEP size :

if your step size is **too small**,

your training might take forever.

You are **guaranteed to find the minimum** though, and I'd use the word the, because for the moment we're going to assume that there is only one, however, in the future there might be more than one,

If your step size is **too big**,

you might either **bounce from wall to wall of your loss surface** or

bounce out of the valley entirely and into an entirely new part of the loss surface.

the process is **not guaranteed to converge**.

Overshoot: jump (step over) the valley

how should we vary step size?

the slope, or the rate at which the curve is changing, gives us a decent sense of **how far to step and the direction** at the same time.

In practice , our basic algorithm often either **takes too long, finds suboptimal minima or doesn't finish**

Troubleshooting loss curves

Problem: When the slope is too high or too low

Sol: scaling parameter (learning rate)

You can imagine using **brute force** to figure out the best value for learning rate, but recall that learning rate is likely to have a **problem-specific best value**.

it is set before learning begins, learning rate is a **hyperparameter**

is a fraction **significantly less than one**

ML model pitfalls

Convexity: loss surface has one minimum

When we calculate the derivative, the cost of the calculation is proportional to

the number of data points we are putting into our loss function

the number of parameters in our model.

to improve training time, Reduce :

the number of data points we calculate the derivative on and

Mini-batch: 10-1000 examples (**mini-batch size** (hyperparameter) is often just called **batch size**.)

using less memory and of being easy to parallelize

batch gradient descent: computes the gradient on the entire data set using batch processing

~ = **mini-batch gradient descent**

Reason why it works: it's possible to extract samples from our training data that **on average balance each other out**.

sampling strategy selects from our training set with **uniform probability**

the frequency with which we check the loss

the frequency with which we check the loss.
time-based and step-based

Lecture lab: Introducing the TensorFlow Playground

TensorFlow Playground is a powerful tool for visualizing how neural networks work.

How is it possible that a linear model can learn a non-linear decision boundary.

the X1 squared and X2 squared features

feature engineering, or the systematic improvement of, or acquisition of new features, is an extremely important part of machine learning.

what can we do when our attempts to engineer new features for linear models fail?

use more complicated models, e.g., neural networks

Lecture lab: Practicing with neural networks

the choice of activation function is what separates linear models from neural networks

As with learning rate, the optimal batch size is **problem dependent** and can be found using hyperparameter tuning

Before neural networks, data scientists spent much more time doing **feature engineering**. Now the model itself is taking over some of that responsibility, and you can think of the **layers as being a form of feature engineering** unto themselves.

Overfitting

the model has interpreted noise in the data set as significant, (has learned the noise pattern)

the model has more decision-making power than is strictly necessary for the problem.

they generalize poorly, because new data are unlikely to have quite the same pattern of noise,

Performance metrics

consequences

long training times,

suboptimal minima

inappropriate minima.

won't generalize well,

don't reflect the true relationship (between features & label) being modeled or both

problem with our loss function

perfect loss function: rewarded the truly best strategies and penalized the bad ones,

There will always be a gap between the metrics we care about and the metrics that work well with gradient descent

loss function cannot be piecewise (differentiability)

Instead of searching for the perfect loss function during training, we're instead going to use a new sort of metric (performance metrics) after training is complete,

Performance metrics have two benefits over loss functions

easier to understand. This is because they're often simple combinations of countable statistics.

directly connected to business goals.

while loss and the business goal that is being sought will often agree, they won't always agree.

it will be possible to lower loss while making little progress toward the business goal

Confusion matrix

Recall is often inversely related to precision.

linear models can learn non-linear relationships when given non-linear features, and how neural networks learned hierarchies of features.

generalization and sampling

When is the most accurate ML model not the right one to pick?

for regression problems, the loss metric that you want to optimize for is typically mean square error, MSE, or RMSE, the root mean square error.

The RMSE is **directly interpretable** in terms of the **measurement units** on that y-axis, so it's a better measure of **goodness of fit** than a **correlation coefficient**.

A more complex model could have more free parameters -> RMSE=0 : perfectly accurate

it also might help it memorize simpler or smaller datasets

In ML, we often have lots of data and no such intuition.

one of the best ways to assess the **quality** of a model is to see how it performs well against a new dataset that it hasn't seen before.

we can determine whether the model generalizes well across new data points.

It's a good **proxy for production or real-world data**.

models that generalize well will have similar loss metrics or error values across **training and validation**

as soon as you start seeing your models **not perform well against your validation dataset**, like if your loss metrics start to increase or creep up, it's time to stop

Training and evaluating an ML model is an experiment with finding the **right generalizable** model and model parameters that **fits** your training dataset **without memorizing** them

Overfitting or memorizing that training dataset can be often far worse than having a model that only **adequately fits** your data

Somewhere in between an underfit and an overfit is the **right level of model complexity**

When to stop model training

validation data

Model selection

fine-tune the hyperparameters

If there is not significant divergence between the loss metrics from the training run and the loss metric for the validation data set, then the model could still be optimized and tuned.

You will choose the **model configuration** that results in the lower loss on the validation data set, **not** the model configuration that results in lower loss on the training one.

Vertex AI can carry out a **Bayesian search through hyperparameter space**

What data set do you use to do that **final go or no-go evaluation**?

Cannot report the error on the validation data set, Because you used the validation data set to choose **when to stop the training**,

remember? It is **no longer independent**

Once your model has been **trained and validated**, you can then run it once against the independent **test data set to compute the loss metric**.

what happens if your model fails to perform against the test data set even though it passed validation?

It means you **cannot retest** the same ML model and need to either, one, **create and train a brand-new model**, or two, **collect more data samples into your original data set**.

It's far better for your model to fail at the final test than to fail after you have productionalized it. No one likes to waste data, and it seems that the test data is essentially wasted.

bootstrapping or cross-validation: The compromise method is to do the training validation split many, many times. The upside is that you get to use all the data, but you have to train lots more times.

If you have lots of data, use the approach of having a completely independent, held-out test data set. If you don't have that much data, use the cross-validation approach.

Creating repeatable samples in BigQuery

SQL, **structured query language**,

get 80 percent of your dataset: `WHERE RAND()<0.8`

a simple random function will just grab a **new** set of five randomly selected rows shown here, **each time you run the query**.

This makes it extremely difficult, almost impossible to **identify and split the remaining** 20 percent of your data for those validation and testing buckets.

In addition, the dataset might also be **sorted**, which **could add bias** into your sample.

And simply adding in **ORDER BY** also comes with its own problems when doing something like **mini-batch gradient descent**.

you want to be able fundamentally to create these **repeatable** samples of data

One way to achieve this is to use the last few digits of a hash function (FARM_FINGERPRINT: turn the date into a long string of digits) on the field that you're using to split or bucketize your data

FARM_FINGERPRINT Hash function **returns the exact same value any time it's evoked on a specific date**

80% Training: `MOD(ABS(FARM_FINGERPRINT(date)), 10) < 8`

10% Training: `MOD(ABS(FARM_FINGERPRINT(date)), 10) == 8`

So be sure to split your data into those valid sets based on the column that you can afford to lose it's distributed and noisy.

prototyping: Starting out with ML model developments best to develop your TensorFlow code on a **small subset of data**. And then later, **scale it out** to the Cloud for true productionalization.

Summary

We describe Veex AI AutoML and how to build, train, and deploy an ML model without writing a single line of code.

We then discuss how to optimize a machine learning model and how generalization and sampling can help assess the quality of ML models for custom training.

training phase and an inference phase.

In any ML project, aer you dene the business use case and establish the success criteria, the process of delivering an ML model to production involves the following steps (manually or can be completed by an automated pipeline):

TensorFlow on Google Cloud

Introduction to TensorFlow

TensorFlow is an **open-source, high-performance library for numerical computation** (all kinds of GPU computing) **del training**

you can write your own computation code in a high-level language like **Python** and have it be executed in a **very fast** way at run time

The way TensorFlow works is that you create a directed graph (DAG), to represent the computation that you want to do **solve missing values**

Scalar: zero-dimensional, or rank zero.

Your data in TensorFlow are tensors. They flow through the graph, hence the name TensorFlow

So why does TensorFlow use directed graphs to represent computation? And the answer is **portability**.

The directed graph is a **language independent representation of the code** in your model

In statistics, machine learning, and other data science model tasks, it's not surprising that the characteristics, often with **visual** methods, **predictions**, you can use this same Python code and execute it both on CPUs, GPUs, and TPUs. This provides **language and hardware portability**.

Three popular ways this is achieved are **classical exploratory and Bayesian** JVM, and its bytecode

The representation helps with the portability of Java code. As a developer, you write the code in a high-

In regression level language like Java, have it executed in different platforms by the JVM. **four features to predict the continuous** value of

our label. Now the JVM is itself very efficient and targeted to the exact OS and hardware written in C or C++

As a developer you write your code in a high level language like Python. And have it executed in **different platforms**. By the TensorFlow execution engine. Now the TensorFlow execution engine is **training and hyper-**

parameter very efficient and **targeted toward the exact hardware chip** and its capabilities. And it's written in C++

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

- **Portability** between devices enables a lot of power and flexibility.
- **Train** on your local machine or on the cloud. You train a TensorFlow model on the cloud. On lots and lots of powerful hardware.
 - Train on the cloud
 - Evaluate on the cloud
 - Transfer to the edge
- **Upload** the model to the edge. Then you can do predictions with the model
- **Deploy** the model to the edge. Then you can do predictions with the model
- **Send predictions** to your endpoint
- **Specify a model** to use for predictions.
- **Manage** the model during predictions.

TensorFlow is this portable, powerful, production-ready software to do numeric computing

its popularity (2019) **deep learning** **researchers** **community** **became** **it** **and** **the** **ability** **to** **do** **it** **is** **also** **known** **as** **the** **coefficient** **of** **determination**

05:31 because the ability to **productionalize models**, to do things at scale. **tured data** using just **SQL**.

TensorFlow API hierarchy

Steps for working with BigQuery ML:

1. Write a SQL query to extract their training data from BigQuery
2. Create a model, specifying model type

high-level APIs for **super-powerful tasks** like creating an 128-layer neural network

00:14 with just a few lines of code written with the Keras API.

TensorFlow levels of abstraction: **ment whose value is set before the learning process begins.**

hardware platforms (G/TPU/Android)

Recorded **platforms** **to** **use** **to** **develop** **new** **products** **and** **services.**

RMSE does **not** **exist** **at** **the** **lowest** **level** **of** **abstraction** **is** **the** **layer** **that's** **implemented** **to** **target** **the** **different** **hardware** **platforms.**

Generalization **And** **unless** **your** **company** **makes** **hardware**, it's unlikely that you'll do much at this level, Models that it does exist. **have similar error values across training and validation.**

Outlier TensorFlow (C++ API)

The next level is the **TensorFlow C++ API**. This is how you can write a **custom TensorFlow ML operation**.

Core TensorFlow (Python API)

TensorFlow will give you a Python wrapper that you can use just like you would use an existing function (used by ML researcher).

Now, the **core Python API** is what contains much of the numeric processing code

Tf.losses, tf.metrics, tf.optimizers, ...

Then there are sets of **Python modules** that have high-level representation of useful neural network components.

Why are custom neural network models emphasized? Because you often don't need a custom neural network model.

Many times you're quite happy to go with the **relatively standard way of training, evaluating and serving** models. You don't need to customize the way you train. You're going to use one of the family of **gradient descent-based optimizers**,

and you're going to **back-propagate the weights and do this iteratively**. In that case, **don't write the low-level session loop**. Just use an **estimator or a high-level API** such as Keras.

Tf.estimator, tf.data, tf.keras, ...

the **high-level APIs** allow you to easily do **distributed training, data preprocessing, the model definition, compilation and overall training**. It knows how to **evaluate**, how to create a **checkpoint**, how to **save** a model,

how to set it up for TensorFlow **serving** and more

if you see an example of TensorFlow code on the Internet that does not use the estimator API, ignore that code. You'll have to write a lot of code to do **device placement**,

memory management and distribution

Cloud AI platform (CAIP) goes from low-level to high-level APIs.

Regardless of the abstraction level you're writing your TensorFlow code, using Cloud AI Platform, or CAIP, gives you that **managed service**.

It's **fully hosted** TensorFlow, so you can run TensorFlow on the **cloud** on a **cluster of machines** without having to install any software or manage any servers.

Components of Tensorflow: Tensors and variables

one-dimensional tensor: It'll grow horizontally

matrix : a stack of 1-D tensors

3-D : it's a 2-D tensor with another 2-D tensor on top of it

Tf.constant(): constant tensors

Tf.Variable(): modifiable tensors: when we need to **adjust those model weights**

3-D look like? Well, it's a 2-D tensor with another 2-D tensor on top of it

Tf.stack()

Tf.shape: quite handy in debugging

Stack ~= slice down

Tf.reshape(): essentially Python would read the input by **row by row** (MATLAB: col by col) and put numbers into the output tensor

Tf.Variable():

The **variable constructor** requires an initial value for the variable

which can be a tensor of **any shape and type**. This initial value defines the **type** and the **shape** of the variable. **After construction, the type and shape of the variable are fixed**.

The value can be changed using one of the assigned methods: **assign**, **assign_add** or **assign_sub**

all the operators are overloaded for the tensor class, are carried over to variables

TensorFlow has the ability to calculate the **partial derivative of any function with respect to any variable**

To differentiate automatically, TensorFlow needs to **remember what operations happened in what order during that forward pass**. Then during the **backward pass**,

TensorFlow traverses this list of operations in reverse order to compute those gradients. **GradientTape** is a **context manager** in which those partial differentiations aren't calculated. The functions have to be expressed within TensorFlow operations only,

but since most basic operations like addition, multiplication and subtraction are overloaded by TensorFlow ops, these happen seamlessly. Let's say we want to compute a loss gradient. TensorFlow records all operations executed

inside the context of **tf.GradientTape** onto a **tape**. Then it uses that tape and the gradients associated with each recorded operation to **compute the gradients of a recorded computation** using that **reverse-mode differentiation** like we mentioned.

custom-gradient functions

to write a new operation or to modify the calculation of the differentiation.

when the default calculations are **numerically unstable** or you wish to **cache an expensive computation**

from the forward pass

Design and Build an Input Data Pipeline

ML project,

define the business use case

establish the success criteria,

delivering an ML model to production

manually / automated pipeline: **Data extraction, data analysis, data preparation, model training,**

model evaluation, model validation, model serving and model monitoring.

phases in machine learning,

a training phase

an inference phase

In some cases, the data is **raw** and must be transformed to feature vectors

Better features result in **faster training and more accurate predictions** (efficient).

data pipelines (a series of data processing steps)

opening a file if it has not been opened.

fetching a data entry from the file,

using the data for training.

- tf.data** is one way to help build efficient data pipelines
- to build complex input pipelines from simple reusable pieces
- to handle large amounts of data**, read from different data formats and perform complex transformations

Training on large datasets with tf.data API

Dataset abstraction that represents a **sequence of elements**, in which each element consists of one or more **components**.

image pipeline an element might be a single training example with a pair of Tensor components representing the image and it's label.

There are two distinct ways to create a dataset,

a **data source** constructs a dataset from **data stored in memory** or in one or more files.

Or a data transformation constructs a dataset from one or more **tf.dataset objects**.

Large datasets tend to be **sharded** or broken apart into multiple files

you train on **mini** batches of data, **you don't need to have the entire dataset in memory**.

One mini batch is all you need for one training step.

The **dataset API** will help you create input functions for your model that **load data in progressively**

dataset classes

text files like CSVs,
TensorFlow records or
fixed length record files

TextLineDataset

to instantiate a dataset object which is comprised of one or more **text files**.

TfRecordDataset, TFRecord files,

FixedLengthRecordDataset

is a dataset object from FixedLengthRecords or one or more **binary files**

For anything else, you can use the **generic dataset class**

Dataset = TfRecordDataset(files).shuffle(buffer_size=X).map(lambda x:parse(x)).batch(batch_size=X)

the iterator opp is created only once but executed as many times as there are elements in the input pipeline

properly **disposing** of the **iterator resource** is essential as it is not uncommon for your iterator resources to **allocate**, say **hundreds of megabytes to gigabytes of memory** because of **internal buffering**

Working in-memory and with files

data is in-memory

tf.data.Dataset.from_tensors : (single element) [[1,2], [3,4]]

tf.data.Dataset.from_tensor_slices : [1,2], [3,4]

Read CSV

TextLineDataset (optionals: **type of compression of the files**, the number of parallel reads) for reading CSV

Tf.decode_csv

The map function is responsible for **parsing each row of the CSV file**.

tf.data.experimental.make_csv_dataset

If you have **TfRecords** (which is recommended), you may use

tf.data.experimental.make_batched_features_dataset

For sharded CSVs

Dataset.list_files: scan a disk and load a dataset of file names using a

It supports a **glob-like syntax with stars to match file names with a common pattern**

.flat_map (one-to-many transformation. loading a csv file with TextLineDataset, one file name becomes a collection of text files): we'll use a TextLineDataset to load the files and **turn each file name into a dataset of text lines** (one-to-many transformation).

We **flat_map** all of them together **into a single dataset** (flatten into one), and then we map each line of text.

.map (one-to-one transformation): to apply the CSV parsing algorithm

Dataset allows for data to be **prefetched**.

Without prefetching, the CPU will be preparing the first batch while the GPU is just hanging around doing nothing. Once that's done, the GPU can then run the computations on that batch. **When it's finished, the CPU will start prepreparing the next batch.**

Prefetching allows for subsequent batches to be prepared as soon as their previous batches have been sent away for computation.

By combining **prefetching and multi-threaded loading and preprocessing**, you can achieve a **very good performance** by making sure that **each of your GPUs or CPUs are constantly busy**.

Getting the data ready for model training

Tf.data.Dataset : Does more than just ingesting data

Skip, map, filter, cache

Tf.feature_column.

Numeric_column

categorical_column_with_vocabulary_list

By default, out-of-vocabulary values are ignored

categorical_column_with_vocabulary_file

categorical_column_with_hash_bucket

to distribute your inputs into a finite number of buckets by hashing them

categorical_column_with_hash_bucket

Data has already indexed

bucketized_column

columns for continuous values that you want to bucketize (discretizing continuous feature values.)

embedding_column(categorical_column=?, dimension=?)

as the **number of categories** of a feature grow large, it becomes **infeasible** to train a neural network using those one-hot encodings.

we can use an **embedding column**. Embeddings overcome this limitation.

data at a **lower dimensional level** or a **dense** vector (any value, not just 0 and 1)

crossed_column

Categorical columns are represented in TensorFlow as **sparse tensors**

TensorFlow can do math operations on sparse tensors **without having to convert them into dense values first**, and this **saves memory and optimizes compute time**.

Embeddings

neural network embeddings have three primary purposes.

finding nearest neighbors in the embedding space. These can be used to make recommendations based on user interests or cluster categories.

as input into a machine learning model for a supervised task.

for **visualization of concepts and relations between categories**.

Movie recommender: which movies are similar to each other

DNN:

input layer dim = N: # of movies

Hidden layer = d: embedded dimension

$d \ll N$

$d = 3$: good starting point

trade-off in selecting embedding dimensions

Higher-dimensional embeddings can more accurately represent

relationships between input values. However, at the more dimensions you have, the greater chance of **overfitting**. Also, the model gets larger and leads to **slower training**. A good starting point is to go with the **fourth root of the total number of possible values (empirical trade-off)**.

$N=500K$, fourth-root=25, $15 < HP < 35$

Crossed-column

combine the features into a new synthetic feature.

Combining features into a single feature, (feature crosses) (synthetic feature), enables a model to learn separate weights for each combination of features.

Feature crosses help represent **nonlinear relationships**.

Crossed column does not build the full table of all possible combinations which could be very large.

Instead, it is backed by a **hashed column**, so you can choose how large the table is.

To train the column, you simply need to write an **input function** that returns the features dictionary and a label.

When passing data to the **built-in training loops** of a model, you should either use **NumPy** arrays if your data is **small** and **fits in memory** or **tf.data.dataset** objects

Once you have defined the feature columns, you can use a **denseFeatures** layer to input them to the Keras model.

DenseFeatures(feature_columns): produces a **dense tensor** based on the given feature columns

Lab intro: TensorFlow Dataset API

to train on **large data sets that aren't going to fit in memory**.

TensorFlow has that API called "**Datasets**" that can handle this and feed your model while **loading the data progressively from disk**.

Note that the last batch may not contain the exact number of elements you specified because the dataset was **exhausted**.

If you want batches with the exact same number of elements per batch, we will have to **discard the last batch** by setting: `dataset = dataset.batch(batch_size, drop_remainder=True)`

Scaling data processing with tf.data and Keras preprocessing layers

With **Keras preprocessing layers**, you can build and export models that are truly **end-to-end**.

Models that accept **raw images or raw structured data** as input and models that handle **feature normalization or feature value indexing** on their own.

preprocessing layers

text preprocessing, numerical features preprocessing, categorical features preprocessing, image preprocessing, and image data augmentation.

text preprocessing

Tf.keras.layers.TextVectorization

the TextVectorization layer turns raw strings into an **encoded representation** that can be read by an **Embedding** layer or **Dense** layer.

adapt will analyze the data set, determine the **frequency of individual string values**, and create a vocabulary from them.

If there are more unique values in the input than the maximum vocabulary size, the most frequent terms will be used to create the vocabulary.

Tf.keras.layers.Normalization

The numerical features preprocessing layer will coerce its inputs into a distribution centered around zero with standard deviation one.

feature-wise normalization of input features.

The tf.keras Discretization

layer will place each element of its input data into one of several

contiguous ranges, and output an **integer index** that indicates which range each element is placed in.

Essentially, this layer turns **continuous numerical features into bucket data**, with discrete ranges.

Tf.keras.layers.CategoryEncoding

turns **integer categorical** features into **one-hot, multi-hot, or count-dense** representations.

Tf.keras.layers.Hashing

performs **categorical feature hashing**, also known as "the hashing trick."

`Tf.keras.layers.StringLookup`

turns **string categorical** values into an encoded representation that can be read by an Embedding layer or a Dense layer.

`Tf.keras.layers.IntegerLookup`

turns **integer categorical** values into an encoded representation that can be read by an Embedding layer or a Dense layer.

these layers are **nontrainable**. It must be set before training, either by initializing them, from a **precomputed** constant, or by **adapting** them on data.

From `Tf.keras.layers.experimental` import preprocessing

the `adapt` method takes either a **NumPy** array, or a **tf.data.dataset** object

Keras preprocessing provides two different options in applying the data transformation.

the preprocessing layer is **part of the model**

It is **part of the model computational graph** that can be **optimized and executed on a device like a GPU**.

[08:02](#) This is the **best option for the normalization** layer and all **image** preprocessing and data **augmentation** layers, **if GPUs are available**.

[08:10](#) With this option, **preprocessing will happen on the device synchronously with the rest of the model execution**, which means that it will **benefit from GPU acceleration**.

Option two uses **dataset.map** to convert data in the data set.

[08:24](#) Data augmentation will happen **asynchronously** on the **CPU** and is **non-blocking**.

[08:29](#) Its key focus is to take advantage of **multiple threading in the CPU**.

[08:35](#) With this option, your preprocessing will happen on the CPU asynchronously and **will be buffered before going into the model**.

[08:43](#) In addition if you call `dataset.prefetch` `tf.data`.

[08:47](#) **AUTOTUNE** on your data set, the preprocessing will happen **efficiently in parallel with training**.

[08:54](#) Apply it to your `tf.data` data set, to obtain a data set that yields **batches of pre-processed data**.

[09:01](#) **When running pre-processing on a TPU, you should always place preprocessing layers in the `tf.data` pipeline with the exception of**

[09:10](#) **normalization and rescaling, which run fine on TPU**, and are commonly used as the first layer in an image model.

Even if you choose option **two**, you may later want to export an **inference only, end to end model** that will **include the preprocessing layers**.

[09:25](#) As a key benefit, this makes your model **portable** and helps **reduce the training/serving skew**.

When all data preprocessing is part of the model, others can load and use your model

[09:36](#) **without having to be aware of how each feature is expected to be encoded and normalized**.

This is especially powerful if you're **exporting your model to another run time**, such as **TensorFlow.js**.

You won't have to **re-implement your preprocessing pipeline in JavaScript**

Classifying Structured Data using Keras Preprocessing Layers

If you were working with a very large CSV file (so large that it does not fit into memory), you would use `tf.data` to read it from disk directly

you will wrap the dataframes with `tf.data`, in order to shuffle and batch the data

```
dataframe = dataframe.copy() labels = dataframe.pop('target') ds = tf.data.Dataset.from_tensor_slices((dict(dataframe), labels))
[(train_features, label_batch)] = train_ds.take(1)
```

For each of the Numeric feature, you will use a `Normalization()` layer to make sure the mean of each feature is 0 and its standard deviation is 1

If you many numeric features (hundreds, or more), it is more efficient to concatenate them first and use a single [normalization](#) layer.

```
normalizer = preprocessing.Normalization(axis=None) # TODO # Prepare a Dataset that only yields our feature. feature_ds =
dataset.map(lambda x, y: x[name]) # Learn the statistics of the data. normalizer.adapt(feature_ds)
```

Often, you don't want to feed a number directly into the model, but instead use a one-hot encoding of those input

The Keras functional API is a way to create models that are more flexible than the [tf.keras.Sequential](#) API

```
tf.keras.utils.plot_model(model, show_shapes=True, rankdir="LR")
```

The model you have developed can now classify a row from a CSV file directly, because the preprocessing code is included inside the model itself.

To get a prediction for a new sample, you can simply call `model.predict()`. There are just two things you need to do:

1. Wrap scalars into a list so as to have a batch dimension (models only process batches of data, not single samples)
2. Call `convert_to_tensor` on each feature

Building Neural Networks with TensorFlow and Keras API

How do you escape from having just a linear model?

adding a nonlinear transformation layer, which is facilitated by a nonlinear activation function such as a sigmoid, tan H or ReLU.

usually neural networks have all layers nonlinear for the first and minus-one layers and then

[04:27](#) have the final layer transformation be linear for regression or sigmoid or a softmax for classification.

these kind of have a saturation, which leads to what we

[04:58](#) call the **vanishing gradient problem** where, where there's zero gradients, the models' weights don't update anything

rectified linear unit, or ReLU, for short, is one of our favorites because it's simple, and it works really well.

Networks with ReLU hidden activations often have 10 times the speed of training than networks with sigmoid hidden activations.

dying ReLU effect

Softplus, or a smooth ReLU function

The logistics sigmoid function is a smooth approximation of the derivative of the rectifier.

Leaky ReLU function

parametric ReLU

learns parameters that control the leakiness and shape of the function.

exponential linear unit, or ELU, or ELU,

is a generalization of the ReLU that uses a parameterized exponential function to transform from positive to small negative values.

Its negative values push the mean of the activations closer to zero.

08:05 That means that activations are closer to zero, enable faster learning as they bring the gradient closer to a natural gradient.

The Gaussian error linear unit, or GELU,

that's another high-performing neural network activation function like the ReLU, but its nonlinearity

08:22 results in the expected transformation of a stochastic regularizer, which randomly applies the identity or zero map to that neuron's input.

Training neural networks with TensorFlow 2 and the Keras Sequential API

TF.Keras, again, that's TensorFlow's high level API for building and training your deep learning models. useful for fast prototyping, state-of-the-art research and productionalizing these models

Keras advantages
 user-friendly
 modular and composable
 easy to extend

A sequential model, is appropriate for a

00:58 plain stack of layers where **each layer has exactly one input tensor and one output tensor**.

01:05 **Sequential models, not really advisable** if the model that you're building has **multiple inputs or multiple outputs**.

Any of the layers in the model have multiple inputs and multiple outputs that, that model needs to

01:15 do layer sharing or the model has a nonlinear topology such as a residual connection or if it multi-branches.

With a single layer, the model is linear.

the **deeper** neural net gets, generally the more powerful it becomes in learning patterns from your data, but **one thing you really have to watch out**

02:30 for is this can cause the model to **overfit** as it may almost learn all the patterns in the data by **memorizing** it and **not generalize to unseen data**.

Once we define the model object, we compile it.

Other parameter options could be the **lost weight, the sample weight mode and the weighted metrics**

An optimizer that's generally used in machine learning is SGD, or Stochastic Gradient Descent.

Adam is an optimization algorithm that can be used instead of the classical Stochastic Gradient Descent procedure to update network weights iteratively based on training data.

Adam

computationally efficient
 little memory requirements
 invariability due to the diagonal rescaling of the gradients
 well-suited for models that have large and large and large data sets or if you have a lot of parameters that you're adjusting.
 very appropriate for problems with very noisy or sparse gradients and nonstationary objectives

momentum, which reduces learning rate when the gradient values are small,

AdaGrad, which gives frequently occurring features low

05:08 **learning rates,**

Adadelta, which improves AdaGrad by avoiding and reducing LR to zero,

FTRL, or follow the regularized leader.

Adam and FTRL make really good defaults for deep neural nets as well as linear models

an epoch is a complete pass on the entire training data set,

steps for epoch, which is the number of batch iterations before a training epoch is considered

05:56 **finished,**

Callbacks are utilities called at certain points **during model training** for activities such as **logging** and visualization using tools such as **TensorBoard**.

The predict function in the tf.

07:12 Keras API returns a **NumPy** array or arrays of the predictions.

07:19 The steps parameter determines the total number of steps before declaring the prediction round finished.

if steps is set to none, predict will run until the input data set is exhausted.

Serving models in the cloud

We'll export the model to a TensorFlow **SavedModel** format.

Once we have a model in this format, we have lots of ways to serve the model (for the others as a service), **web application**, code like JavaScript, from a **mobile application**, etcetera.

SavedModel is a universal serialization format for TensorFlow models.

SavedModel provides a **language neutral format** to save your machine learning models that is both recoverable and hermetic.

Tf.keras.models.load_model

Tf.saved_model.save

One of the ways that we can serve the model is to utilize the Cloud AI platform managed service.

The AI platform service also performs **scaled training**

we need to create a version for our model.

a bucket in which to save this staging training archives.

A staging bucket is only required if a file upload is necessary, that is, other flags include local paths.

Once the model is created and then pushed to AI platform, you can just use this command `gcloud ai-platform predict`

Introducing the Keras Sequential API on Vertex AI Platform

The Keras Sequential API allows you to create TensorFlow models layer by layer.

it does not allow you to create models that share layers, re-use layers or have multiple inputs or outputs

The `.fit()` function works well for small datasets which can fit entirely in memory. However, for large datasets (or if you need to manipulate the training data on the fly via data augmentation, etc) you will need to use `.fit_generator()` instead.

The `.train_on_batch()` method is for more fine-grained control over training and accepts only a single batch of data.

when calling `.fit` the method inspects the data, and if it's a generator (as our dataset is) it will invoke automatically `.fit_generator` for training.

The `steps_per_epoch` parameter is used to mark the end of training for a single epoch

The `steps` parameter determines the total number of steps before declaring the prediction round finished.

We'll export the model to a TensorFlow SavedModel format. Once we have a model in this format, we have lots of ways to "serve" the model, from a web application, from JavaScript, from mobile applications, etc.

```
tf.saved_model.save(model, EXPORT_PATH)
```

Training neural networks with TensorFlow 2 and the Keras Functional API

wide and deep learning (Keras Functional API): jointly training a **wide linear model for memorization** alongside a **deep**

00:39neural network for generalization one can combine the strengths of both to bring us one step closer to human-like intuition.

It's useful for generic large scale regression and classification problems with sparse inputs.

Such as recommender systems, search, and ranking problems.

memorization also allows us to further refine our generalized rules with exceptions.

You want to use linear models (good for sparse and independent features) to minimize the number of free parameters, and if the columns are independent, linear models may suffice.

Nearby pixels, however, tend to be highly correlated so putting them through a neural network or a

02:24deep neural network, we have the possibility that the inputs get decorrelated and mapped to a lower dimension.

the number of hidden nodes is much less than the number of input nodes.

The Functional API gives your model the ability

to have multiple inputs and outputs.

share layers

ad hoc network graphs (nonlinear topology)

With that functional API, models are defined by creating instances of layers and then connecting them directly to each other in pairs,

03:14then defining a model that specifies the layers act as the input and the output to the model when stringing everything together.

03:23The functional API, it's a way for you to create models that are more flexible than the sequential API.

In the functional API, models are created by specifying their inputs and outputs in a graph of layers.

In the functional API, models are created by specifying their inputs and outputs in a graph of layers.

04:35That means a single graph of layers can be used to generate multiple models.

04:40You can treat any model as if it were a layer by calling it on an input or an output of another layer

Shared layers are often used to encode inputs that come from, say, similar places.

they enable the sharing of the information across these different inputs and they make it possible to train a model on much less data.

To share a layer in the functional API, just call the same layer instance multiple times.

Strengths

It's less verbose than using keras.

07:35Model subclasses.

07:36It validates your model while you're defining it.

In the functional API, your input specification, that's your shape and your [dtype] is created in advance through the input.

And every time you call a layer, the layer checks that the specification passed to its matches, its assumptions, and it'll raise a super helpful error message if not.

07:57This guarantees that any model that you built with a functional API will run.

All debugging other than conversions related to debugging will happen statically during the model construction and not at execution time.

And you can easily access intermediate nodes/layers in this graph.

Your functional model can be serialized or cloned.

Because a functional model is a data structure rather than a piece of code, it's safe to serialize and can be saved

08:41as a single file that allows you to recreate the exact same model without having access to any of the original code.

Weaknesses

It does not support dynamic architectures.

The functional API treats models as DAGs, or directed acyclic graphs, of those layers.

This is true for most deep-learning architectures, but not all.

recursive networks or Tree-RNNs, do not follow this assumption and cannot be implemented in the functional API

Sometimes you just need to write everything from scratch.

Model subclassing

The sequential model is very straightforward and consists of a simple list of layers but is limited to **single-input, single-output**

The functional API is an easy-to-use, fully featured API that supports arbitrary model architectures.

Most people in most use cases should use the functional API, which is the Keras industry strength model.

The third way to create a Keras model is model subclassing where you **implement everything from scratch** on your own.

You should use this if you have complex, out-of-the-box research use cases.

Choosing a model depends on the amount of customization you need.

the functional API also has limits on customization.

With model subclassing, you can create your own fully customizable models in Keras.

This is done by subclassing the model class and implementing a call method.

Dropout will randomly drop out units and correspondingly scale up activations of the remaining units.

Although you can use the sequential API for ease of use or the functional API for more customization, use model subclassing for complete flexibility and control.

(Optional) Lab intro: Making new layers and models via subclassing

Subclassing refers to inheriting properties for a new object from a base or superclass object.

Layer class, which is a combination of state, weights, and computation.

Layers can be recursively composable by creating sublayers.

Compute the loss by using `add_loss` method and compute the average by using `add_metric` method.

you can also enable the serialization on layers

Subclassing refers to inheriting properties for a new object from a base or superclass object.

Defer weight creation until the shape of the inputs is known.

The `__call__()` method of your layer will automatically run `build` the first time it is called

we recommend creating layer weights in the `build(self, input_shape)` method of your layer.

Layers are recursively composable

If you assign a Layer instance as an attribute of another Layer, the outer layer will start tracking the weights of the inner layer.

We recommend creating such sublayers in the `__init__()` method (since the sublayers will typically have a `build` method, they will be built when the outer layer gets built).

Note that the `__init__()` method of the base Layer class takes some keyword arguments, in particular a name and a dtype.

It's good practice to pass these arguments to the parent class in `__init__()` and to include them in the layer config

```
def __init__(self, units=32, **kwargs): super(Layer, self).__init__(**kwargs) self.units = units
```

```
def get_config(self): config = super(Layer, self).get_config() config.update({"units": self.units}) return config
```

Some layers, in particular the BatchNormalization layer and the Dropout layer, have different behaviors during training and inference.

For such layers, it is standard practice to expose a training (boolean) argument in the `call()` method.

By exposing this argument in `call()`, you enable the built-in training and evaluation loops (e.g. `fit()`) to correctly use the layer in training and inference.

You will find it in all Keras RNN layers.

A mask is a boolean tensor (one boolean value per timestep in the input) used to skip certain input timesteps when processing timeseries data.

Keras will automatically pass the correct mask argument to `__call__()` for layers that support it, when a mask is generated by a prior layer. Mask-generating layers are the Embedding layer configured with `mask_zero=True`, and the Masking layer.

In general, you will use the Layer class to define inner computation blocks, and will use the Model class to define the outer model -- the object you will train.

several ResNet blocks subclassing Layer, and a single Model

the Model class has the same API as Layer, with the following differences:

- It exposes built-in training, evaluation, and prediction loops (`model.fit()`, `model.evaluate()`, `model.predict()`).
- It exposes the list of its inner layers, via the `model.layers` property.
- It exposes saving and serialization APIs (`save()`, `save_weights()`...)

the Layer class corresponds to what we refer to in the literature as a "layer" (as in "convolution layer" or "recurrent layer") or as a "block" (as in "ResNet block" or "Inception block").

Meanwhile, the Model class corresponds to what is referred to in the literature as a "model" (as in "deep learning model") or as a "network" (as in "deep neural network").

So if you're wondering, "should I use the Layer class or the Model class?", ask yourself: will I need to call `fit()` on it?

Will I need to call `save()` on it? If so, go with Model. If not (either because your class is just a block in a bigger system, or because you are writing training & saving code yourself), use Layer.

- A Layer encapsulate a state (created in `__init__()` or `build()`) and some computation (defined in `call()`).
- Layers can be recursively nested to create new, bigger computation blocks.
- Layers can create and track losses (typically regularization losses) as well as metrics, via `add_loss()` and `add_metric()`
- The outer container, the thing you want to train, is a Model. A Model is just like a Layer, but with added training and serialization utilities.

Regularization basics

generalization theory, or G theory, that goes about defining the statistical framework.

14th-century principle laid out by William Ockham

Occam's razor, that principle, as our basic heuristic guide in favoring those simpler models, which make less assumptions about your training data.

balance complexity against loss.

right balance between simplicity and actually accurate fitting of the training data

the complexity measure is multiplied by a lambda coefficient (hyperparameter), which will allow us to control our emphasis on model simplicity

optimal lambda value for any given problem is really data-dependent, which means that we almost need to spend time tuning this either manually or automated search.

How can we measure model complexity: L1 vs. L2 Regularization

Regularization

early stopping

parameter norm penalties

L1

L2 (weight decay)

Max-norm

dataset augmentation

noise robustness,

sparse representations

Regularization refers to any technique that helps generalize a model.

A generalized model performs well not just on training data, but also on never-seen test data.

Both L1 and L2 regularization methods represent model complexity as the magnitude of the weight vector (norm function.)

L2 norm denoted by the double bars and the subscript of two

model complexity is measured in the form of magnitude of the weight vector.

use the square of L2 norm to simplify calculation of derivatives.

Lambda: This is a simple scaler value that allows us to control how much emphasis we want to put on model simplicity over minimizing training error.

L1 regularization results in a solution that is more sparse.

This property of L1 regularization is extensively used as a feature selection mechanism.

Training at Scale with Vertex AI

different abstraction layers for distributed training

how do you actually run distributed TF at scale in production?

distributed TensorFlow on Vertex AI.

gather and prepare your training data and,

put that training data in an online source that Vertex AI can access, like Cloud Storage.

When sending training jobs to Vertex AI, it's common to split most of the logic into a task.py file and a model.py file.

Task.py is the entry point to your code that Vertex AI will start and knows job-level details like how to parse the command line arguments, how to long run, where to write the outputs, how to interface with the hyperparameter tuning and so on.

To do core ML, task.py will invoke model.py.

Sharing code between computers always involves some type of packaging.

Sending your model to Vertex AI platform for training is no different.

TensorFlow and Python in particular require a very specific but standardized packaging structure

Python modules need `__init__.py` in every folder

test your code locally, which is the first step.

To make the code compatible with Vertex AI,

upload the data to Google Cloud Storage,

move the code into a trainer Python package,

we package our code as a source distribution using setup.py and setup tools.

Use the sdist command to create a source distribution

then copy that Python package to your GCS bucket.

submit the training job with gcloud to train on Vertex AI.

There are two general configurations: with a prebuilt container and without.

When you create a custom job, you specify settings that Vertex AI needs to run your training code, 02:18including location; the region where the container or Python package will be run; job name, which is required

02:26-- It's a display name for the custom job -- and Python package URIs, which is a comma-separated

02:33list of Cloud Storage URIs specifying the Python package files, which are the training program and its dependent packages.

02:42The maximum number of package URIs is 100.

02:46One worker pool for single node training or worker pool spec; machine type, which is the type of machine;

02:54replica count, which is the number of worker replicas to use -- In most cases, this is set to one

02:59for your first worker pool -- Python package executor image URI, which is the URI of the container image

03:09that runs the provided Python package; Python module, which is the Python module name to run after installing the packages.

If you want to perform distributed training, then you can specify the worker pool spec flag multiple times, once for each worker pool.

components:

region, display-name, python-package, worker-pool and any arguments

prebuilt container :will run in a specific region.

The python-package is held in a Google Cloud Storage bucket.

03:59The spec of the worker pools includes machine type and Docker image.

04:04Arguments include output directory, the training data path, batch size and the number of examples to train.

04:13If you want to specify configuration options that are not available in the preceding examples, you can use the config flag

04:20to specify the path to a config yaml file in your local environment that contains the fields of custom job spec.

if an option is specified both in the configuration file and via the command line arguments, the command line arguments override the configuration file.

To get the best performance for ML jobs, make sure you select a single region bucket in Google Cloud Storage.

The default is multi-region, which is better suited for web serving than ML training.

You can monitor training jobs with the Google Cloud Console.

You can see exactly how they were invoked, check out their logs and see how much CPU and memory they are consuming.

While inspecting log entries may help you debug technical issues like an exception, it's really not the right tool to investigate the ML performance.

05:21TensorBoard, however, is a great tool.

05:24To use it, make sure your job saves summary data to Google Cloud Storage location, and then when you start TensorBoard, simply provide that directory

Vertex AI will build you a production-ready web app out of your trained model and offer a batch service for your less-latency-sensitive predictions.

Since these are both REST APIs, you'll be able to make scalable, secure inferences from whatever language you want to write the client in.

Training at scale with the Vertex AI Training Service

We'll take advantage of Google Cloud's Vertex AI Training Service.

00:09 Vertex AI Training Service is a managed service that allows the training and deployment of ML models without having to provision or maintain servers.

you'll organize your training code into a Python package, train your model using cloud Infrastructure via Vertex

00:29 AI Training Service, and run your training package using Docker containers and push training Docker images on a Docker registry.

Create a Cloud Storage bucket

Create a bucket using the Google Cloud console:

- i. In your Cloud Console, click on the **Navigation menu** (), and select **Cloud Storage**.
- ii. Click on **Create bucket** and choose a Regional bucket.
- iii. Set a unique name (use your project ID because it is unique).
- iv. Click **Create**.

Objectives

1. Learn how to organize your training code into a Python package
2. Train your model using cloud infrastructure via Google Cloud Vertex AI Training Service
3. (optional) Learn how to run your training package using Docker containers and push training Docker images on a Docker registry

Vertex AI Training Service is a managed service that allows the training and deployment of ML models without having to provision or maintain servers.

Create BigQuery tables

If you have not already created a BigQuery dataset for our data, run the following cell:

In [5]:

```
bq = bigquery.Client(project=PROJECT)
dataset = bigquery.Dataset(bq.dataset("taxifare"))

try:
    bq.create_dataset(dataset)
    print("Dataset created")
except api_core.exceptions.Conflict:
    print("Dataset already exists")
```

Export the tables as CSV files

In order to make our code compatible with Vertex AI Training Service we need to make the following changes:

1. Upload data to Google Cloud Storage
2. Move code into a trainer Python package
3. Submit training job with gcloud to train on Vertex AI

A Python package is simply a collection of one or more .py files along with an `__init__.py` file to identify the containing directory as a package

The `__init__.py` sometimes contains initialization code but for our purposes an empty file suffices.

package directory: `__init__.py` `model.py` `task.py`

Paste existing code into model.py

A Python package requires our code to be in a .py file, as opposed to notebook cells.

We specify these parameters at run time via the command line. Which means we need to add code to parse command line parameters and invoke `train_and_evaluate()` with those params. This is the job of the `task.py` file.

Summary

TensorFlow gives you the ultimate machine learning solution.

TensorFlow is commonly used for deep Learning, classification and predictions, image recognition, and transfer learning.

uses directed graphs (DAGs)

DAGs are used in models to illustrate the flow of information through a system and is simply a graph that flows in one direction, with nodes (mathematical operations).

TensorFlow uses DAGS to represent computation because of portability. TensorFlow applications can be run on almost any platform: local machines, cloud clusters, iOS and Android devices, CPUs, GPU, or TPUs

TensorFlow contains multiple abstraction layers, `.estimator`, `.keras`, and `.data`.

The `.data` API enables you to build complex input pipelines from simple, reusable pieces to handle large amounts of data, read from different data formats, and perform complex transformations

The Dataset API will help you create input functions for your model that will load data progressively.

There are specialized Dataset classes that can read data from text files like CSVs (TextLineDataset), TensorFlow Records (TFRecordDataset), or fixed length record files (FixedLengthRecordDataset).

As the number of categories of a feature grows large, it becomes infeasible to train a neural network using one-hot encodings.

We can use an embedding column to overcome this limitation. Instead of representing the data as a one-hot vector of many dimensions, an embedding column represents that data as a lower-dimensional, dense vector in which each cell can contain any number, not just 0 or 1

Keras preprocessing layers allow you to build and export models that are truly end-to-end: models that accept raw images or raw structured data as input and models that handle feature normalization or feature value indexing on their own.

Sparse & independent features -> Linear models

Dense & highly correlated features -> DNN

Feature Engineering.

Feature Store benefits

Vertex AI Feature Store and feature engineering, using BigQuery machine learning, Keras, and tf.

However, the team is now divided into multiple teams recreating the same features, which leads to redundant work.

00:25 And each time the team needs to add a new feature to a model, they are dependent on the ops team.

The XYZ team's software developer needs to understand how to add ML to their apps via online prediction

00:50 endpoints without having to understand how to procure or generate the features required by the models at prediction time.

XYZ's DevOps resource is new to the XYZ team and needs to understand how to monitor and maintain feature serving and management infrastructure.

Feature challenges (feature management pain points): Vertex AI Feature Store solves these feature management problems

Features are hard to share and reuse, and reliably serving in production with low latency is a challenge.

01:27 In addition, inadvertent skew in feature values between training and serving is common (training-serving skew).

Vertex AI Feature Store provides a centralized repository for organizing, storing and serving machine learning features.

By using a central feature store, the XYZ team can efficiently share, discover and reuse ML features at scale, letting the team increase the speed at which they can develop and deploy new management learning applications.

02:36 Feature Store is a fully managed solution.

It manages and scales the underlying infrastructure for you, which means that your data scientist can

02:45 focus on the feature computation logic instead of worrying about the challenges of deploying features into production.

the team can share and reuse ML features across use cases because Feature Store has a centralized
03:00 feature repository with easy APIs to search and discover features, fetch them for training and serving and manage permissions.

The team can also serve ML features at scale with low latency because the operational overhead is handled by Feature Store.

A common problem in machine learning is termed training-serving skew, which means the data you train with may be skewed or different from the data that is served in production.

03:32 Feature Store alleviates training-serving skew because it let's you compute feature values once, and reuse them for both training and serving.

03:45 You can also track and monitor for drift and other quality issues.

Feature Store also allows for feature ingestion in large batches or in real time as data streams in.

With Vertex AI Feature Store, the team can store features with batch and stream import APIs and register the feature to its feature registry.

04:10 This allows other team members such as data analysts, ML engineers, software developers and the data scientists to easily

04:17 find the feature with Discovery API and retrieve the feature values for fast online serving with continuation feature monitoring.

They can also retrieve feature batches for training jobs and point-in-time lookup to prevent data leakage

Feature Store terminology and concepts

A feature store is a top-level container for your features and their values.

When you set up a feature store, permitted users can add and share their features without additional engineering support.

Users can define features and then ingest or import feature values from various data sources.

An entity type is a collection of semantically related features.

00:33 You define your own entity types based on the concepts that are relevant to your use case.

An entity is an instance of an entity type.

In a feature store, each entity must have a unique ID and must be of type STRING.

if features are distributed across multiple entity types, you can retrieve them together

01:57 in a single request that you can feed to a machine learning or batch prediction request.

a feature is a value that is passed as an input to a model.

Essentially a feature describes some entity

Feature store captures feature values for a feature at specific point in time.

02:36 In other words, you can have multiple values for a given entity and feature.

feature store associates a tuple identifier with each feature value, entity_ID, feature_ID and timestamp, and which it then uses to look up values at serving time.

If all feature values were generated at the same time, you are not required to have a time stamp column.

Feature store keeps feature values up to the data retention limit.

This limit is based on a time stamp associated with the feature values, not when the values were imported.

There is a one-to-many relationship between an entity and a feature.

Feature ingestion is the process of importing feature values computed by your feature engineering jobs into a feature store.

Before you can ingest data, the corresponding entity type and features must be defined in the feature store.

Feature store offers batch ingestion so that you can do a bulk ingestion of values into a feature store.

your computed source data might live in locations such as BigQuery or cloud storage.

05:37 You can then ingest data from those sources into a feature store, so that feature values can be served in a unified format.

Feature serving is the process of exporting stored feature values for training or inference.

Feature store offers two methods for serving features, batch and online.

06:01 Batch serving is for high-throughput and serving large volumes of data for offline processing, like for model training or batch predictions.

06:11 Online serving is for low-latency data retrieval of small batches of data for real-time processing, like for online predictions.

Feature Store data model.

The Feature Store data model includes an entity type, entities, features, and feature values.

00:14 Feature Store uses a time series data model to store a series of values for features.

00:21 This model enables Feature Store to maintain feature values as they change over time.

In the Feature Store, the timestamps are an attribute of the feature values, not a separate resource type.

If all feature values were generated at the same time, you are not required to have a timestamp column.

challenges with ML features that come up often.

00:18 Features are hard to share and reuse.

00:20 Reliably serving a feature in production with low latency is a challenge, and inadvertent skew in feature values between training and serving is common.

Before creating a feature store, you'll need to preprocess your data.

00:43 Ensure that your features are clean and tidy, which means that there are no missing

00:48 values, data types are correct, and any one-hot encoding of categorical values has already been done.

There are some requirements for your source data.

01:01 Vertex AI Feature Store can ingest data from tables in BigQuery or files in Cloud

01:07 Storage, but for files in Cloud Storage, they must be in the Avro or CSV format.

01:15 You must have a column for entity IDs, and the values must be of type STRING.

Your source data values or your source data value types must match the value types of the destination feature in the feature store.

All columns must have a header that is of type STRING.

01:48 There are no restrictions on the names of the headers.

01:52 For BigQuery tables, the column header is the column name.

01:56 For Avro, the column header is defined by the Avro schema that is associated with the binary data.

02:04 And for CSV files, the column header is the first row.

If you provide a column for feature generation timestamps, use one of the following timestamp formats.

02:15 For BigQuery tables, timestamps must be in the TIMESTAMP column.

02:20 For Avro, timestamps must be of type long and logical time or logical type timestamp-micros.

02:28 For CSV files, timestamps must be in the RFC 3339 format.

CSV files cannot include array data types.

02:39 Use Avro or BigQuery instead.

For array types, you cannot include a null value in the array although you can include an empty array.

You can create a feature store in the Vertex AI console or the Vertex AI Workbench using the API.

One, you cannot delete a feature store from the console at this time.

03:32 You must do it from the API.

if you need to add another feature store, you must add it using the API.

Now that the feature store is created, you need to create an entity type.

Step three is to add features.

05:33 Recall that a feature is a measurable attribute of an entity type.

After you add features in your entity type, you can then associate your features with values stored in BigQuery or Cloud Storage.

feature monitoring: Feature owners, such as data scientists, might monitor feature values to detect data drift over time.

In feature store, you can monitor and set alerts on feature stores and features.

is monitoring a feature store to track its CPU utilization

Ingestion jobs import features data from BigQuery or Cloud Storage

Before you import data, you need to define the corresponding entity type and features.

Feature store offers batch ingestion so that you can do a bulk ingestion of values into a feature store.

Note the number of ingested entities of 1,200, which barely meets the minimum number of 1,000 rows required for a data set to be uploaded into Vertex AI.

As a reminder, data for ingestion should have the following columns, entity_id, the ID of the ingested entity, timestamp,

09:24 the timestamp at which the feature was generated or computed, and feature columns that match the destination feature name.

Serving features: Batch and online

the process of exporting stored feature values for training or inference.

The key APIs are batch ingestion API.

00:36 When a user ingests feature values via the batch ingestion API, the data is reliably written both to an offline store and to an online store.

00:49 The offline store will retain feature values for a long time so that they can later be retrieved for training.

00:57 The online store will contain the latest feature values for online predictions.

The online serving API will be used by client applications to fetch feature values to perform online predictions.

Online serving is for low latency data retrieval of small batches of data for real-time processing, like for online predictions.

The batch serving API is used to fetch data from the offline store for training a model or for performing batch predictions.

Batch serving is for high throughput and serving large volumes of data for offline processing, like for model training or for batch predictions.

Batch ingestion let's you ingest feature values in bulk from a valid data source.

For each request, you can import values for up to 100 features for a single entity type.

Note that you can run only one batch ingestion job per entity type to avoid any collisions.

Because each batch ingestion request is for a single entity type, your source data must also be for a single entity type.

Another way to add feature values is by streaming near real-time data with the online serving API and calling the API with the list of required features.

As part of a batch serving request, the following information is required.

04:50A list of existing features to get values for, a read-instance list that contains information for each training example, the destination URI and format where the output is written.

05:05In the output, Feature Store essentially joins the table from the read instances list and the feature values from the feature store.

05:13Specify one of the following formats and locations for the output: CSV file in a regional or multi-regional Cloud Storage bucket, or TFRecord file in a Cloud Storage bucket.

05:26As a final note, batch serving jobs must be created in the Feature Store API.

Using Feature Store

Vertex Feature Store, a managed cloud service for machine learning engineers and data scientists to store, serve, manage, and share machine learning features at a large scale.

Vertex SDK for Python

Automatically/programmatically restart kernel

Make sure to [choose a region where Vertex AI services are available](#). You may not use a Multi-Regional Storage bucket for training with Vertex AI.

Authenticate your google Cloud account

You need a BigQuery dataset to host the output data in us-central1

the name of your Cloud Storage bucket must be unique across all of your Cloud Storage buckets.

Feature Store organizes data with the following 3 important hierarchical concepts:

Featurestore -> EntityType -> Feature

- **Featurestore:** the place to store your features
- **EntityType:** under a Featurestore, an *EntityType* describes an object to be modeled, real one or virtual one.
- **Feature:** under an EntityType, a *feature* describes an attribute of the EntityType

The method to create a featurestore returns a [long-running operation](#) (LRO). An LRO starts an asynchronous job. LROs are returned for other API methods too, such as updating or deleting a featurestore.

You can use [GetFeaturestore](#) or [Featurestores](#) to check if the Featurestore was successfully created

Search created features

While the [ListFeatures](#) method allows you to easily view all features of a single entity type, the [SearchFeatures](#) method searches across all featurestores and entity types in a given location (such as us-central1). This can help you discover features that were created by someone else.

You can query based on feature properties including feature ID, entity type ID, and feature description. You can also limit results by filtering on a specific featurestore, feature value type, and/or labels.

how to import feature values by calling the ImportFeatureValues API using the Python SDK.

BigQuery table/Avro/CSV are supported

each imported entity *must* have an ID; also, each entity can *optionally* have a timestamp, specifying when the feature values are generated

When importing, specify the following in your request:

- Data source format: BigQuery Table/Avro/CSV
- Data source URL
- Destination: featurestore/entity types/features to be imported

Online serving

The [Online Serving APIs](#) lets you serve feature values for small batches of entities.

It's designed for latency-sensitive service, such as online model prediction.

The ReadFeatureValues API is used to read feature values of one entity.; hence its custom HTTP verb is readFeatureValues. By default, the API will return the latest value of each feature, meaning the feature values with the most recent timestamp.

To read feature values, specify the entity ID and features to read. The response contains a header and an entity_view. Each row of data in the entity_view contains one feature value, in the same order of features as listed in the response header.

To read feature values from multiple entities, use the StreamingReadFeatureValues API, which is almost identical to the previous ReadFeatureValues API. Note that fetching only a small number of entities is recommended when using this API due to its latency-sensitive nature.

Batch Serving

Batch Serving is used to fetch a large batch of feature values for high-throughput, typically for training a model or batch prediction

BatchReadFeatureValues API.

Note that there is a timestamp column in Table 2. This indicates the time when the ground-truth was observed. This is to avoid data inconsistency.

Cleaning up

To clean up all Google Cloud resources used in this project, you can [delete the Google Cloud project](#) you used for the tutorial.

You can also keep the project but delete the featurestore:

Overview of feature engineering

Feature engineering:

the transformation of a given feature with the objective of reducing the modeling error for a given target.

Feature engineering can be defined as a process that attempts to create additional relevant features from

01:05the existing raw features in the data and to increase the predictive power of the learning algorithm.

the process of combining domain knowledge, intuition and data science

01:21 skill sets to create features that make the models train faster and provide more accurate predictions.

In some cases, the data is raw and must be transformed into feature vectors.

Better features result in faster training and more accurate predictions.

feature vectors can be numerical, categorical, bucketized, crossed, embedding and hashed.

Engineering features is primarily a manual time-consuming task.

The process involves brainstorming where you delve into the problem.

customer data, sales data, product data, inventory data, operational data and people-management data

Formats: csv, json, xml

Feature extraction is a process of dimensionality reduction by which an initial set of raw data is reduced to more manageable groups for processing.

By reducing the dimension of your feature space, you have fewer relationships between variables to consider, and you are less likely to overfit your model.

devise features

There's no well defined basis for performing effective feature engineering.

It involves domain knowledge, intuition and most of all, a lengthy process of trial and error.

The key learning is that different problems in the same domain may need different features.

feature engineering in reality is an iterative process.

you create a baseline model with little to no feature engineering as

09:21 determined by your data types and then add feature engineering to see how your model improves.

Feature engineering types

include using indicator variables to isolate key information,

for example geolocation features for

09:41a New York City taxi service where you isolate a certain area for your training data set.

09:49 It can also include highlighting interactions between two or more features,

meaning that you can actually include the sum

09:55 of two features, the difference between two features, the product of two features and the quotient of two features.

10:05 Other types include representing the same feature in a different way.

10:10 For example, in numeric to categorical mappings where you have grade, you can create categories

10:18 and create a new feature, grade, with elementary school, middle school and high school as classes.

10:27 Another example is to group sparse classes where you group similar classes and then group the remaining ones into a single other class.

10:38 And another example is to transform categorical features into dummy variables.

Raw data to features

Good feature engineering can take, on average, 50 to 75 percent of your time working with machine learning.

Good feature

If something is not related to the objective, throw it away.

[00:24](#) You have to make sure that it's known at prediction time.

[00:28](#) You need to make sure it is numeric.

[00:33](#) It has to have enough examples.

[00:36](#) And you need to have some human insights.

You need to have a reasonable hypothesis of why a particular feature might matter for this particular problem.

[00:51](#) Don't just throw arbitrary data in there in the hope that there is some relationship somewhere.

[00:56](#) You don't want to do what is called data dredging, that's dredging the large data set and finding whatever spurious correlations

[01:03](#) exist, because the larger the data set is, the more likely it is that there are lots of these spurious correlations.

Features should be known at prediction-time

A common mistake people make is to look into their data warehouse and take all the data in there.

Some of the information in a data warehouse is known immediately, and some other information is not known in real time.

Feature definitions should not change over time.

You cannot train with current data and predict with stale data.

If you go to a data warehouse for training, you cannot use all of the values for a

[04:17](#) customer's credit card usage because not all of those values are going to be available at the same time

The key point is that you have to train with stale data if stale data is what you have in real time.

You have to correspondingly correct for the staleness of your data.

If you've trained your model assuming that you know exactly the data, up to the minute, but then, at prediction [05:06](#) time, you'll only know the data as of three days ago, then you'll have a very poorly performing machine learning model.

You have to think about the temporal nature of all the input variables that you're using.

Features should be numeric

The third key aspect of a good feature is that all your features have to be numeric, and they have to have meaningful magnitude.

That's a number code, but they don't have meaningful magnitudes.

arial:0, times new roman:1

So the meaningful magnitude part is important. (ordinal)

To make the words numeric, you can run something called Word2vec. (NLP)

When you look at these vectors at the end of Word2vec, these vectors are such that, if you take the vector from "man" and you

03:56take the vector from "woman" and you subtract them, the difference is similar to the difference of the vector for "king" and the vector for "queen."

You could just go ahead and throw random encoding in there, but your model is not going to be as good as if you

04:17had started with a vector encoding that's nice enough that it understands the context of male and female, "man" and "woman," "king" and "queen."

Your features need to be numeric, and they need to have meaningful magnitudes.

04:31They need to be useful.

04:32You need to be able to do arithmetic on them, and you need to find vector representations in such a way that these kinds of qualities exist.

Features should have enough examples

a rule of thumb, is that you need to have at least five examples of any value before using it in your model.

discretization.

If there is not enough samples

You may have to group up your values so that you have enough examples of each value.

Representing features

in TensorFlow, you're going to take this thing, which is a JSON input, comes out of your web application, and ultimately goes into a data warehouse

There are values like price or wait time. These are already numeric.

01:10That's easy to encode. You take them and use them as is. They're numeric. They have meaningful magnitude (This is what is called a real-valued column)

So these numbers that you used as is, they'll just be real-valued columns.

Transaction ID is 42.

01:44No, that's way too specific.

01:46Throw it out.

01:47It can't be used as a feature.

Employee ID is 72365.

02:03It's a number.

02:04But does it have meaningful magnitude? no

So you can't use the employee ID as it is.

02:20You have to do something with it.

What you can do is say if this is employee number 72365, I'll represent this employee ID

02:39by this vector, the vector 01000, because I've defined the second column as corresponding to employee 72365.

02:51So essentially, I make it like a bit mask (one-hot in coding).

one-hot in coding: In TensorFlow, this is called a sparse column

You basically say that I want to create a sparse column with the keys.

03:19The column name is Employee ID.

03:21And the keys are 8345, 72365, et cetera.

You would preprocess this data to find out all the keys that occur in your training data set, and create a vocabulary of keys.

04:04That's the preprocessing.

before you get to training the model, you need to create a vocabulary.

04:19And this vocabulary needs to be available at prediction time because at prediction time, the user is going to come back and say employee ID 72365.

04:29And the model needs to know that at training time, it decided that 72365 was the second column.

The vocabulary needs to be identical.

04:39And the mapping of the vocabulary needs to be identical at prediction time.

So consider what would happen if you hire a new employee.

04:51At this point, you don't have a place to put this new employee.

04:54So what this means is that you are unable to predict for this new employee.

then you decide that perhaps you're going to find the average of all your current employees and use them.

Meanwhile, you collect data

Sometimes, your data might already be indexed.

05:44Maybe, for example, you have employee ID.

05:46And they just happen to be numbers 1 to 1,000.

05:50At that point, they're already indexed.

05:52They're not arbitrarily big numbers all over the place.

05:55They're just 1 to n.

If that's the case, you say, I want to create a sparse column with the integerized feature, which is employee ID.

Suppose you don't have a vocabulary and it's not integerized.

So what you do is that you say, I'm going to take my employee ID, hash

06:43it, compute the hash of the employee ID, and just break that hash up into 500 buckets.

So if you have something like a rating, and you want to use it as an input feature, you could do one of two things.

07:33You could treat it like a continuous number, 1 through 5.

07:36It's numeric.

07:38It sort of has meaningful magnitude for us, more than 3, or you can say that 4 stars is

07:43very different from 5 stars is very different from 2 stars, in which case I'm going to one-hot encode it.

07:51So in some cases, you have choices to the customer rating.

07:55You can one-hot encode it or you can treat it as a number-- up to you.

One option is to use two columns, one for the rating and one for whether or not you got a rating.

You would say I got a rating 4-- 0, 0, 0, 1-- or I didn't get a rating-- 0, 0, 0, 0.

08:43But don't make the mistake of not having the second column.

08:47You don't want to mix magic numbers with real values.

08:51You have to add an extra column to state whether or not you observed the value or not. If you have missing data, you need to have another column.

Machine learning versus statistics

if you had taken statistics, you might see if those missing values -- You would normally

00:03impute a value like the average for that column, so that's where philosophically, ML and statistics start to diverge.

The difference in philosophy affects how you treat outliers.

With statistics, you say, I've got all the data I'll ever be able to collect, so you throw out outliers.

Statistics is often used in a limited data regime where ML operates with lots of data quantile boundaries so that the number of values in each bin is constant.

modern architectures for ML end up taking variable magnitudes into account because of what's called batch normalization

Basic feature engineering

BigQuery preprocessing involves two aspects, representation transformation and feature construction.

00:08Feature representation is converting a numeric feature to a categorical feature through bucketization.

00:15And converting categorical features to a numeric representation through one-hot encoding, learning with counts, sparse feature embeddings and so on.

00:25Some models work only with numeric or categorical features, while others can handle mixed type features.

00:32Even when models handle both types, they can benefit from different representation, numeric and categorical of the same feature.

00:41Feature construction is creating new features, either by using typical techniques such as polynomial expansion by using univariate mathematical functions or feature crossing to capture feature interactions.

00:55Features can also be constructed by using business logic from the domain of the ML use case.

BigQuery ML supports two types of feature preprocessing, automatic and manual.

01:08Automatic preprocessing occurs during training.

01:12BigQuery ML provides the transform clause for you to define custom preprocessing using the manual preprocessing functions.

01:20You can also use these functions outside the transform clause.

BigQuery can help with feature engineering because it lets you use SQL to implement common preprocessing tasks.

here are three types of preprocessing for dates using SQL in BigQuery ML.

02:25Extracting the parts of the date into different columns: year, month, day, et cetera.

02:32Extracting the time period between the current date and columns in terms of years, months, days, et cetera.

02:39Extracting some specific features of the date, name of the week day, weekend or not, holiday or not, et cetera.

Note that BigQuery automatically splits the data we gave it and trained on only a part of the data, using the rest for evaluation.

04:09After creating your model, you evaluate the performance of the regressor using the ML.EVALUATE function.

04:17The ML.EVALUATE function evaluates the predicted values against the actual data ... root mean squared error, RMSE.

One nice property of RMSE is that the error is given in the units being measured,

A BigQuery dataset is a container for tables, views, and models built with BigQuery ML.

A baseline model is a solution to a problem without applying any machine learning techniques.

a model in BigQuery ML represents what an ML system has learned from the training data.

The primary evaluation metric for this ML problem is the root mean-squared error

One nice property of RMSE is that the error is given in the units being measured, so you can tell very directly how incorrect the model might be on unseen data.

R2 : model explains 0-100% of the variability of the response data around the mean

Advanced feature engineering: Feature crosses

ML.FEATURE_CROSS(STRUCT(features)) does a feature cross of all the combinations.

The TRANSFORM clause which allows you to specify all preprocessing during model creation.

00:16

The preprocessing is automatically applied during the prediction and evaluation phases of machine learning.

00:24

And ML.BUCKETIZE(f, split_points) where split_points is an array.

Feature crosses are about memorization.

00:34

Memorization is the opposite of generalization, which is what machine learning aims to do.

In a real-world ML system, there's a place for both.

Memorization works when you have so much data that for any single grid cell within your input space the distribution of data is statistically significant.

00:56

When that is the case, you can memorize.

00:59

You are essentially just learning the mean for every grid cell.

Whether you want to feature cross or you want to use many layers, you need a lot of data.

01:11

If you're familiar with traditional machine learning, you may not have heard much about feature crosses because they memorize and only work on large data sets.
 You will find feature crosses extremely useful in real-world data sets.
 Larger data allows you to make your boxes smaller and you can memorize more finely.
 Feature crosses are a powerful feature preprocessing technique on large data sets.
 When you do a feature cross, the input is very very sparse.
 TensorFlow will give us easy tools to deal with this.

Sparse models contain fewer features and therefore are easier to train on limited data.

03:03

Fewer features also means less chance of overfitting.

03:07

Fewer features also mean it is easier to explain to users because only the most meaningful features remain.

This is what a sparse matrix looks like, very, very wide with lots and lots of features.

[03:20](#) You want to use linear models to minimize the number of free parameters.

[03:25](#) And if the columns are independent, linear models may suffice.

The signature of any geography function starts with ST_.

ST_Distance returns the shortest distance in meters between two non-empty geographies.

BigQuery ML, by default, assumes that numbers are numeric features and strings are categorical features.

Pluralsight: Getting Started with GCP and Qwiklabs

interactive hands-on labs platform called Qwiklabs, which is part of Google Cloud.

Bucketize and Transform Functions

buckets or bins.

00:05

That is, it bucketizes a continuous numerical feature into a string feature with bucket names as the value.

ST_geogpoint function to derive the pickup and dropoff feature.

we should encapsulate the entire feature set in a TRANSFORM clause.

BigQuery ML now supports defining data transformations during model creation.

00:46

Which will be automatically applied during prediction and evaluation.

When the TRANSFORM clause is used, user specify transforms during training will be automatically applied during model serving, prediction, evaluation, et cetera.

we are using the TRANSFORM clause to separate the raw input data from the transformed features.

These transformed columns are post-process, with standardization for numerics and one-hot encoding for categorical variables by default.

The advantage of encapsulating features in the TRANSFORM clause is that the client code doing the predict doesn't change.

Note that the TRANSFORM clause must be placed after the CREATE statement.

For these types of time-windowed features, you will use Beam, batch and streaming data pipelines.

Other features that could be preprocessed one data point at a time can be implemented either in TensorFlow directly or using Beam.

02:35

Apache Beam and complementary Google Cloud technology called Dataflow will be important to this part of the module.

Predict housing prices

The tf.data API enables you to build complex input pipelines from simple, reusable pieces.

The tf.data API makes it possible to handle large amounts of data, read from different data formats, and perform complex transformations.

As a result, much of the actual effort in deploying machine learning algorithms goes into the design of preprocessing pipelines and data transformations.

Our first step is to build an input data pipeline.

The output of a feature column becomes the input to the module.

When using this column, your model will receive the column value from the data frame unchanged.

Combining features into a single feature, better known as feature crosses, enables a model to learn separate weights for each combination of features.

merge.

03:41Combine the inputs from multiple models into a single model.

the optimizer is a search technique used to update weights in your model.

05:04Adaptive moment estimation or adam is a popular gradient descent optimizer that uses adaptive learning rates

batch size -- that is, the number of training instances shown to the model before a weight update is performed.

An epoch is a number of times that the model is exposed to the training data set.

Estimate taxi fare

Here is step one of a two-step process to build the pipeline.

00:17Step one is to define our columns of data, which column we're predicting for and the default values.

Step two is to define two functions, a function to define the features and label we want to use and a function to load to the training data.

Here we define a function to build a deep neural network model.

The Sequential API allows you to create models layer by layer for most problems.

01:02It is limited in that it does not allow you to create models that share layers or have multiple inputs or outputs.

ReLU for our hidden layers and Linear for the final output layer.

Temporal and geolocation features

numpy.decode UTF-8 to decode the numpy array of string elements.

The datetime STRP time Python class method creates a datetime object from a string, representing a date and

00:20time and a corresponding format string

TF.py_function: maps a string to a tensor so that every element in day of week is passed to the model as a tensor.

Scaling latitude and longitude, it is very important for numeric variables to get scaled before they are fed into the neural network.

Normal functions are defined using the def keyword.

03:55 In Python, anonymous functions are defined using the Lambda keyword.

Keras employs a similar naming scheme to define anonymous custom layers.

04:05 Lambda layers in Keras help you to implement layers or functionality that are not pre-built, and which do not require trainable weights.