Topic Cheatsheet for GCP's Professional Machine Learning Engineer Beta Exam

Authors/contributors: David Chen, PhD

Credits & disclaimers can be found in README section of the source repository. Some references are included as %comments or hyperlinks the source file.

Abbreviations

Common abbreviations. ML, machine learning; DL, deep learning; AI, artificial intelligence, CV, computer vision; GC(P), Google Cloud (Platform); CI/CD: continuous integration / continuous delivery; SDK, software development kit; API, application programming interface; K8s, Kubernetes; GKE, Google Kubernetes Engine
Technical abbreviations. MLE, maximum likelihood estimation; ROC, receiver-operation curve; AU(RO)C, area under the (receiver-operation) curve

I. Preparation for ML

Defining an ML Problem

ML as Solution to Business Problems

- (Re)define your business problems
- Consider whether the problem could be solved *without ML*
- Define/anticipate utility of the ML output
- Identify data sources
- Pre-define "success" to solving the business challenge
  - Metric(s) used to define success
  - Key results (product or deliverables)
  - Incorrect or low-quality output (i.e. "unsuccessful" models)

Components of an ML Solution

- Define *Predictive Outcome*
- Identify Problem Type: Supervised (Classification or Regression), Unsupervised, Reinforcement
- Identify Input Feature Format
- Feasibility and implementation

"Data Science Steps for ML"

1. Data extraction
2. Exploratory data analysis
3. Data preparation for the ML Task
4. Model training
5. Model evaluation
6. Model validation
7. Model serving
   - Microservices with REST API
   - Deployment on mobile devices
   - Batch predictions
8. Model monitoring

Data Preparation

Data Ingestion: obtaining & importing data for use or storage

- File input types
- Database maintenance, migration
- Data streaming (e.g. IoT devices)

Exploratory Data Analysis is an important step prior to building any model!

- Evaluation of data quality (domain- and organization-specific knowledge/information may be needed)
- Data visualization (descriptive statistics)
- Inferential statistics (e.g. t-test to compare means, KS-tests to compare distributions) as needed, scale as needed

Feature Engineering may be necessary and/or beneficial in many ML tasks:

- Encoding structured data types
- Feature Crosses: used to define a *synthetic feature* when data cannot be linearly separated (e.g. feature cross products $\mathbf{x}_1 \times \mathbf{x}_2$
- Feature selection, e.g.
  - Univariate statistical methods (e.g. χ^2 test, t-test/linear model)
  - Recursive Feature Elimination (RFE)

Special considerations:

- Class imbalance
  - Needs to be *known*, at minimum
  - Affects the metrics to employ (e.g. F1 score, AUC would be superior to crude accuracy in imbalanced binary classification)
  - Can affect optimization choices: modify objective function; oversampling the minority class(es)
- Data leakage
  - Certain features available in your training data might not be available in the unknowns to predict!
  - When training, be careful not to include raw or engineered features that are computed from the classification/regression label
-

Data Pipeline should be designed & built in advance for at-scale applications

- Batching vs. Streaming
  - Use of data from live streams, *single event-focused*
  - Use of data stored in data lakes, processed in *periodic* intervals
- Monitoring deployed pipelines using tools such as Google Site Reliability Engine (SRE)
  - "Four Golden Signals" of your cloud-based service: latency, traffic, error, saturation
  - *Cloud Monitoring* (formerly *Stackdriver*): metric set for GC services
  - Dashboards (Stackdriver Cloud Monitoring Dashboards API) can be a powerful tool in displaying multiple metrics.
- Privacy, compliance, legal issues: Know what the restrictions are and plan ahead (e.g. privacy-preserving ML/AI, corrupting input, ...)

II. ML Model Development

Model Development At-a-Glance

Generic ML Workflow

1. Training
   - Choose a model framework
     - Supervised
     - Unsupervised
   - Consider *Transfer Learning* (if applicable)
   - Monitoring / tracking metrics
   - Strategies to handle overfitting (e.g. regularization, ensemble learning, drop-out) & underfitting (increase model complexity)
   - Interpretability
2. Validation
   - Check overfitting & underfitting
   - Compare trained model against pre-defined baseline (e.g. simple model or benchmark)
   - Unit tests
3. Scale-up & Serving**
   - Unit tests
   - Cloud AI model explainability
   - Distributed training
   - Scalable Model Analysis

ML Models

Gradient descent is used to optimize the *objective functions* of a machine-learning model:

Gradient Descent	*n*	*Resolution*
Full-batch	all (N)	complete
Mini-batch	$1 < n < N$	intermediate
Stochastic	1	noisy approximation

An *epoch* is the number of passes through the entire training dataset, and is a *hyperparameter* to be defined/tuned by the user.

Supervised Learning (with related concepts)

- Naive Bayes (flavors: Gaussian, Bernoulli, Multinomial)
- Decision trees (concept of *entropy*)
- Support Vector Machine (SVM)
  - Linearly vs. non-linearly separable
  - Kernels

Unsupervised Learning

- Clustering
  - K-means
  - Hierarchical Clustering
  - DBSCAN
- Dimensionality reduction
  - Principal Component Analysis (PCA)
  - t-SNE
- Gaussian Mixture Model (GMM), optimized by *Expectation-Maximization (EM)*:
  1. E step
  2. M step

  Repeat until convergence

Overfitting

Bias-variance trade-off

- Characteristics of Loss vs. iteration curves, separately plotted for
  - Training set
  - Validation and/or test set
- Underfitting vs. overfitting patterns

Ways to address overfitting

1. Get more high-quality, well-labeled training data
2. Regularization
   - L2 penalty
   - L1 (LASSO) penalty
   - Elastic net
3. Ensemble learning
   - Bagging
     - Random Forest: Only the randomly chosen $1 \leq m < M$ features used in split
     - Bagged Trees: all M features available used in split
   - Boosting (e.g. Gradient Boosted Trees/*XGBoost*)

Recommendation Systems

	User info	Domain knowledge
Content-based		✓
Collaborative Filtering	✓	
Knowledge-based		✓

A *hybrid recommendation systems* uses more than one of the above, though not 100% possible at all times, it is generally the preferred solution.

Deep Learning

Subtypes of Neural Networks

- Feed forward neural network
- Convolutional Neural Network (CNN) & computer vision
- Recurrent Neural Network (RNN)
  - Sequence data (speech/text, time series)
  - Vanishing gradient problem
  - Gated Recurrent Units (GRU)
  - Long-short term memory (LSTM)
  - Application to Natural Language Processing (NLP)
    - Language models
    - Embeddings
    - Architectures (e.g. transformers)
- Autoencoders (deep learning)
  - General architecture
    - Encoding layers
    - Lower-dimensional representation (returned or used as input for subsequent autoencoder in a stack)
    - Decoding layers
  - Flavors to address *trival solutions*:
    - Undercomplete autoencoder
    - De-noising autodencoders
    - Sparse autencoders
  - Application
    - Data representation (feature engineering)
    - Dimensionality reduction / data compression

III. Production-level ML with Cloud

MLOps: CI/CD in an ML System

	DevOps	Data Engineering	MLOps
Version ctrl.	Code	Code	Code, data, model
Pipeline	-	Data, ETL	Training, serving
Validation	Unit tests	Unit tests	Model valid.
CI/CD	Production	Data pipeline	(both)

Relevant GCP Tools

BigQuery

- Google-managed data warehouse
- Highly scalable, fast, optimized
- Suitable for analysis & storage of *structured* data
- Multi-processing enabled

Cloud Dataprep:

- Managed cloud service for quick data exploration & transformation
- Auto-scalable, eases data-preparation process

Cloud Dataflow: provides serverless, parallel, distributed infrastructure for both *batch* & *stream* data processing by making use of Apache Beam TM

Cloud ML APIs

- Cloud Vision AI
- Cloud Natural Language
- Cloud Speech to Text
- Cloud Video Intelligence

ML Pipeline Automation & Orchestration

Virtualization Basics

- Virtual Machines (VMs)
- Containers
  - Clusters
  - Pods
- Kubernetes (K8s)

ML Pipeline Design

The ML code is only a small part of a production-level ML system

- Identify components, parameters, triggers, compute needs
- Orchestration Framework
  - Cloud Composer (based on Apache Airflow deployment)
  - GCP App Engine
  - Cloud Storage
  - Cloud Kubernetes Engine
  - Cloud Logging & Monitoring
- Strategies beyond single cloud:
  - Hybrid Cloud: blend of public & private cloud for mixed computing, storage, & services, allowing for *agility* (i.e. quick adaptation during business digital transformation)
  - Multi Cloud: multiple clouds designated for different tasks (*but unlike parallel computing, synchronization across different ventors is NOT essential)

Procedures during Implementing a Training Pipeline

- Perform data validation (e.g. via Cloud Dataprep)
- Decouple components with Cloud Build (fully server-less CI/CD platform supporting any language)
  - Add layer of technical abstraction
  - Separate content producer & end users
  - Ensures software components are not tightly dependent on one another
- Construct & test *parametrized pipeline definition* in SDK (e.g. `gcloud ml-engine`)
- Tune compute performance
- Store data & generated artifacts (e.g. binaries, tarballs) via Cloud Storage

	Type	Transac.?	Complex Q?	Cap.
Cloud Datastore	NoSQL	✓	✗	Terabytes+
Bigtable	NoSQL	(limited)	✗	Petabytes+
Cloud Storage	Blobstore	✗	✗	Petabytes+
Cloud SQL	SQL	✓	✓	Terabytes
Cloud Spanner	SQL	✓	✓	Petabytes
BigQuery	SQL	✗	✓	Petabytes+

Considerations for Implementing the Serving Pipeline

- Model binary options
- Google Cloud serving options
- Testing for target performance
- Setup of trigger & pipeline schedule

Deployment with CI/CD (final step in MLOps), along with

- A/B testing: Google Optimize
- Canary testing, automated by GKE with Spinnaker

ML Solution Monitoring

Considerations in monitoring ML solutions:

1. Monitor performance/quality of ML model predictions on an ongoing-basis (via Cloud Monitoring (Compute Engine) with a metric model), and then debug with Cloud Debugger
2. Use robust logging strategies (e.g. Cloud Logging, especially Stackdriver (aka Cloud Operations) with beautiful dashboards)
3. Establish *continuous evaluation* metrics

Troubleshoot ML Solutions:

- Permission issues (IAM)
- Training error
- Serving error
- ML system failures/biases (at production)

Tune performance of ML solutions in production

- Simplify (optimize) of *input pipeline*
  - Reduce data redundancy in NLP model
  - Utilize Cloud Storage (e.g. object storage)
  - Simplification can take place in various places during the pipeline
- Identify of appropriate *retraining policy*
  - Under what circumstance(s)? How often? (e.g. when significant deviation or drift identified; periodically)
  - How? (e.g. by batch vs. online learning)