Google

Nail Your Next ML Gig

Natalia Ponomareva
Research, Machine Learning
Google

What is Machine Learning?

What is Machine Learning (ML)

Conceptually: given (training) data, discover some underlying pattern and use this discovered pattern (on new data).

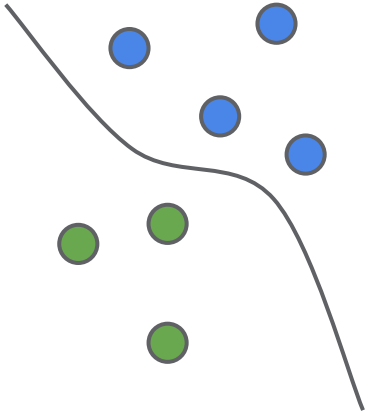
Types

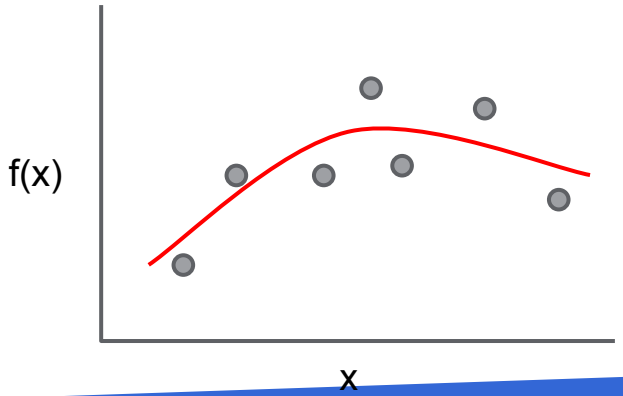
- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Etc...

What is ML: continued

Supervised Learning - training input data is labeled.

- Classification (learn a decision boundary). Example: text / image / video classification, spam detection etc
- Regression (learn to predict a continuous value). Example: predict a house price, predict how much user is willing to spend etc





What is ML: continued

Unsupervised Learning - input data does not have labels, tries to find "hidden" structure in the data

- Clustering
- Outlier/anomaly detection

Example: cluster streaming service users into groups, analyse these groups and the popularity of a video among each of these groups

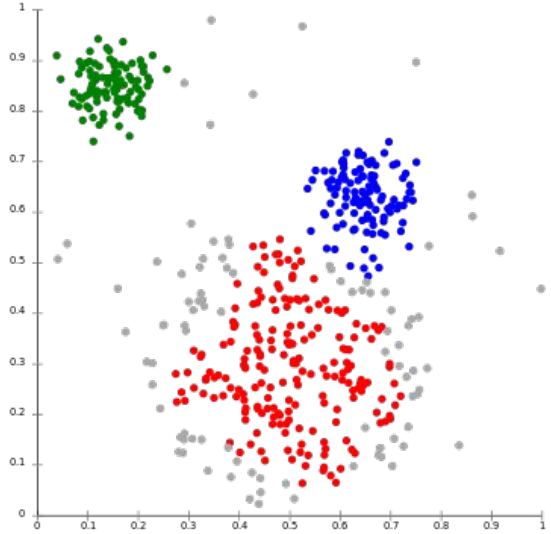


Image is from wikipedia https://en.wikipedia.org/wiki/Cluster_analysis

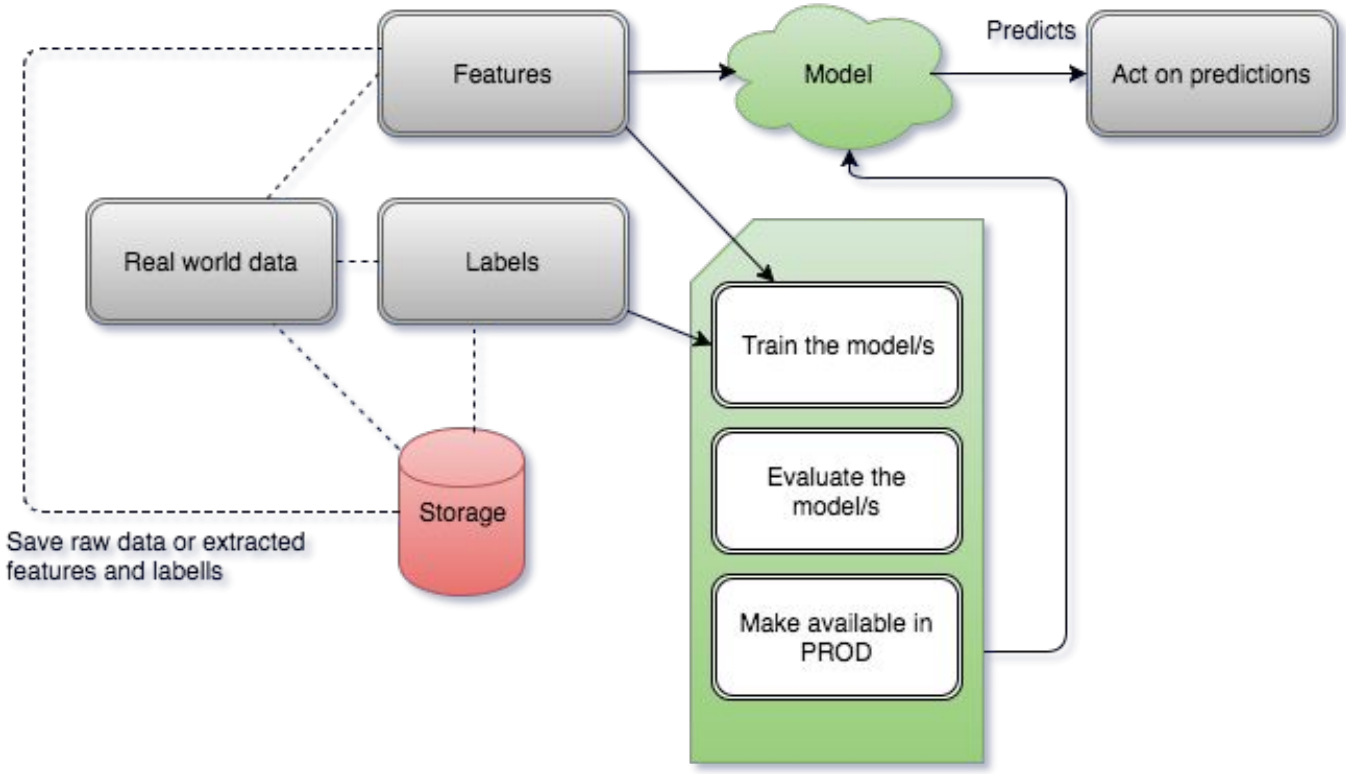
How ML can help your business

- Personalization of services (provide each user with a unique experience, tailored to the user), which can maximize the engagement and revenue
- Automate some tasks that are error prone/take a lot of time (transcription, character recognition etc)
- Analyze the data to come up with better decisions
- Etc ...

Where do I start?

- Start small
  - Sample your data
  - Come up with initial features
  - Build a simple model, see if it looks promising
- Scale
  - Train on the full data available
  - Work on improving features (feature engineering)
  - Try different algorithms (model selection)
- Think about productionalizing

Real world supervised ML conceptually



Feature engineering: what is it?

Conceptually, feature engineering is the process of transforming your raw data (logs, history of products bought or behaviour on the website etc.) into a vector that can be used by the learning algorithm for training and prediction.

- It is highly domain specific
- It depends on what you are trying to learn from the data
- It is labour-intensive

Feature engineering: how to go about it

High level steps are

- Decide on the insight you are trying to obtain (for example, we want to train a model to suggest a user another song to listen to).
- Decide how you will model the insight (there are numerous ways!)
  - For example, we will have a classification model that, given the user and a song, will return whether the user will be interested in this song or not
  - We will have the list of songs, run it throught the model and will show the songs to the user which our model thinks might be interesting to the user
- Consider what data you have (for example, history of songs the user listened to and user profile information)

Feature engineering: continued

- Consider what might be relevant
  - User age (probably)? User name and email address (not at all). Location (possibly)? History of songs the user listened to (yes).
  - What genres of songs user listened to before (country, rock, pop etc) (very relevant)
- Come up with digital representation for the relevant information
  - Come up with the features that
    - Describe the user
    - Describe the songs

Feature engineering: continued

- Prepare your final training data
  - Given the features for the user $u(u_1, u_2, u_3 \ldots u_n)$
  - And features for the songs s_1 $(s_{11}, s_{12}, s_{13}, \ldots, s_{1k})$, s_2 $(s_{21}, s_{22}, .. s_{2k})$...
  - Create training instances for a user u (u,s_i) (1 - listened to) and (u,s_j) (0 - didn't listen to)

Note: the same representation will apply during prediction time: for a given user we will try to predict whether a song is of interest = > our data during prediction time is also (u_m,s_i)

Feature normalization

Your features are most likely to be on a different scale:

- User age: numeric value between 0 and 100
- User income: from 0 to millions!

Some machine learning models may not work well with such a variety of features

- Regularization will penalize features differently
- Distance will be governed by the feature with the largest range
- Some optimization algos can converge faster (gradient descent)
- etc...

Solution: normalize the features to have approximately the same ranges, it never hurts!

Model evaluation

How do you know if your model is any good?

- What do you care about ultimately? Is it "No false positives"? Or overall how accurate the predictions are?
  - Express it as a calculable metric
  - Make sure that your metric has direct connection with what you care about!
  - Use this metric for
    - Choosing the model
    - Testing the model before deploying
    - Any refinements of the model or data (eg adding more features)

Model selection: rough guidelines

- Start simple. For example, try a linear model. Refine.
- What works well usually:
  - Classification: logistic regression, perceptron like algorithms, AdaBoost, SVMs (linear kernel for a lot of data, RBF for smallish data), random forests
  - Regressions: linear regression, random forest
- Do try several models
- Choose the model based on its performance on hold-out dataset
- Make sure that final performance metric is evaluated after a model selection on a different dataset (e.g. do not use the same dataset to choose the winning model and to report its metric)

Model selection: bringing it to the next level

Consider deep learning

- If you have a lot of labelled data (think millions of instances)
- If you have a hard time coming up with features or the connection between features is very complicated (example: object detection)
- Can tolerate longer training/refinement time
- If you know what you are doing
    - What architecture to choose? (how many layers? Fully connected or not? etc)
    - How to prevent overfitting (DNN can model rare dependencies in the data, but should they be allowed to?)

Hyperparameter tuning

ML models have hyperparameters: these are parameters that are fixed before the training starts and which affect the training process and the complexity.

Example: learning rate, regularization constant etc.

- Default values are just that: they work ok on average
- To get the best out of your ML model, you need to tune the hyperparameters to your data

Process: Set the values, train model, evaluate, refine the values (based on evaluation)

Ways:

- Grid (sweep through grid values of hyperparameters)
- Algo assisted hyperparameter tuning (Bayesian etc...)

Supervised ML Pipelines

You will need to setup pipelines for

- Training
  - Getting the data (and maybe storing the data)
  - Feature extraction and labelling the data
  - Fitting models
  - Testing the models/choosing the model
  - Storing the model
- Prediction (real-time use of your model)
  - Getting real time data
  - Extracting features from it
  - Retrieving the model
  - Using the model to predict on this new data
  - Acting on prediction

Tools/Frameworks

Things to consider before choosing a framework

- Where the training data will be stored?
  - Database? Cloud?Are you going to store features and labels, or will you extract features and label during the training?
- How you are going to train
  - In cloud? Offline?
  - How often your data changes (do you need an online model or stationary model with periodic retraining)
- How to make the model available for predictions
  - Using framework tools? Writing the pipeline from scratch?
- Do you want the framework to have monitoring functionality?
  - Fallback when things go wrong (versioning)

Next steps

So preliminary analysis is promising, what do I do next?

- Verify that you need to train on a lot of data (does your model do better when you increase your training size)
- If yes, consider training on full data
- Consider additional requirements
    - Do you need to update model as new data arrives or you can retrain occasionally
    - Does your training data all fits into memory
    - Do you have resources to setup a full ML cloud pipeline in the cloud (DIY approach) OR
    - You want to go with ML as a service

ML tools for production: Hands-on approach

1) Hands on approach:
    - Choose your storage (e.g. Google Cloud, Amazon etc.)
    - Write the code to save the data, train and predict
        - ML can be done with Open source frameworks (liblinear, Weka, Tensorflow etc) or your own implementation of the model
    - Deploy the code and pay for the resources used

Pros:

- Might be cheaper (you will pay for the cluster usage/data storage only)
- Very flexible
- Strong community support for popular frameworks

Cons

- More engaged (you need a team of developers/data scientists)

ML tools for production: Hands-on approach

- Data processing frameworks
  - Map/Reduce+Hadoop - distributed storage and processing system. M/R is a paradigm for processing large amount of data
  - Pig, Hive, Cascalog - frameworks on top of Map/Reduce, provide a level of abstraction over the map reduce operations
  - Spark - full stack solution for data processing and training
  - Google Cloud Dataflow

ML tools: ML as a service

ML as a service: Prebuilt full stack solutions (easily train and deploy the model using the stack)

- Less engaged (for example, you can train a deploy the service for prediction from a browser)
- Different components work seamlessly together (storage, clusters, training and predictions etc)
- Might be less flexible

Some of the options:

- Amazon ML
- Microsoft Azure
- IBM Watson
- Google Cloud ML

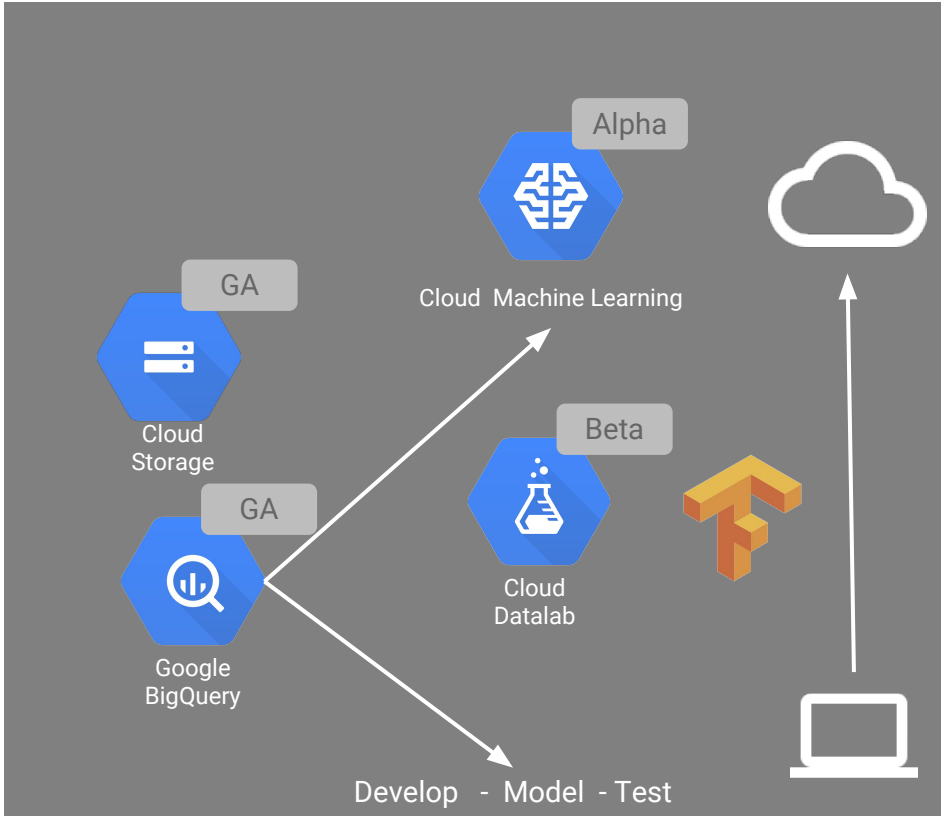
Google Cloud Offering: DIY and ML as a service!

Google

Google Cloud Services

Cloud Storage - Store your data on Google's infrastructure with very high level of durability and availability.

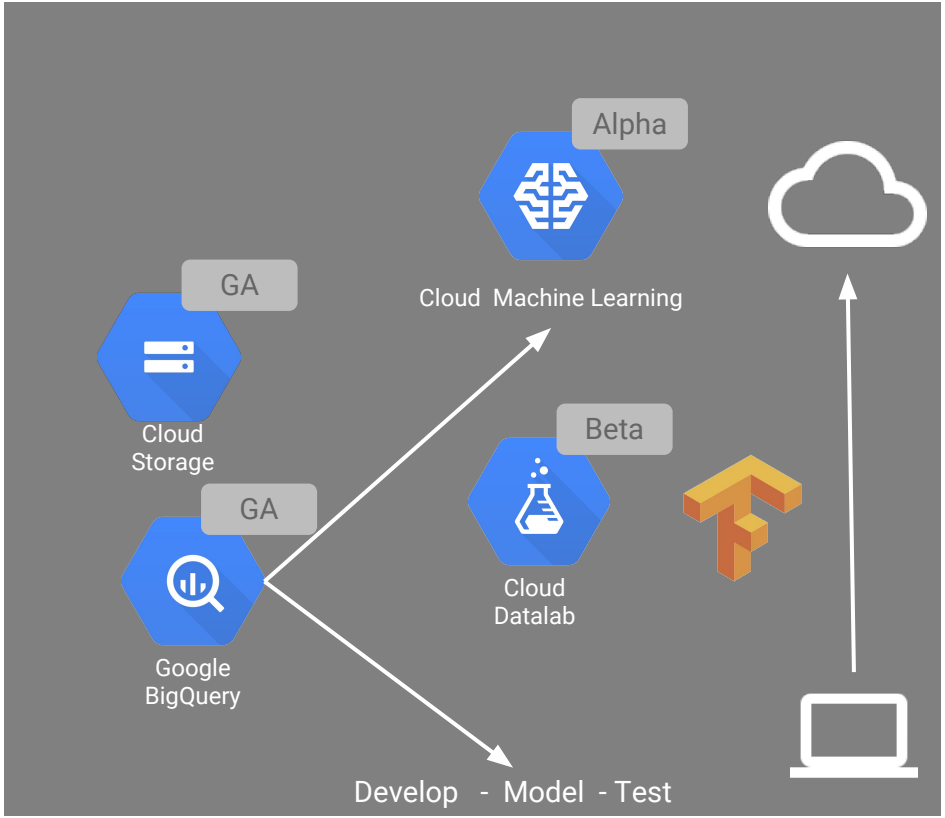
BigQuery - fully managed, petabyte scale, low cost analytics data warehouse. Run super-fast SQL-like queries against your data or read and write data in BigQuery via DataFlow, Spark and Hadoop



Google Cloud Services

Cloud DataLab - a notebook for code, results and documentation for exploring, analysing and visualizing your data.

Cloud DataFlow - unified abstraction over Map/Reduce and a managed service for developing and executing a wide range of data processing patterns including ETL, batch computation, and continuous computation.



TensorFlow

- Open sourced computation engine, specifically designed for neural networks but not limited to them
- Represent your computations as a Data Flow graph (nodes, edges and tensors)
- Very flexible
  - Use predefined components that are common to construct neural networks
  - Write your own graphs for your specific computation need
- Easily deployable on CPUs or GPUs and on desktop, server, mobile computing platforms etc
- Python and C++ interface + interactive iPython notebook!
- Great community support

Google Cloud Machine Learning (alpha)

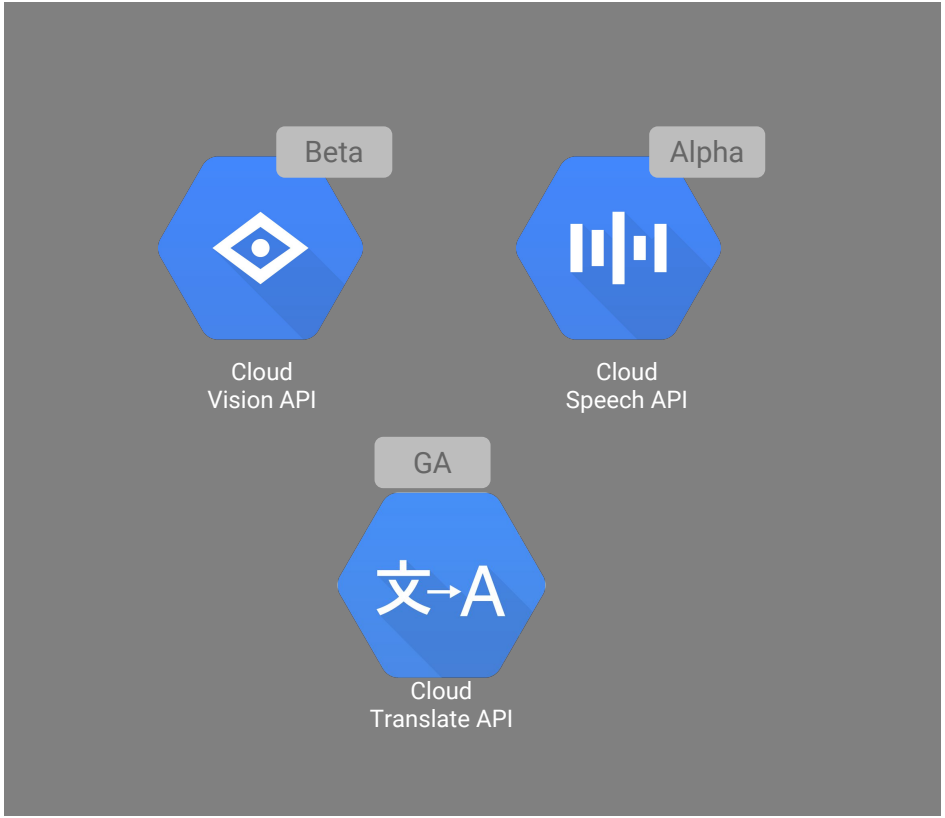
Best way to run TensorFlow at scale on cloud. Makes it easy to build sophisticated, large scale machine learning models in short amount of time. Based on same deep learning framework that powers Gmail, Google Photos etc..

- Managed no-ops infrastructure
- Can train models on ANY dataset size
  - Scale on Google's high performance cloud
  - Scale to the largest datasets
- Native support of deep learning algorithms using TF
- Datalab (Jupyter notebook) experience for interactive model development
- Works with data in many different formats and is integrated with other Google Cloud Platform products

Pretrained Models

If your problem falls into one of the following categories, consider using pre-trained pay-per-use offerings. All these models are hard to build, require extensive research/programming knowledge and better be left for the experts

- Image recognition
- Language detection and translation
- Speech recognition



Questions??????

Thanks!

Google™