



# Improving ML through feature engineering

## Serverless Machine Learning with Cloud ML

Google Cloud

©Google Inc. or its affiliates. All rights reserved. Do not distribute.  
May only be taught by Google Cloud Platform Authorized Trainers.

### Notes:

50 slides + 1 lab + 1 demo: 3 hour

# Agenda

ML abstraction levels + Lab

Hyperparameter tuning + Demo

ML Abstraction Levels

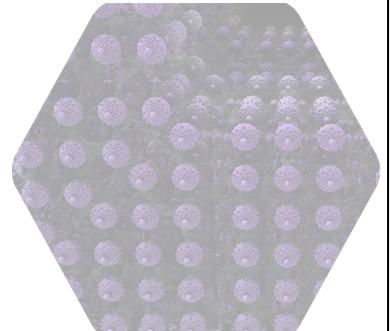
# What's left? Ways to build effective ML



Big Data



Feature  
Engineering



Model  
Architectures

## Notes:

<https://pixabay.com/en/large-data-dataset-word-895563/> (cc0)

<https://pixabay.com/en/fractal-complexity-render-3d-1232494/> (cc0)

<https://pixabay.com/en/robot-artificial-intelligence-woman-507811/> (cc0)

Now that you know \*how\* to build ML, let's learn how to do it well in the rest of the course.

Ordered from easiest to most difficult.

In the previous chapter, we refactored the model to make it easier to edit, but we haven't actually done anything to improve it.

# What makes a good feature?

Represent raw data in a form conducive for ML

1. Should be related to the objective
2. Should be known at production-time
3. Has to be numeric with meaningful magnitude
4. Has enough examples
5. Brings human insight to problem

## 1. Related to what is being predicted?

- Reasonable hypothesis for why feature value matters
- Different problems in same domain may need different features

## Quiz: Related or Not?

Objective	Feature	Good feature?
Predict total number of customers who will use a discount coupon	Font of the text with which the discount is advertised on partner websites	
	Price of the item the coupon applies to	
	Number of items in stock	
Predict whether a credit-card transaction is fraudulent	Whether cardholder has purchased these items at this store before	
	Credit card chip reader speed	
	Category of item being purchased	
	Expiry date of credit card	

### Notes:

Yes (more readers); yes (maybe more for higher prices); no (instock vs. outofstock is probably valid, # items in stock is too vague; if discount is \$1 off 3 items, then outofstock could be defined as #items in stock < 3).

Yes (goes to likelihood); no (what hypothesis is there?); yes (thieves tend to go after jewelry and other items easily exchangeable for cash); no (maybe time the account has been active is more valid since newer cards might be subject to more fraud, but expiry date is a poor signal. Issue date and expiry date are not necessarily related since cards get renewed).

## 2. Value should be known for prediction

- Feature value known at the time prediction is made?
  - Causal: can not rely on future information
  - Must ingest that data in timely manner
  - Legal/ethical to collect/use that information?



### Notes:

Cannot use a time machine :)

<https://pixabay.com/en/hand-robot-human-clock-time-1571846/> (cc0)

Some personally identifiable information is not legal to use.

For example, race of borrower of a loan can not be used.

## Quiz: Value knowable at prediction time or not?

	Feature	Good feature?
Predict total number of customers who will use a discount coupon	Total number of discountable items sold	
	Number of discountable items sold the previous month	
	Number of customers who viewed ads about item	
Predict whether a credit-card transaction is fraudulent	Whether cardholder has purchased these items at this store before	
	Whether item is new at store (and can not have been purchased before)	
	Category of item being purchased	
	Online or in-person purchase?	

### Notes:

No. If you know how many items were sold, you presumably also know which ones were sold at discounted price.

Yes. this is historical data.

Yes, provided you have the ability to get this in near-real-time.

Ditto

Yes, provided you have the ability to track this. Near-realtime behavior is not necessary; you can get close by looking at transactions till the previous month.

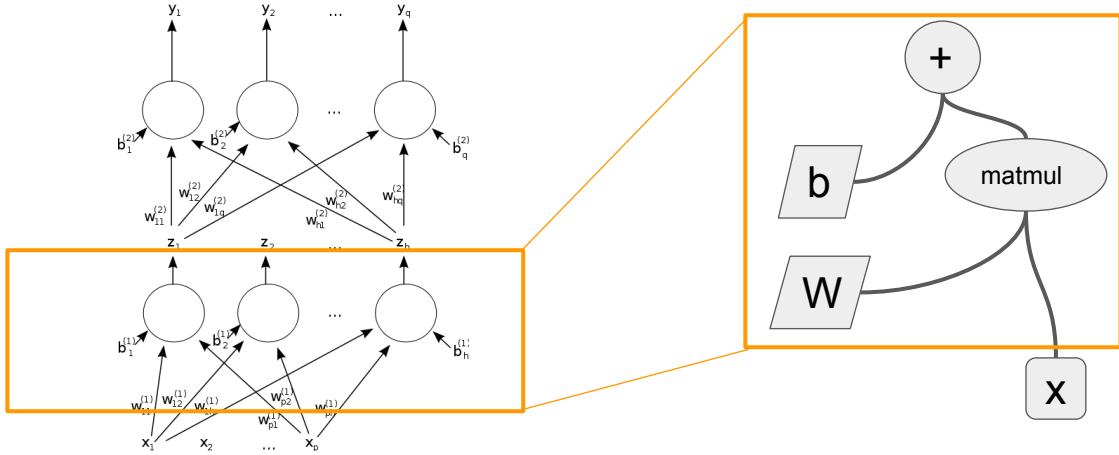
No, can a bank know this? Items get new SKUs all the time ...

Yes.

Yes, provided your data collection system supports it.

### 3. Numeric with meaningful magnitude?

A NEURAL NETWORK IS A WEIGHING AND ADDING MACHINE



# Quiz: Which of these is numeric?

Feature of discount coupon to predict number of coupons that will be used	Numeric?
Percent value of the discount (e.g. 10% off, 20% off, etc.)	
Size of the coupon (e.g. 4 cm <sup>2</sup> , 24 cm <sup>2</sup> , 48 cm <sup>2</sup> , etc.)	
Font an advertisement is in (Arial 18, Times New Roman 24, etc.)	
Color of coupon (red, black, blue, etc.)	
Item category (1 for dairy, 2 for deli, 3 for canned goods, etc.)	

**NOTE: NON-NUMERIC FEATURES CAN BE USED;  
IT'S JUST THAT WE NEED TO FIND A WAY TO  
REPRESENT THEM IN NUMERIC FORM.**



## Notes:

Percent value: yes

Size-of-coupon: yes

Font-family: no

color: no

Category: no. even though these are numbers, the magnitudes are not meaningful.

The hourglass indicates that we'll cover this later in the course.

<https://pixabay.com/en/hourglass-sandglass-patience-time-297765/> (cc0)

## 4. Enough examples

- Each value of each feature in dataset has to be understandable in context
- If you have category=auto, you must have enough transactions (fraud/no-fraud) of auto purchases

**Notes:**

<https://pixabay.com/en/autos-technology-vw-214033/> (cc0)

# Quiz: Which of these is difficult to have enough examples of?

	Feature	Good feature?
Predict total number of customers who will use a discount coupon	Percent discount of coupon (20%, 30%, etc.)	
	Date that promotional offer starts	
	Number of customers who opened advertising email	
Predict whether a credit-card transaction is fraudulent	Whether cardholder has purchased these items at this store before	
	Distance between cardholder address and store	
	Category of item being purchased	
	Online or in-person purchase?	

## Notes:

Percent discount not hard as long as we have historical sales data corresponding to different discount percentages.

Specific date is pointless. If training data is from 2015, and prediction is for 2017, then the dates in training data have no correspondence with what needs to be predicted. So, will never have enough examples. Even if we keep only the month-day part, it's too specific. ML will learn that coupons issued on Aug 12 will sell 13,298 items. That may mean nothing for Aug 12, 2016. On the other hand, coupons that start 2 weeks ahead of a festival (Easter, Diwali, Ramadan, etc.) may be meaningful if they are coded that way. For holidays that have fixed start dates (Christmas, New Year), there is some cross-pollination between two ideas but people should realize it's a function of the festival (Christmas) and not of the date (Dec 25).

No. of customers who opened mail is also hard – you'll find it hard to do A/B testing on such aggregate features, since this value will be the same for all variations of a coupon (size/font/etc.).

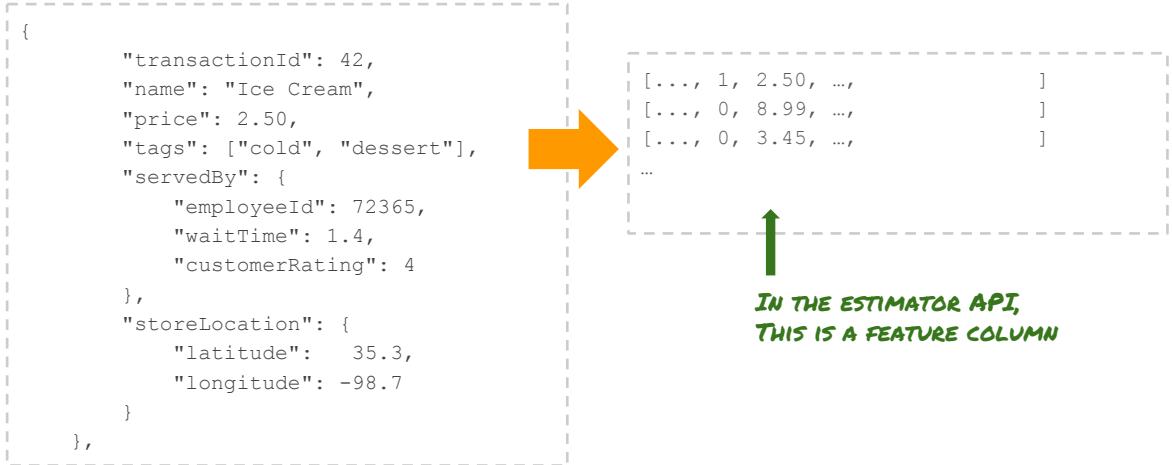
Specific customer, specific items. How will you ever have enough examples? ML will learn the combination of store and items that is fraudulent in the training data.

Specific distance of 13.223232232 km also pretty much identifies a customer.  
(Have to round off this floating point  
number perhaps to nearest 10km: called *discretization*.)

Category: good, except if the categories are overly specific. category=jewelry is fine. category=diamond tiara, not so fine.

online/in-person: good.

# Raw data are converted to numeric features in different ways



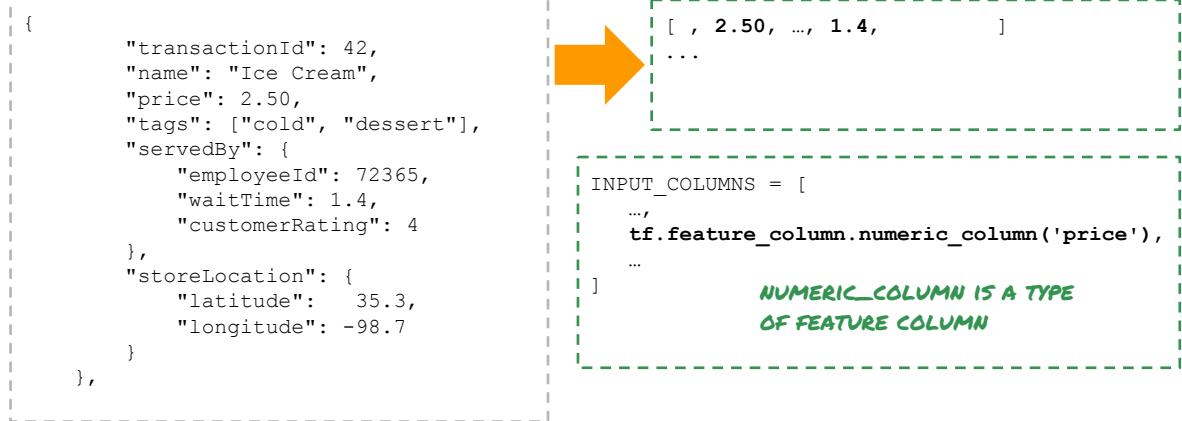
## Notes:

Assume that our raw data is JSON.

Perhaps it arrives in real-time as people purchase things.

Or maybe it's just a log file stored in a data warehouse.

## Numeric values can be used as-is



### Notes:

Assuming price in currency-units (dollars, euros, yen, etc.) and waitTime in minutes.

# Overly specific attributes should be discarded

```
{  
    "transactionId": 42,  
    "name": "Ice Cream",  
    "price": 2.50,  
    "tags": ["cold", "dessert"],  
    "servedBy": {  
        "employeeId": 72365,  
        "waitTime": 1.4,  
        "customerRating": 4  
    },  
    "storeLocation": {  
        "latitude": 35.3,  
        "longitude": -98.7  
    }  
},
```

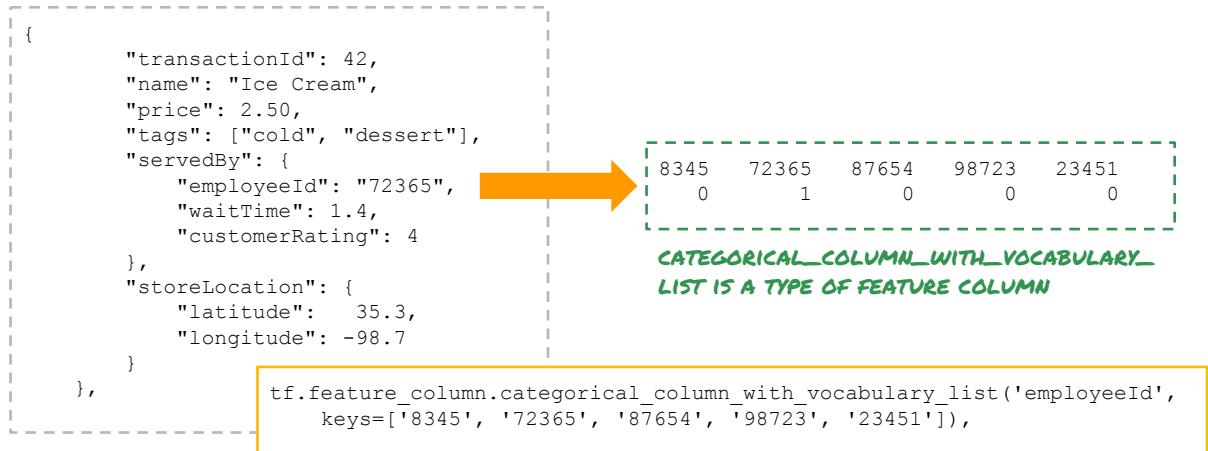


DON'T TRAIN ON IDS OR OTHER SUPER-SPECIFIC INFORMATION

## Notes:

Not enough examples. How many transactionId=42 are there in the data?

# Categorical values could be one-hot encoded



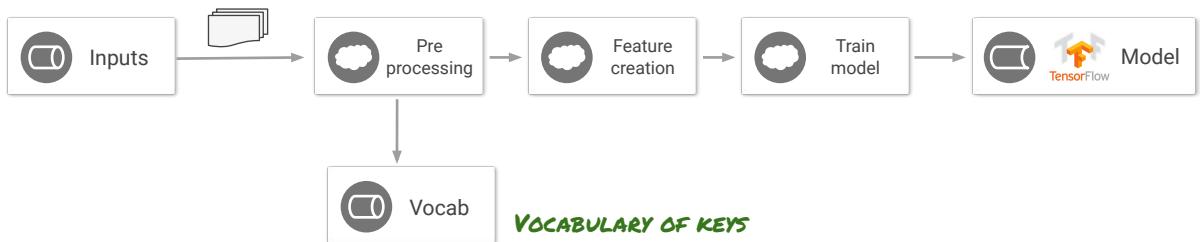
## Notes:

Shows how employeeId=72365 would get one-hot encoded assuming that there are a total of 5 employees who could be doing the serving.

Sparse-column-with-keys converts 72345 to an indexed-value (e.g., 2) and one-hot-column changes it to (0-1-0-0-0).

Similarly, all the possible tags would get coded, and 1s assigned to all valid tags.

# Preprocess data to create a vocabulary of keys



## Notes:

This is where preprocessing comes in.

# Options for encoding categorical data

THESE ARE ALL DIFFERENT WAYS TO  
CREATE A CATEGORICAL COLUMN

If you know the keys beforehand:

```
tf.feature_column.categorical_column_with_vocabulary_list('employeeId',  
    vocabulary_list = ['8345', '72345', '87654', '98723', '23451']),
```

If your data is already indexed; i.e., has integers in [0-N]:

```
tf.feature_column.categorical_column_with_identity('employeeId',  
    num_buckets = 5)
```

If you don't have a vocabulary of all possible values:

```
tf.feature_column.categorical_column_with_hash_bucket('employeeId',  
    hash_bucket_size = 500)
```

## Notes:

If you don't know how many employees you have (i.e. don't have the complete vocabulary) use hash buckets. You'll create many more hash buckets than employees (empty buckets mean nothing to the model).

Customer rating can be used as continuous or as one-hot encoded value

```
{  
    "transactionId": 42,  
    "name": "Ice Cream",  
    "price": 2.50,  
    "tags": ["cold", "dessert"],  
    "servedBy": {  
        "employeeId": 72365,  
        "waitTime": 1.4,  
        "customerRating": 4  
    },  
    "storeLocation": {  
        "latitude": 35.3,  
        "longitude": -98.7  
    }  
},
```

```
[..., 0,0,0,1,0, ...]
```

(OR)

```
[..., 4, ...]
```

### Notes:

Can be categorical (assuming ratings are in range [1,5]) or ordinal.

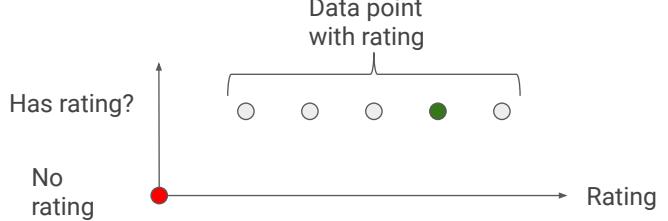
## Don't mix magic numbers with data

```
{
    "transactionId": 42,
    "name": "Ice Cream",
    "price": 2.50,
    "tags": ["cold", "dessert"],
    "servedBy": {
        "employeeId": 72365,
        "waitTime": 1.4,
        "customerRating": -1
    },
    "storeLocation": {
        "latitude": 35.3,
        "longitude": -98.7
    }
},
```

[..., 0,0,0,1,1, ...] # 4  
 [..., 0,0,0,0,0, ...] # -1

(OR)

[..., 4,1, ...] # 4  
 [..., 0,0, ...] # -1



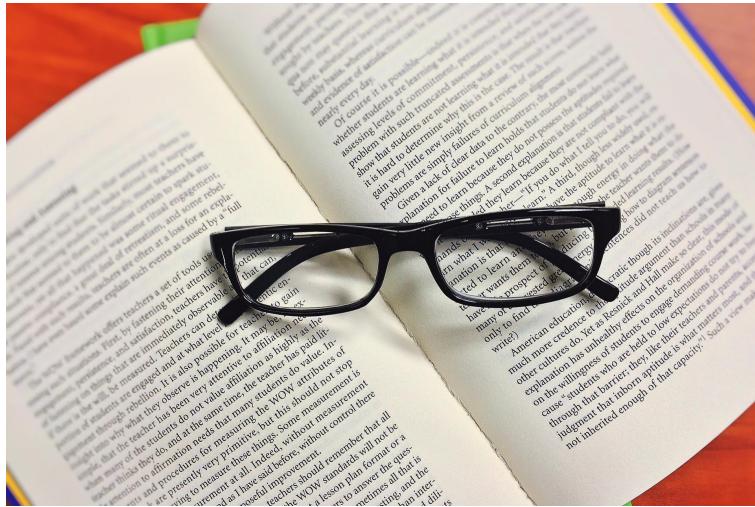
### Notes:

Suppose the customer doesn't rate the employee, then in the JSON, let's say this is coded as -1 (or that the field is omitted).

In the one-hot encoding, this is easy to handle -- make everything zero.

In normal numbers, add a second variable that is 0 or 1.

## 5. Good features bring human insight to problem



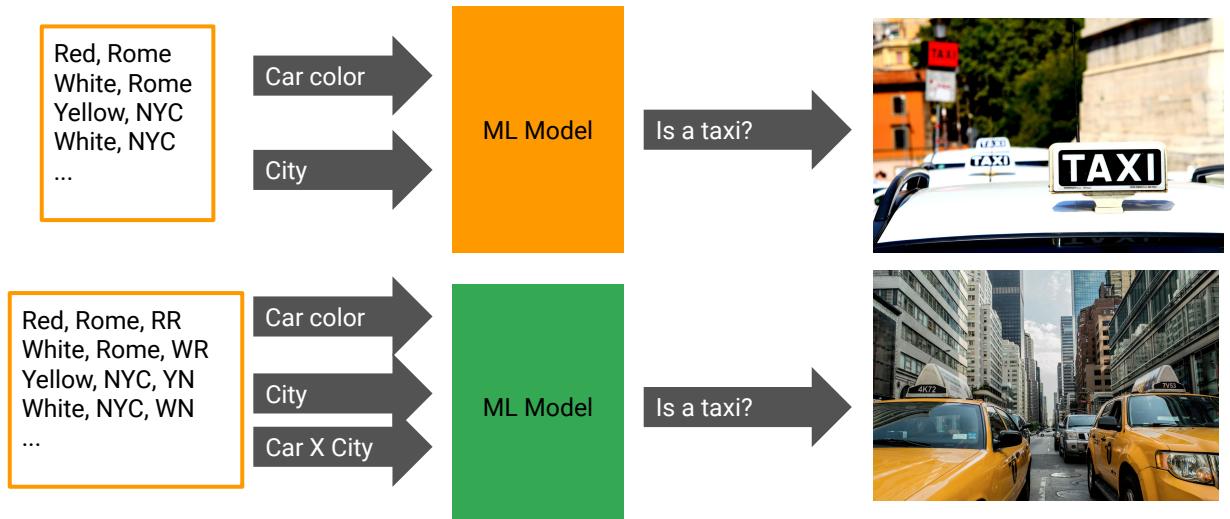
### Notes:

You may have forgotten, but this section started with a list of 5 things you need for a good feature. This is the fifth .

<https://pixabay.com/en/book-glasses-read-education-1176256/> (cc0)

Not one of those things that can be easily taught. You have to understand the problem, the objective, an ideal solution and craft features that sort of get at the form of the solution.

# Feature crosses can simplify learning



## Notes:

<https://pixabay.com/en/taxi-auto-rome-road-white-1184799/> (cc0)  
<https://pixabay.com/en/taxi-cab-traffic-cab-new-york-381233/> (cc0)

An example of insight that comes from some knowledge of the problem ... Without a feature cross, learning that white-cars in Rome and yellow-cars in New York are probably taxis is very hard for some ML models. For example, it is impossible for a linear model -- you can't assign a high weight to either white or Rome ...

RedRome is assigned category 1; WhiteRome is assigned category 2, etc. and the result is one-hot encoded. Now, you can assign a high weight to YN and WR and zeroes to everything else ... makes the ML model very easy to construct.

# Creating feature crosses using TensorFlow

YOU CAN CROSS TWO  
SPARSE COLUMNS

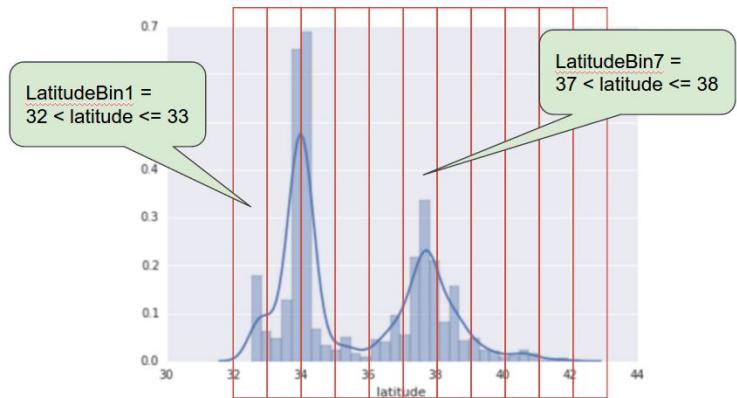
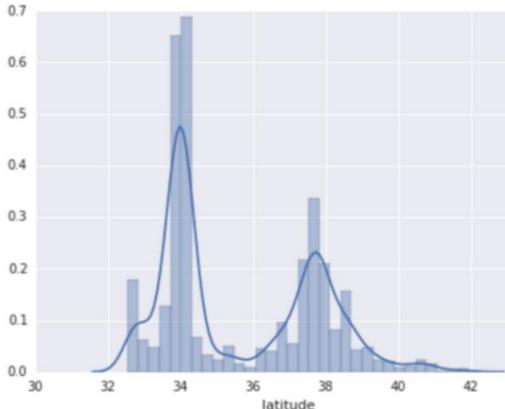
```
day_hr = tf.feature_column.crossed_column([dayofweek, hourofday], 24*7)
```

## Notes:

24\*7 is the total number of buckets

The idea is that the system can learn that Fridays at 3pm are terrible, trafficwise.

# Discretize floats that are not meaningful



## Notes:

The example here is of predicting house value. The dataset includes latitude. Two peaks – one for SFO and the other for LAX. It doesn't make sense to represent latitude as a floating-point feature in our model. That's because no linear relationship exists between latitude and housing values. For example, houses in latitude 35 are not 35/34 more expensive (or less expensive) than houses at latitude 34. And yet, individual latitudes probably are a pretty good predictor of house values.

Instead of having one floating-point feature, we now have 11 distinct boolean features (LatitudeBin1, LatitudeBin2, ..., LatitudeBin11).

Here, we used fixed bin boundaries. Another option is to use quantile boundaries so that the number of values in each bin is constant.

# Creating bucketized features using TensorFlow

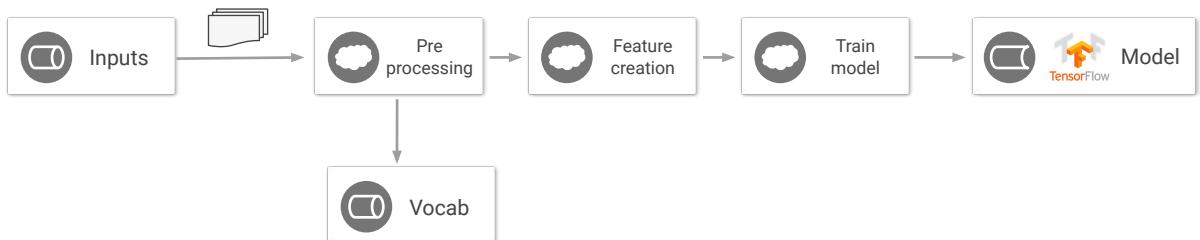
NUMBER OF BUCKETS IS  
A HYPER PARAMETER

```
latbuckets = np.linspace(32.0, 42.0, nbuckets).tolist()  
discrete_lat = tf.feature_column.bucketized_column(lat, latbuckets)
```

## Notes:

32 and 42 from previous slide

# Pipeline for bucketized and crossed features



# What's left? Ways to build effective ML



Big Data



Feature  
Engineering



Model  
Architectures

## Notes:

<https://pixabay.com/en/large-data-dataset-word-895563/> (cc0)

<https://pixabay.com/en/fractal-complexity-render-3d-1232494/> (cc0)

<https://pixabay.com/en/robot-artificial-intelligence-woman-507811/> (cc0)

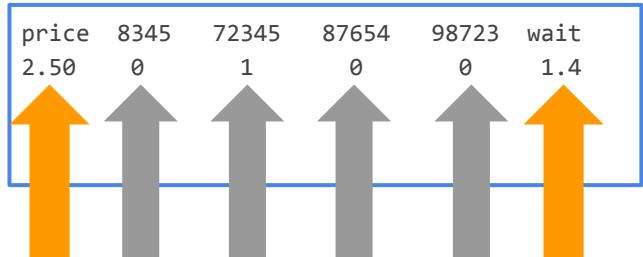
Now that you know \*how\* to build ML, let's learn how to do it well in the rest of the course.

Ordered from easiest to most difficult.

In the previous chapter, we refactored the model to make it easier to reim, but we haven't actually done anything to improve it.

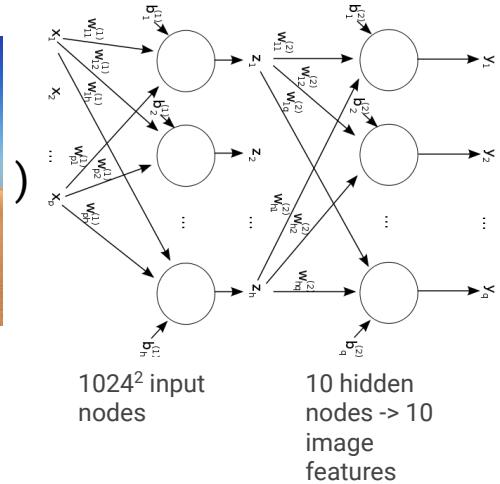
## Two types of features: Dense & sparse

```
{  
    "transactionId": 42,  
    "name": "Ice Cream",  
    "price": 2.50,  
    "tags": ["cold", "dessert"],  
    "servedBy": {  
        "employeeId": 72365,  
        "waitTime": 1.4,  
        "customerRating": 4  
    },  
    "storeLocation": {  
        "latitude": 35.3,  
        "longitude": -98.7  
    }  
},
```



DNNs good for dense, highly correlated

pixel\_values (



### Notes:

<https://pixabay.com/en/algodones-dunes-dunes-sand-dunes-1654439/> (cc0)  
[https://commons.wikimedia.org/wiki/File:Two\\_layer\\_ann.svg](https://commons.wikimedia.org/wiki/File:Two_layer_ann.svg)

Nearby pixels tend to be highly correlated, so putting them through a NN, we have the possibility that the inputs get decorrelated and mapped to a lower dimension (intuitively, this is what happens when your input layer takes each pixel value, and the number of hidden nodes is much less than the number of input nodes).

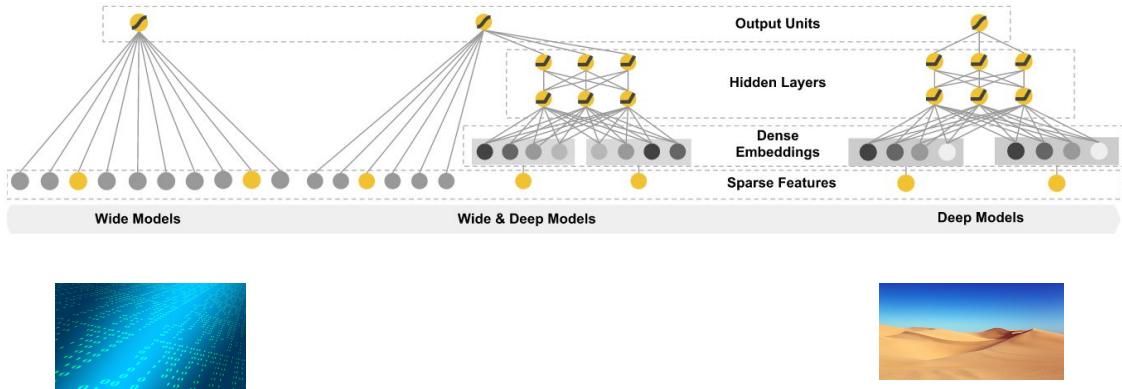
## Linear for sparse, independent features

```
[ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]  
[ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]  
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]  
[ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]  
[ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]  
[ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]  
[ 0.  0.  0.  0.  0.  0.  1.  0.  0.  0.]  
[ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]  
[ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]  
[ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
```

### Notes:

This is what a sparse matrix looks like -- very, very wide, with lots and lots of features. You want to use linear models to minimize the number of free parameters. And if the columns are independent, linear models may suffice.

# Can you have your cake & eat it too?



## Notes:

Image from

[https://www.tensorflow.org/versions/master/tutorials/wide\\_and\\_deep/index.html](https://www.tensorflow.org/versions/master/tutorials/wide_and_deep/index.html)

# Wide-and-deep network in tf.estimator

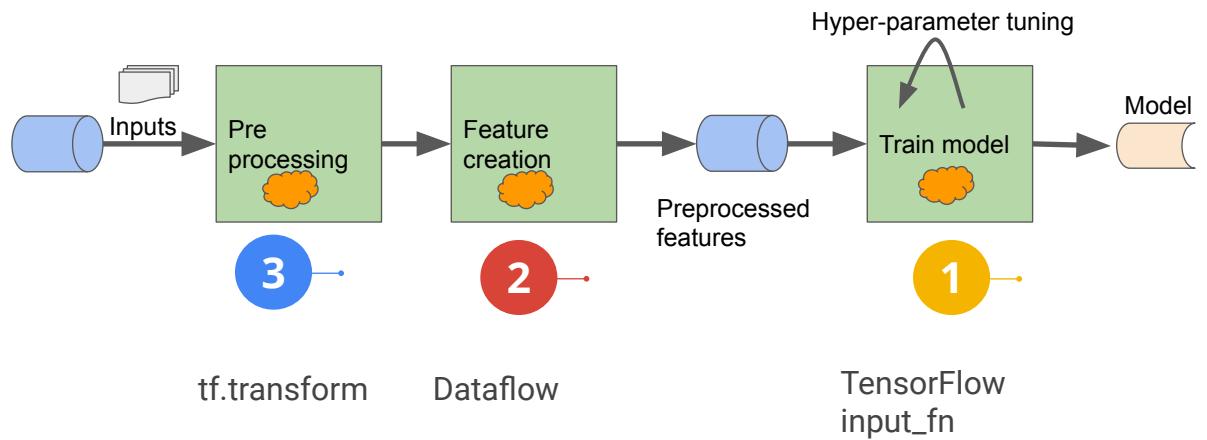
```
model = tf.estimator.DNNLinearCombinedClassifier(  
    model_dir=...,  
    linear_feature_columns=wide_columns,  
    dnn_feature_columns=deep_columns,  
    dnn_hidden_units=[100, 50])
```



## Notes:

Or you can build it layer-by-layer with `tf.layers()` – it's pretty straightforward.  
<https://pixabay.com/en/cake-piece-of-pie-black-forest-1227842/> (cc0)

## Three possible places to do feature engineering



## 1 tf.feature\_column supports some preprocessing

```
latbuckets = np.linspace(38.0, 42.0, nbuckets).tolist()
lonbuckets = np.linspace(-76.0, -72.0, nbuckets).tolist()
b_plat = fc.bucketized_column(plat, latbuckets)
b_dlat = fc.bucketized_column(dlat, latbuckets)
b_plon = fc.bucketized_column(plon, lonbuckets)
b_dlon = fc.bucketized_column(dlon, lonbuckets)
# feature cross
ploc = fc.crossed_column([b_plat, b_plon], nbuckets*nbuckets)
dloc = fc.crossed_column([b_dlat, b_dlon], nbuckets*nbuckets)
```



Supports a lot of common preprocessing steps

## 1

Cloud ML will execute your TF model for predictions, so “automatic”

### Notes:

Some not all preprocessing. For example, if you want to scale, you can't do it explicitly -- you will have to somehow know the min/max/mean/etc.

**1**

## Feature creation in TensorFlow also possible

```
def add_engineered(features):
    lat1 = features['pickuplat']
    ...
    dist = tf.sqrt(latdiff*latdiff + londiff*londiff)
    features['euclidean'] = dist
    return features
```



```
def _input_fn():
    ...
    features = dict(zip(CSV_COLUMNS, columns))
    label = features.pop(LABEL_COLUMN)
    return add_engineered(features), label
```

Can be quite powerful since it is so flexible

**1**

Will need to add call to all input functions (train, eval, serving)

### Notes:

This function takes the features you add from CSV file and then computes new ones.

## 2 → Can add new features in Dataflow

```
train = pipeline
    | beam.Read('ReadTrainingData', training_data)
    | 'addfields_train' >> beam.FlatMap(add_fields)
# adds 'passHourCount'
```

Ideal for features that involve time-windowed aggregations (streaming)

2 → You will have to compute these features in real-time pipeline for predictions (i.e., will have to use Dataflow for predictions also)

### Notes:

Example of windowed aggregation: the average number of purchases in the past one hour. In training, you can use Dataflow to compute this, but the nature of such a feature implies that you have to use Dataflow in real-time to compute that as well ...

The add\_fields in this example is a ParDo that takes the input fields, pulls out the passenger count, accumulates them, and adds the accumulated passenger count as the field 'passHourCount'.

The lab reads from BQ and writes to CSV using a Dataflow pipeline; it is into that pipeline that you would add other stuff ...

## 3 → tf.transform

BEYOND THE SCOPE  
OF THIS COURSE

### Preprocessing for Machine Learning with tf.Transform

ary + Confidential

Wednesday, February 22, 2017

Posted by Kester Tong, David Soergel, and Gus Katsiapis, Software Engineers

When applying machine learning to real world datasets, a lot of effort is required to preprocess data into a format suitable for standard machine learning models, such as neural networks. This preprocessing takes a variety of forms, from converting between formats, to tokenizing and stemming text and forming vocabularies, to performing a variety of numerical operations such as normalization.

Today we are announcing `tf.Transform`, a library for TensorFlow that allows users to define preprocessing pipelines and run these using large scale data processing frameworks, while also exporting the pipeline in a way that can be run as part of a TensorFlow graph. Users define a pipeline by composing modular Python functions, which `tf.Transform` then executes with `Apache Beam`, a framework for large-scale, efficient, distributed data processing. Apache Beam pipelines can be run on `Google Cloud Dataflow` with planned support for running with `other frameworks`. The TensorFlow graph exported by `tf.Transform` enables the preprocessing steps to be replicated when the trained model is used to make predictions, such as when serving the model with `Tensorflow Serving`.

Computes min, max, vocab, etc. and store in metadata.json

### 3 →

In serving function, use the metadata to scale the raw inputs before providing to model

<https://research.googleblog.com/2017/02/preprocessing-for-machine-learning-with.html>

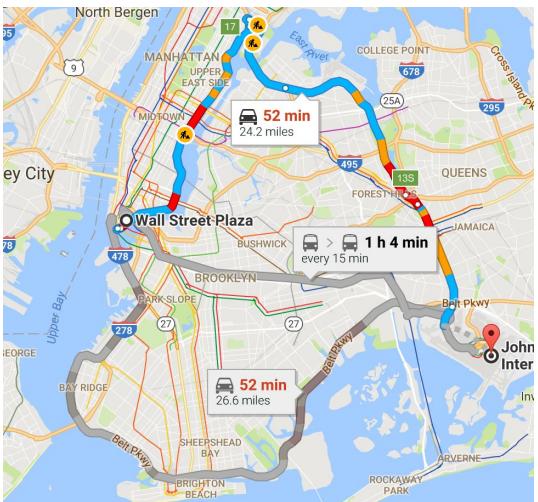
### Notes:

<https://research.googleblog.com/2017/02/preprocessing-for-machine-learning-with.html>

This is beyond the scope of this course, but here's an example notebook:

[https://github.com/GoogleCloudPlatform/training-data-analyst/tree/master/courses/machine\\_learning/feateng/tftransform.ipynb](https://github.com/GoogleCloudPlatform/training-data-analyst/tree/master/courses/machine_learning/feateng/tftransform.ipynb)

# Goal: To estimate taxi fare



Taxi fares:



\$2.50 initial charge  
+  
50c per  $\frac{1}{5}$  mile  
(or)  
50c per minute if stopped  
+  
Passenger pays tolls  
+  
Various special charges

[http://www.nyc.gov/html/tlc/html/passenger/taxicab\\_rate.shtml](http://www.nyc.gov/html/tlc/html/passenger/taxicab_rate.shtml)

## Notes:

Left: a trip from Hanover Square to Time Square (downtown to midtown). Two routes – the shorter one takes longer.

Right: a trip from JFK to Manhattan offers 2 routes, both very roundabout as compared to what a bus would take. You pay a toll to take the triborough bridge.

But assume we don't know any of these things. Instead of hardcoding a bunch of rules, let's try to infer the fare amount simply from the data.

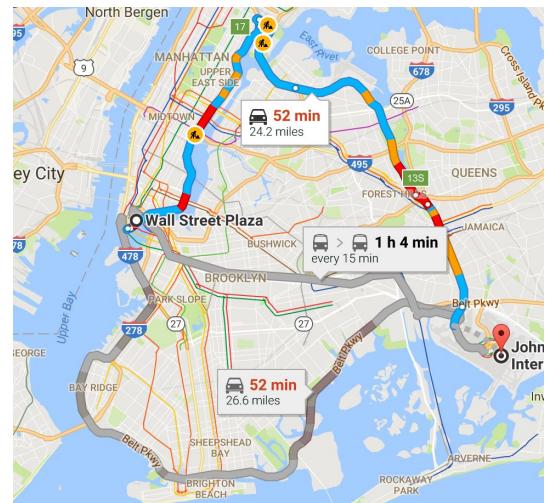
<https://pixabay.com/en/question-mark-question-faq-1421013/> (cc0)

## Discussion: ML framing

Can we use as input:

1. Trip distance/duration
2. Timestamp
3. Hour of day
4. Day of week
5. Latitude, Longitude
6. Other datasets?

Directly, or after transform?



### Notes:

1. No, they are not knowable at prediction time.
2. No, not directly. There are two issues: (a) Those specific timestamps will not be present at prediction time. We should pull out day-of-week and hour-of-day perhaps. (b) We need to pull those from (timestamp - duration) because the timestamp is the time-of-report, which is after the actual dropoff happened! At the time of prediction, we know only the pickup time, not the dropoff time.
3. Sure, although it may be better to quantize it into bins (morning rush hour, mid-day, evening rush hour, night/morning) and then one-hot encode these categories). This way, we don't get into the problem that 5pm is very much like 6pm, but is not "less" than it.
4. One-hot encode the day of week, but also add weekend vs. weekday. If we have access to list of NY holidays, we should use that too.
5. Yes (we need it for precise distances, which blocks, etc.), but they are not useful in isolation. We could add new features around binning by neighborhood (polygon boundaries exist), proximity to specific bridges and highways, side of street (for one-ways) etc. Something that is simpler is to bucketize into 0.1 degrees and do a feature cross (which when you think about it is essentially putting the lat/lon points into grid-cells).

1. Holiday data, weather data, common routes, landmarks (airports, etc.), location of streets.

Classification or regression is actually an interesting question. The knee-jerk reaction is to say “regression” because we are predicting cost. However, a case can be made that this is a classification problem -- you see that there are two possible fares from JFK to Wall Street. One goes through the tri-borough bridge, involves a toll, takes 52 minutes and 24.2 miles.

The other goes through Lincoln Tunnel and has different characteristics. You might think of predicting cost as a classification problem, so that you get probabilities for both routes. In this case, you'd have to bin the cost into categories (2.50-7.50, 7.50-12.50, etc.) and predict the likelihood of that category.

Please take 5 minutes to give us feedback

<https://goo.gl/uqfGRa>

## Lab 7: Feature Engineering

In this lab you will incorporate feature engineering into the pipeline.

1. Working with feature columns
2. Adding feature crosses in TensorFlow
3. Reading data from BigQuery
4. Creating datasets using Dataflow
5. Using a wide-and-deep model

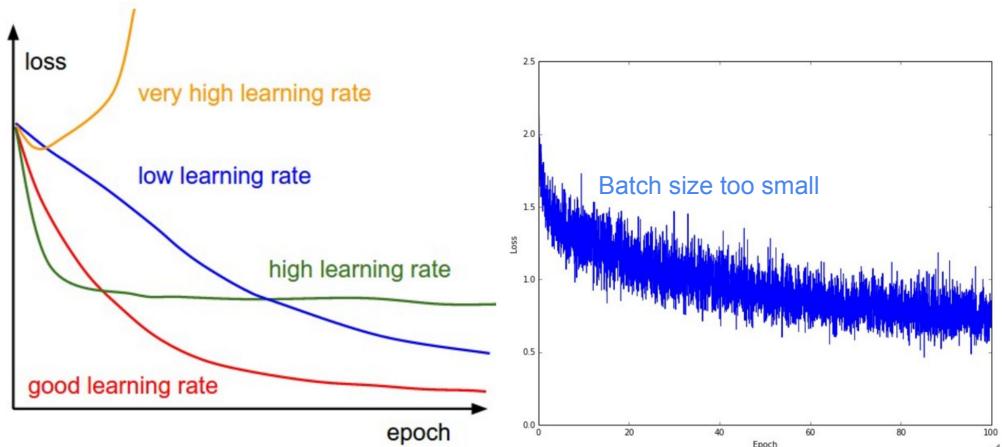
**Notes:**

<https://pixabay.com/en/book-glasses-read-education-1176256/> (cc0)

# Agenda

Hyperparameter tuning + Demo

## Very sensitive to batch\_size and learning\_rate



Source: <http://cs231n.github.io/neural-networks-3/> by Andrej Karpathy

### Notes:

Learning rate, batch-size matter. These graphs are by epoch, but unfortunately, TF doesn't know much about epochs. You'll have to figure out the epoch based on calculating how many steps of batch-size each will end up traversing our dataset once.

At low learning rates, improvement is linear, but you tend to not get the best possible performance.

At high learning rates, improvement is exponential initially, but again it's not great.

There's often a goldilocks learning rate, but good luck finding it ...

The hyperlink is good reading. Please read it.

## There are a variety of model parameters too

- Size of model
- Number of hash buckets
- Embedding size
- Etc.

WOULDN'T IT BE NICE TO HAVE THE NN TRAINING  
LOOP DO META-TRAINING ACROSS ALL  
THESE PARAMETERS?



### Notes:

<https://pixabay.com/en/boy-idea-sad-eyes-school-thinking-1867332/> (cc0)

# Cloud MLE supports hyperparameter tuning

1. Make the parameter a command-line argument
2. Make sure outputs don't clobber each other
3. Supply hyperparameters to training job



# 1. Make parameter a command-line argument and use it in model

```
parser.add_argument(  
    '--nbuckets',  
    help='Number of buckets into which to discretize lats and lons',  
    default=10,  
    type=int  
)  
parser.add_argument(  
    '--hidden_units',  
    help='List of hidden layer sizes to use for DNN feature  
columns',  
    default="128 32 4"  
)
```

## 2. Make sure that outputs don't clobber each other

```
output_dir = os.path.join(  
    output_dir,  
    json.loads(  
        os.environ.get('TF_CONFIG', '{}')  
    ).get('task', {}).get('trial', '')  
)
```



Name	Size
checkpoint	132 B
eval/	—
events.out.tfevents.1488250047.master-2d5cef50bf-0-...	3.25 MB
export/	—
graph.pbtxt	1.47 MB
model.ckpt-0.data-00000-of-00003	9.28 MB
model.ckpt-0.data-00001-of-00003	532.07 KB

### 3. Supply hyperparameters to training

```
%writefile hyperparam.yaml
trainingInput:
  scaleTier: STANDARD_1
hyperparameters:
  goal: MINIMIZE
  maxTrials: 30
  maxParallelTrials: 1
  hyperparameterMetricTag: rmse
  params:
    - parameterName: train_batch_size
      type: INTEGER
      minValue: 64
      maxValue: 512
      scaleType: UNIT_LOG_SCALE
    - parameterName: nbuckets
      type: INTEGER
      minValue: 10
      maxValue: 20
      scaleType: UNIT_LINEAR_SCALE
    - parameterName: hidden_units
      type: CATEGORICAL
      categoricalValues: ["128 64 32", "256 128 16", "512 128 64"]
```

```
gcloud ml-engine jobs submit training $JOBNAME \
\ --region=$REGION \
--module-name=trainer.task \
...
--config=hyperparam.yaml \
-- \
--output_dir=$OUTDIR \
--train_steps=1000
```

#### Notes:

Minimize the RMSE on the evaluation dataset in this example.

Systematically try various neural network architectures and different processing methods.

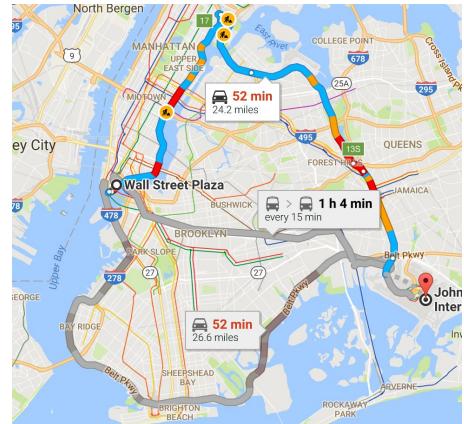
Make it a command-line parameter of the training program.

Cloud ML will algorithmically search in promising areas.

# Demo: Hyperparameter tuning

- This notebook demonstrates model & training modifications to support hyper parameter tuning

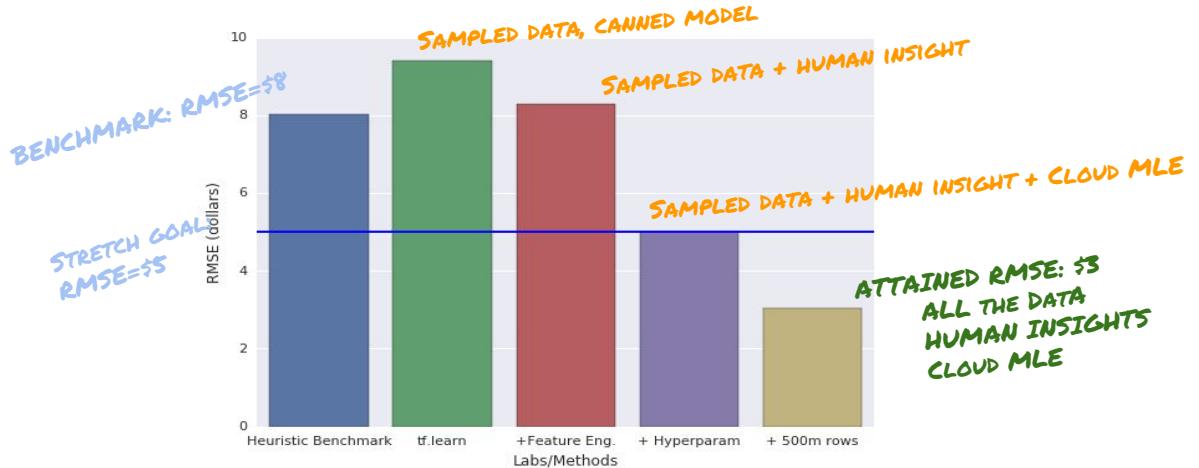
[https://github.com/GoogleCloudPlatform/training-data-analyst/tree/master/courses/machine\\_learning/feateng/hyperparam.ipynb](https://github.com/GoogleCloudPlatform/training-data-analyst/tree/master/courses/machine_learning/feateng/hyperparam.ipynb)



## Notes:

This takes several hours to run, so just a demo.

# Accuracy improves through feature engineering, hyperparameter tuning, and lots of data



# Agenda

ML Abstraction Levels

# The ML marketplace is moving towards increasing levels of ML abstraction

Custom image model to price cars

**AUCNET**

Build off NLP API to route customer emails

**ocado**

Use Vision API as-is to find text in memes

**GIPHY**

Use Dialogflow to create a new shopping experience

**UNIQLO**

Google Cloud

Training and Certification

Aucnet built their own custom model to classify car parts and estimate price.

Ocado used parsed results from the NL API to route customer emails

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.

Aucnet:

<https://cloud.google.com/blog/big-data/2017/08/how-aucnet-leveraged-tensorflow-to-transform-their-it-engineers-into-machine-learning-engineers>

Ocado:

<https://cloud.google.com/customers/ocado/>

Giphy:

<https://cloud.google.com/blog/big-data/2017/12/building-a-better-giphy-with-google-clouds-machine-learning-tools>

UNIQLO:

<https://cloud.google.com/dialogflow-enterprise/>

Note: The company names on this slide are just colored text for visual interest and are not company logos or intended to represent company logos.

## Pre-built AI building blocks



Google Cloud

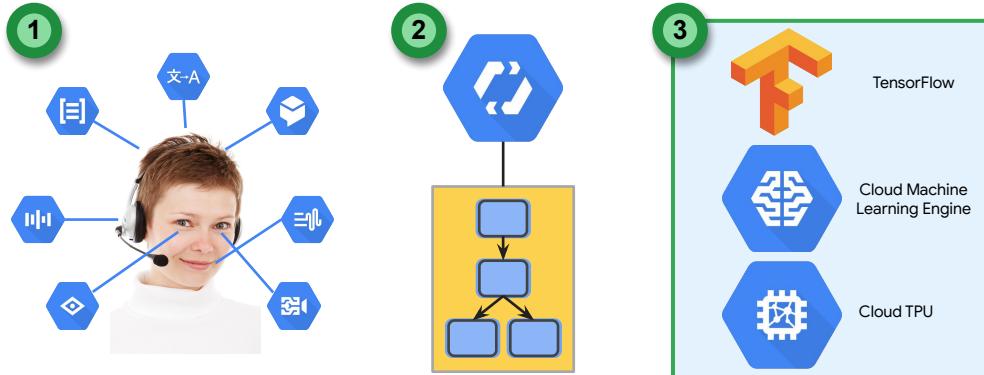
Training and Certification

Pre-built models are offered as services. In many cases these building blocks can be used to create the application you want without the expense or complexity of creating your own models.

Cloud Speech-to-Text converts audio to text for data processing. Cloud Natural Language API recognizes parts of speech called entities and sentiment. Cloud Translation converts text in one language to another. Dialogflow Enterprise Edition is used to build chatbots to conduct conversations. Cloud Text-to-Speech converts text into high quality voice audio. Cloud Vision API is for working with and recognizing content in still images. And Cloud Video Intelligence API is for recognizing motion and action in video.

<https://pixabay.com/photos/agent-business-call-center-18762/>

# Artificial Intelligence application strategy

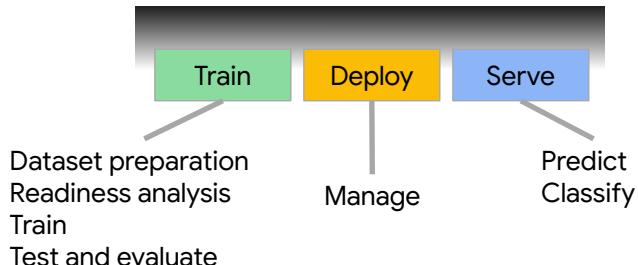
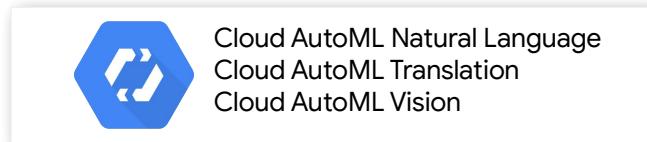


Google Cloud

Training and Certification

<https://pixabay.com/photos/agent-business-call-center-18762/>

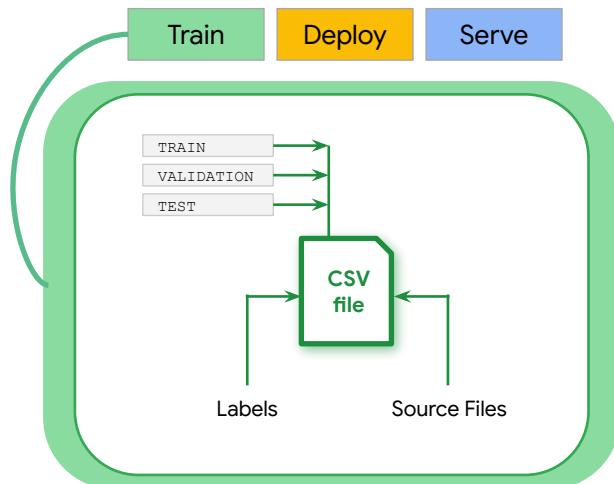
# Cloud AutoML Overview



Google Cloud

Training and Certification

# Creating Prepared Datasets for Cloud AutoML training



Google Cloud

Training and Certification

Cloud AutoML uses a Prepared Dataset to train a Custom Model. You can make small Prepared Datasets for experimentation directly in the Web UI but it is more common to assemble the information in a CSV (comma separated value) file. The CSV file must be UTF-8 encoded and located in the same Cloud Storage bucket with the source files. You can also create and manage Prepared Datasets programmatically in Python, Java, or Node.js.

The first column in the CSV file is optional. It assigns the data in each row into one of three groups, TRAIN, VALIDATION, or TEST. If you leave out this column, the rows will automatically be assigned with 80% going to TRAIN, and 10% each to VALIDATION and TEST.

The next column in the CSV file identifies source files that are hosted in Cloud Storage. These are paths beginning with "gs://..."

The source file format depends on the kind of model you are training but can also be compressed ZIP files.

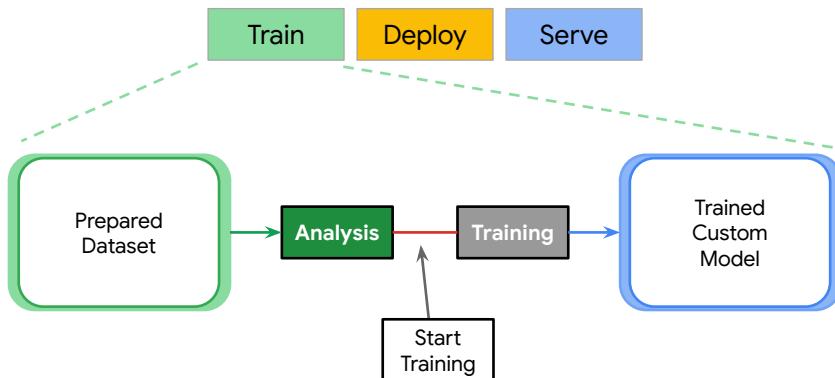
Subsequent columns specify labels. The labels are alphanumeric and can contain underscores, but not special characters.

The CSV file should not contain duplicate lines and may not contain blank lines or unicode characters.

Currently, the CSV file and all the Source Files must be in a Cloud Storage bucket in the project where AutoML runs.

Prepared Datasets do not expire. You may accumulate many Prepared Datasets in a project. You can list and delete those you don't need.

# Training a Custom Model from a Prepared Dataset



Google Cloud

Training and Certification

Cloud AutoML performs basic checks and a preliminary analysis of the Prepared Dataset to determine if there is enough information and if it is properly organized. If the Prepared Dataset is not ready, you will need to add more rows or more labels to the CSV file. When it is ready, you can start training.

Training can take from ten minutes to several hours depending on the kind of model. You can check the status while it is running. Import and training tasks can be canceled.

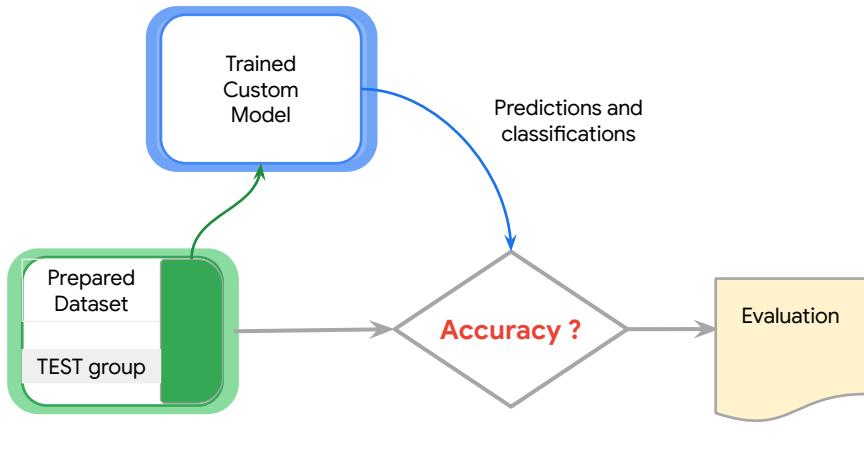
The TRAIN group of data is used to train the Custom Model. The source files have already been associated with the correct labels in the Prepared Dataset, so Cloud AutoML uses a supervised learning method to train the Custom Model. Part of the process uses the VALIDATION group data to verify how well the model works at classifying and predicting.

Supervised learning works on correctable error. Cloud AutoML constructs an algorithm that guesses the labels for source data. When the guess is right, it strengthens the algorithm. When the guess is wrong, the error is used to correct the algorithm. And this is how learning occurs. One full run through all the TRAIN group data is called an epoch. Total error is tracked and minimized through multiple epochs to create the best model possible from the training data provided.

The result is a trained Custom Model.

The custom model works well with the training data. But is it good at categorizing new instances of data it has not seen before?

# Testing and evaluating the custom model



Google Cloud

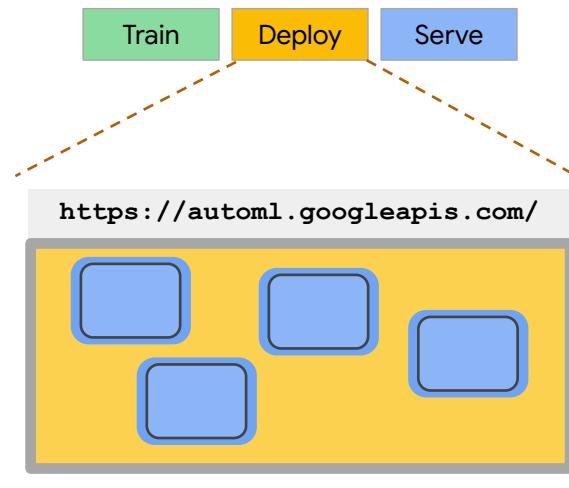
Training and Certification

Data from the TEST group is used to evaluate the Custom Model and to remove bias from the evaluation.

The predictions and classifications are compared with the labels in the Prepared Dataset.

The evaluation report provides indicators that are specific to the kind of model and help understand how effective the model is at predicting and classifying.

# Deploying the Custom Model



There is nothing you need to do to activate a model.  
However, if it has been some time since you used a model, the system may need to "warm up" for a few minutes before the model becomes active.

Once it exists, if you have the project credentials and model-name you can access and use the Custom Model.

Each time you train with a Prepared Dataset it creates a new Custom Model.

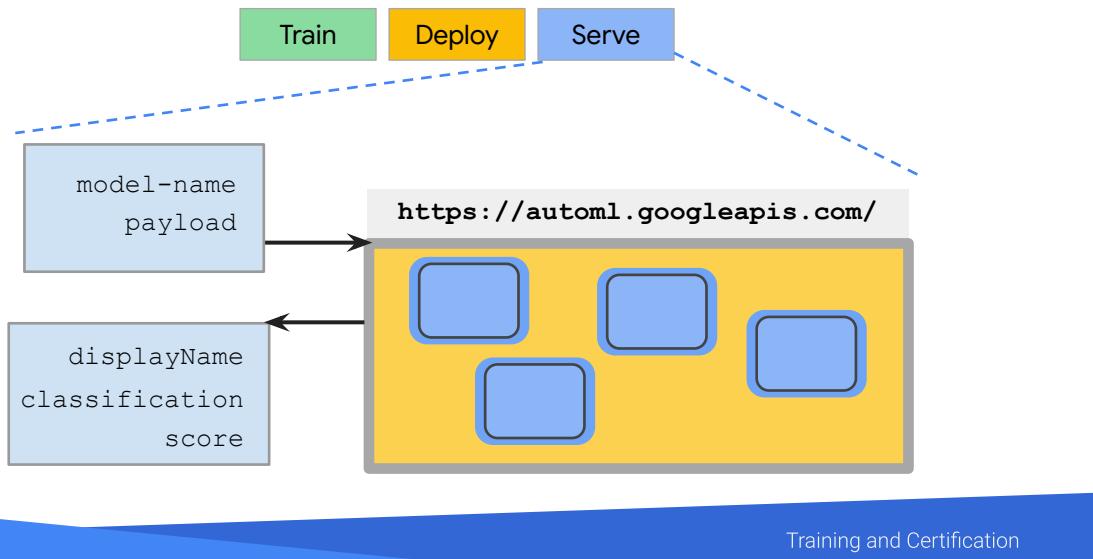
You can list and delete unneeded models.

Custom Models are temporary. They are eventually deleted. And they cannot be exported or saved externally.

Models that **are not used** for prediction are automatically deleted after a period. And models that **are used** are eventually deleted. So you will need to train a new Custom Model periodically to continue predicting and classifying.

How long models remain before they are deleted depends on the model type.

# Serving and Using Custom Models



The primary classification interface is at the URI shown. You can make a classification using the Web UI, or from the command line using CURL to send a JSON-structured request. There are also client libraries for Python, Java, and Node.JS.

After you have set up authentication to use the REST API, you send a request with the model-name and the payload, which is the data you want classified. The service returns JSON containing multiple fields called displayName. These are the labels that matched. Then it contains the keyword classification, followed by a score. The score is a confidence value, where 1.0 is absolute confidence, and lower fractional numbers represent lower confidence in the correctness of the classification.

Quotas apply for both model creation and service requests.

# Cloud AutoML services



Cloud AutoML  
Natural Language



Cloud AutoML  
Translation



Cloud AutoML Vision

Google Cloud

Training and Certification

Cloud AutoML Natural Language classifies English content into custom categories.  
Cloud AutoML Translation enables models that translate into words and phrases specific to your domain.

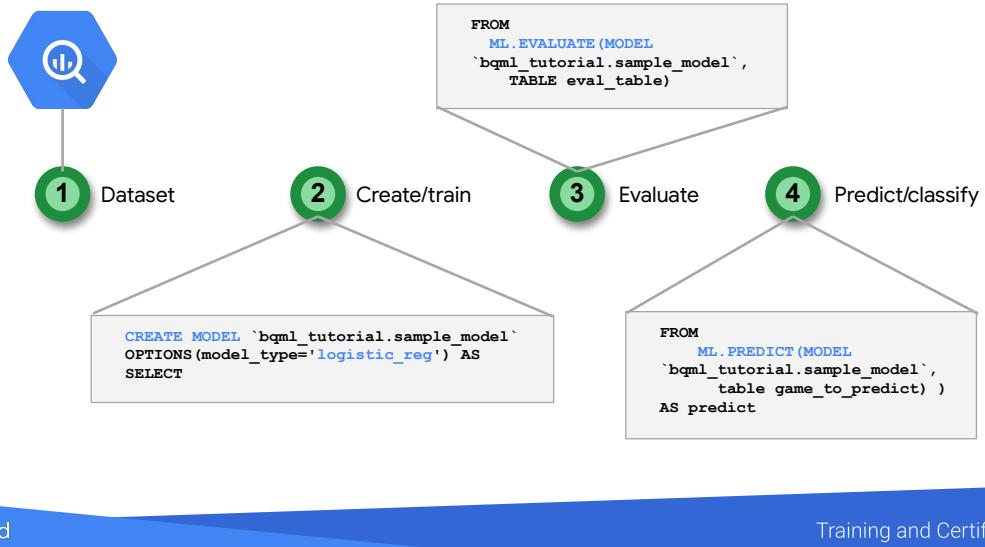
Cloud AutoML Vision classifies images according to your defined labels.

<https://pixabay.com/photos/learn-word-scrabble-letters-wooden-1820039/>

<https://pixabay.com/illustrations/globe-world-languages-translate-110775/>

<https://pixabay.com/illustrations/lens-colorful-background-digital-582605/>

# BigQuery ML



Google Cloud

Training and Certification

There are two key benefits of BigQuery ML. First, it works on your data that is already in BigQuery storage -- in your Data Warehouse or Data Lake.

So the data does not need to be copied out of BigQuery or reformatted. This improves efficiency in many ways, and in some cases resolves privacy concerns.

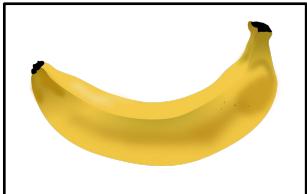
Second, BigQuery ML uses SQL commands. That means BigQuery users can learn to create their own models by extending their use of a language they already know.

Works with full models (not Custom Models like Cloud AutoML).

The locality of your model is the same as your dataset. So that is defined when the dataset is created.

# BigQuery ML supports these machine learning models

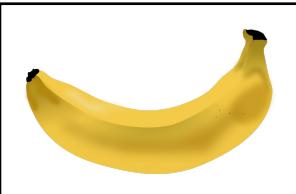
Linear regression



How much does  
this weigh?

0.13 kg

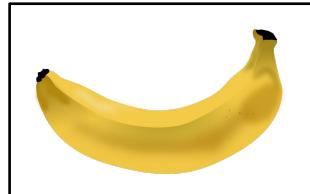
Binary logistic regression



Is this a banana?

YES or NO

Multiclass logistic regression



Which fruit is this?

Apple  
Orange  
Peach  
**Banana**

Google Cloud

Training and Certification

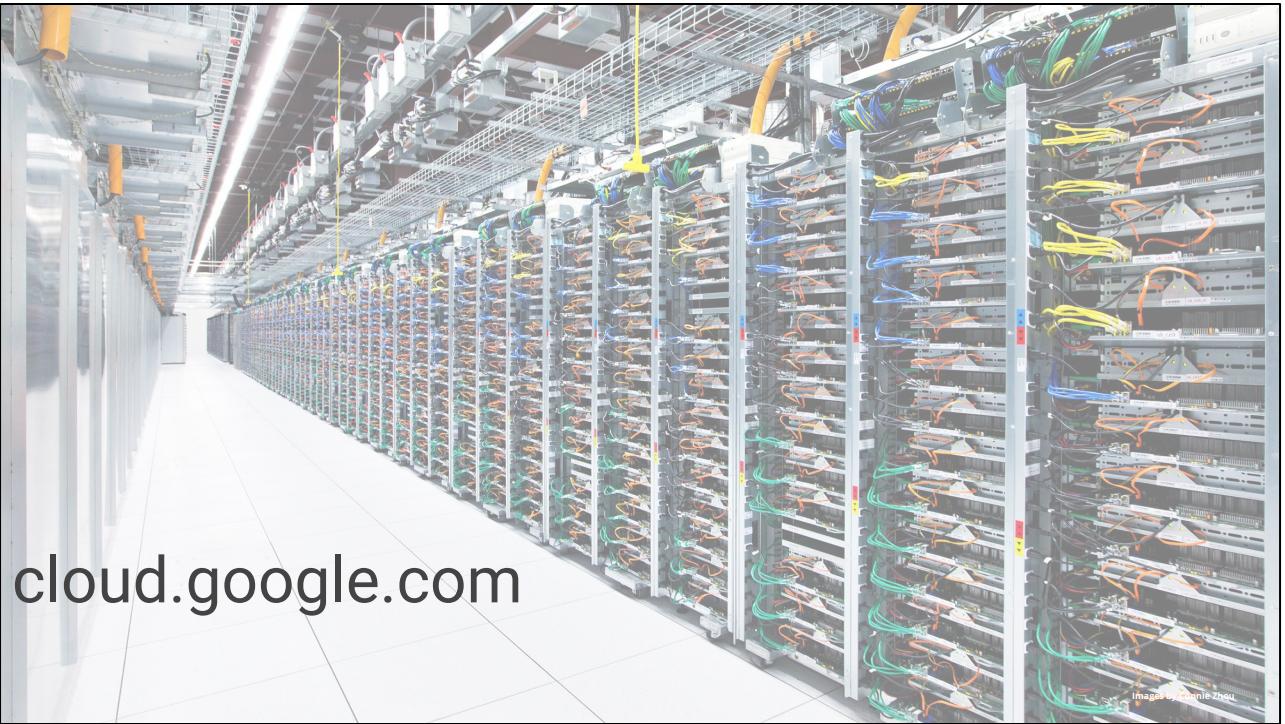
Linear regression models predict a numerical value. Binary logistic regression models identify one of two classes. Multiclass logistic regression models are used for classification.

You can read about currently supported models in the online documentation.

<https://pixabay.com/vectors/banana-fruit-yellow-fresh-healthy-310449/>

# Resources

Cloud AutoML	<a href="https://cloud.google.com/automl/">https://cloud.google.com/automl/</a>
Feature Engineering for KDD Cup 2010	<a href="http://pslcdatahop.org/KDDCup/workshop/papers/kdd2010ntu.pdf">http://pslcdatahop.org/KDDCup/workshop/papers/kdd2010ntu.pdf</a>
Stanford CS class on convolutional neural networks	<a href="http://cs231n.github.io/">http://cs231n.github.io/</a>



cloud.google.com