Google Cloud

Improving ML through feature engineering

Data Engineering on Google Cloud Platform

Notes:

50 slides + 1 lab + 1 demo: 3 hour

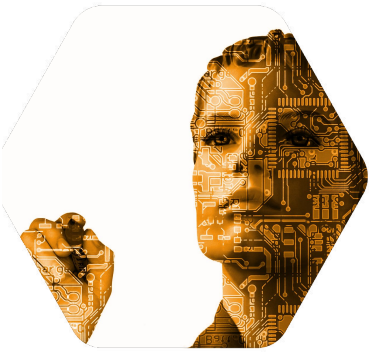
Agenda

Creating good features + Lab

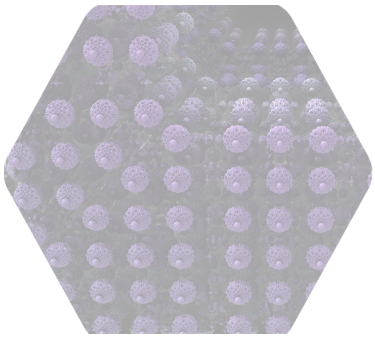
Hyperparameter tuning + Demo

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 eim, 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.

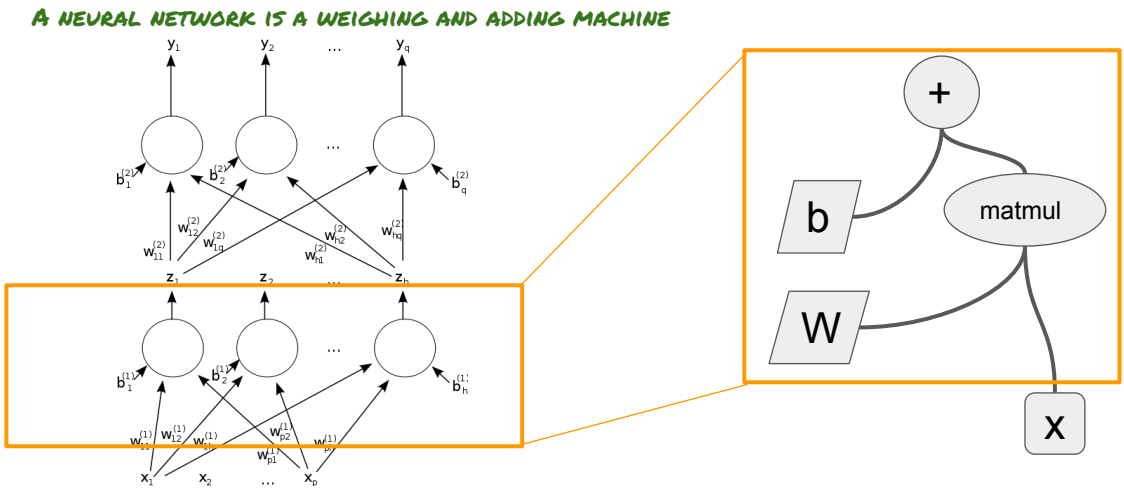
Quiz: Value knowable at prediction time or not?

	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?



Notes:

Image source: https://commons.wikimedia.org/wiki/File:Two_layer_ann.svg

Left: a neural network.

Right: the input layer essentially consists of doing a matrix multiplication. You have to have numbers. 42 is okay. “Red” is not.

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^2, 24 cm^2, 48 cm^2, 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.).

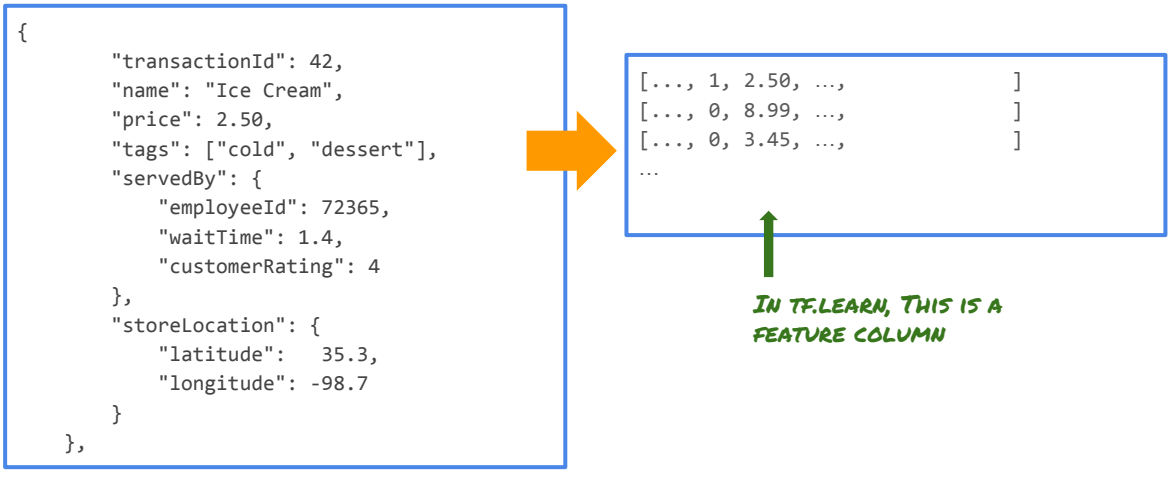
Specific customer, specific items. How will you ever have enough examples? ML will learn the combination of store and items that is fraudulent in the training data.

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 to numeric features



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.

Some values can be used as-is

```
{
        "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
        }
    },
```

```
[ , 2.50, …, 1.4,          ]
…
```

```
INPUT_COLUMNS = [
   …,
   layers.real_valued_column('price'),
   …
]
```

REAL-VALUED-COLUMN IS A TYPE OF FEATURE COLUMN

Notes:

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

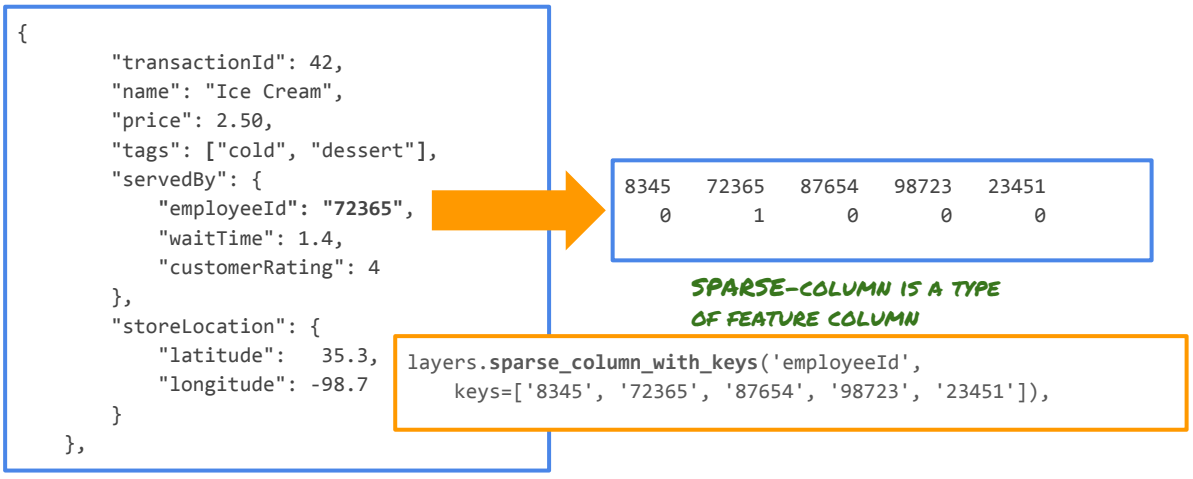
Some attributes are too specific

```
{
        "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
        }
    },
```

Notes:

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

One-hot encode categorical strings



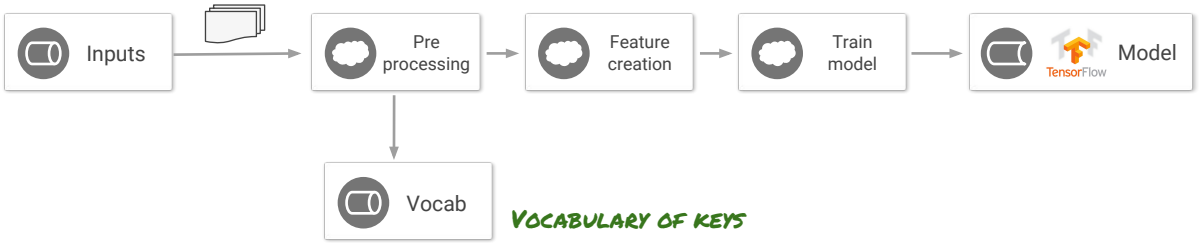
Notes:

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 sparse-column

If you know the keys beforehand:

```
layers.sparse_column_with_keys('employeeId',
    keys=['8345', '72345', '87654', '98723', '23451']),
```

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

```
layers.sparse_column_with_integerized_feature('employeeId', 5)
```

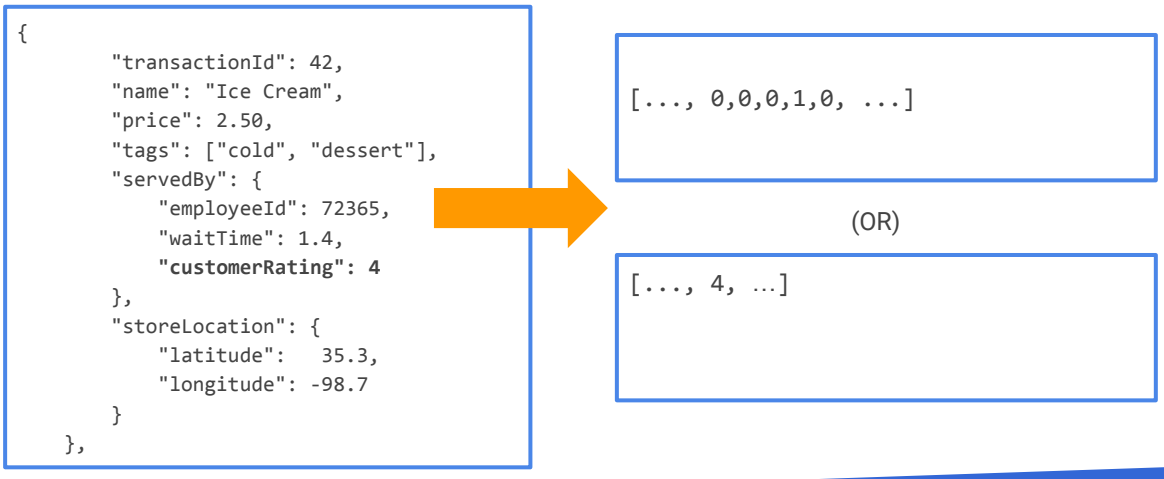
If you don't have a vocabulary of all possible values:

```
layers.sparse_column_hashbucket('employeeId', 500)
```


Notes:

one-hot

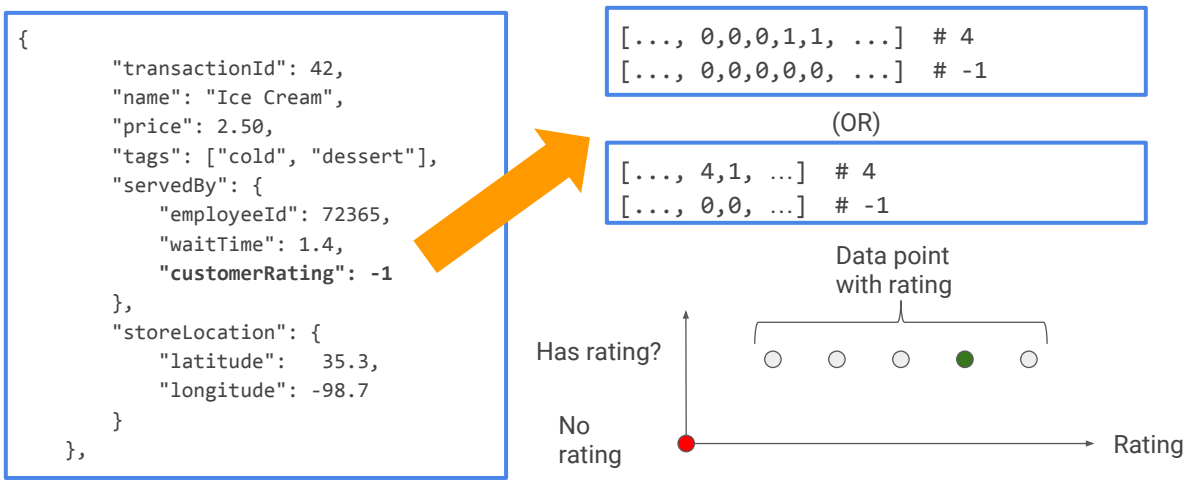
Customer rating can be used as continuous or as one-hot encoded value



Notes:

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

Don’t mix magic numbers with data



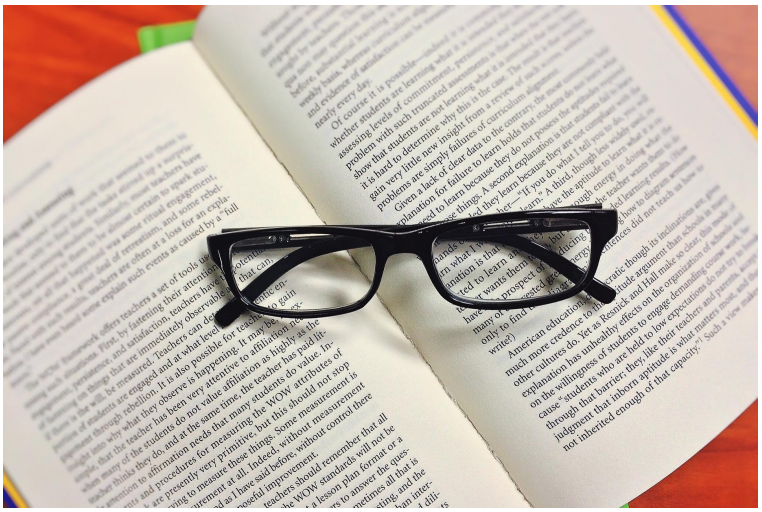
Notes:

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

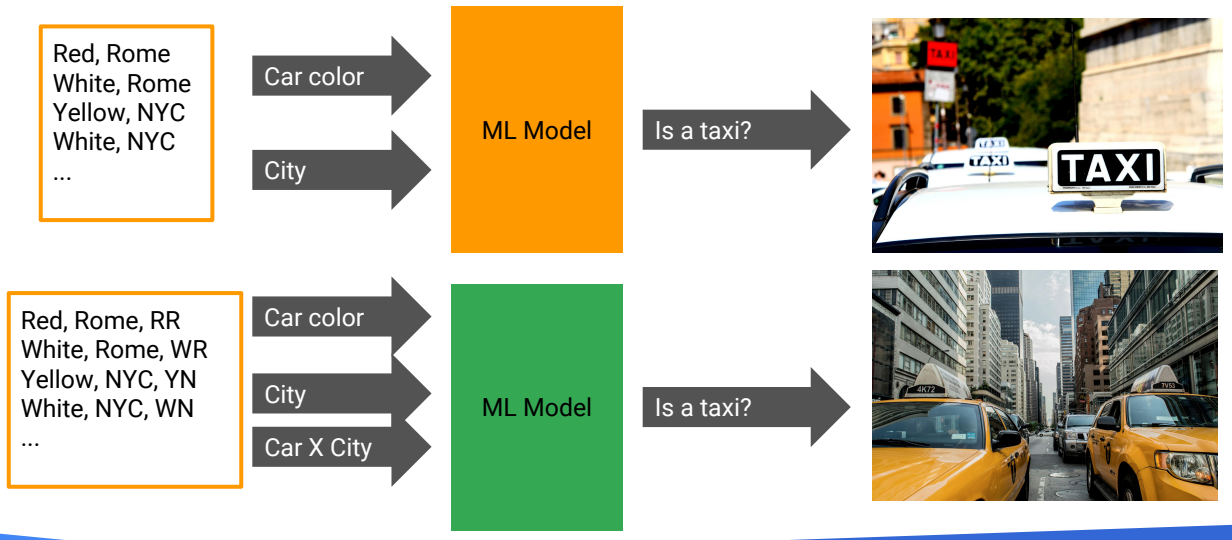


Notes:

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 tf.layers

You can cross two sparse columns

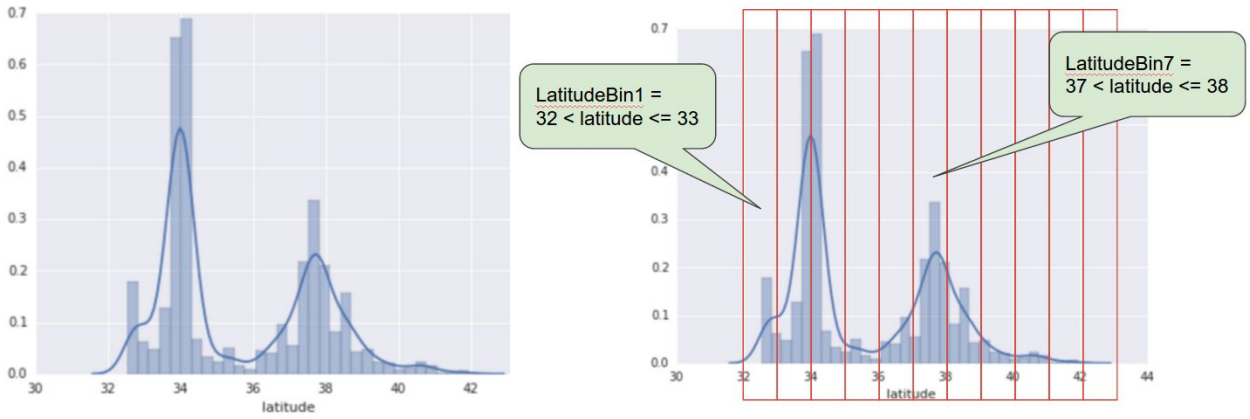
```
day_hr =  layers.crossed_column([dayofweek, hourofday], 24*7)
```


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 tf.layers

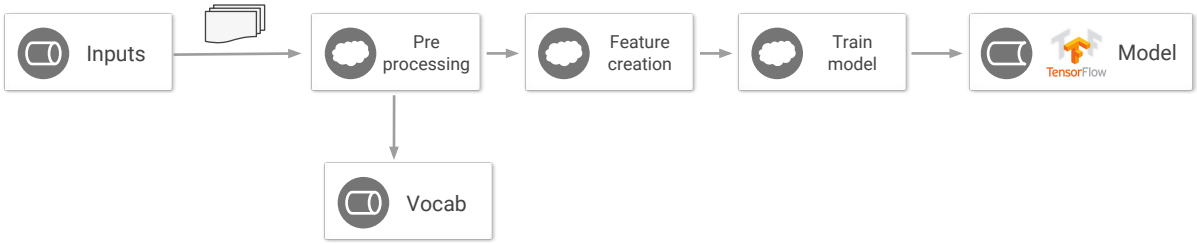
NUMBER OF BUCKETS IS A HYPER PARAMETER

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


Notes:

32 and 42 from previous slide

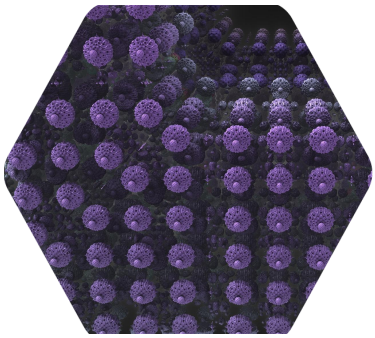
Pipeline for bucketized and crossed features



What's left? Ways to build effective ML

Big Data

Feature Engineering



Model Architectures

Notes:

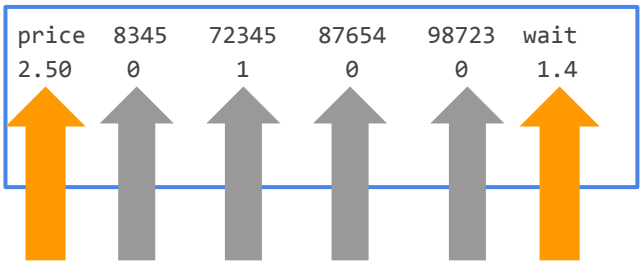
https://pixabay.com/en/large-data-dataset-word-895563/ (cc0)
https://pixabay.com/en/fractal-complexity-render-3d-1232494/ (cc0)
https://pixabay.com/en/robot-artificial-intelligence-woman-507811/ (cc0)

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 eim, but we haven't actually done anything to improve it.

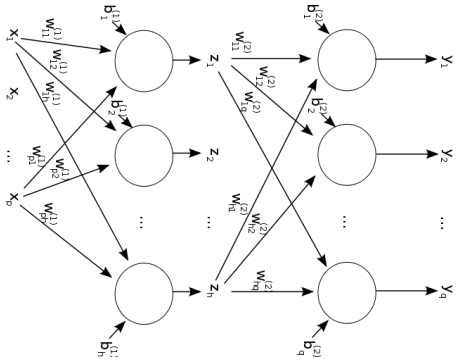
Two types of features: Dense & sparse

```
{
        "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 ()



1024^2 input nodes

10 hidden nodes -> 10 image features

Notes:

https://pixabay.com/en/algodones-dunes-dunes-sand-dunes-1654439/ (cc0)
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).

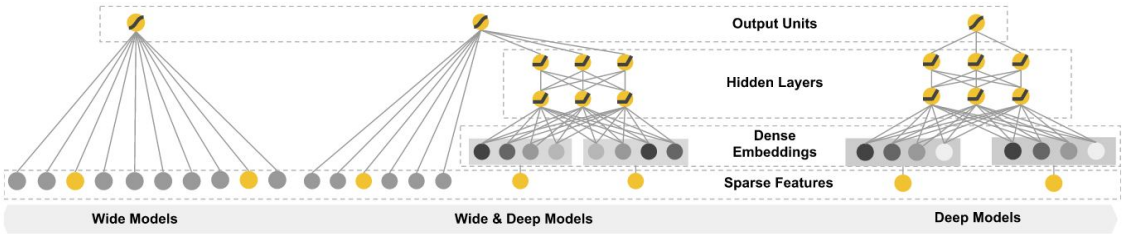
Linear for sparse, independent features

```
[ 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

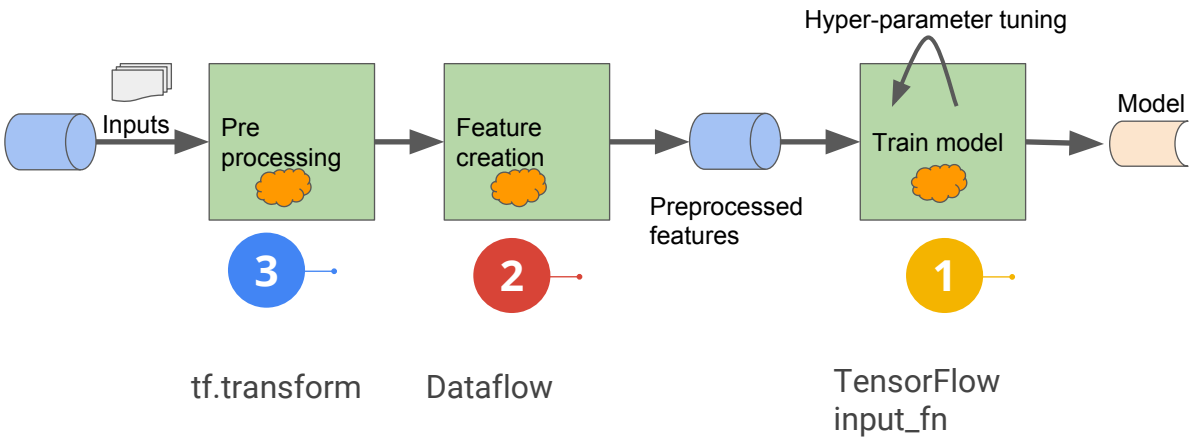
Wide-and-deep network in tf.learn

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

Notes:

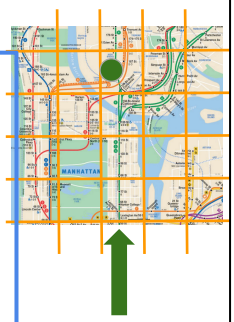
Or you can build it layer-by-layer with tf.layers() -- it's pretty straightforward.
https://pixabay.com/en/cake-piece-of-pie-black-forest-1227842/ (cc0)

Three possible places to do feature engineering



1 tf.layers supports some preprocessing


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



1 Supports a lot of common preprocessing steps

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
```

1 Can be quite powerful since it is so flexible

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'
```

2

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

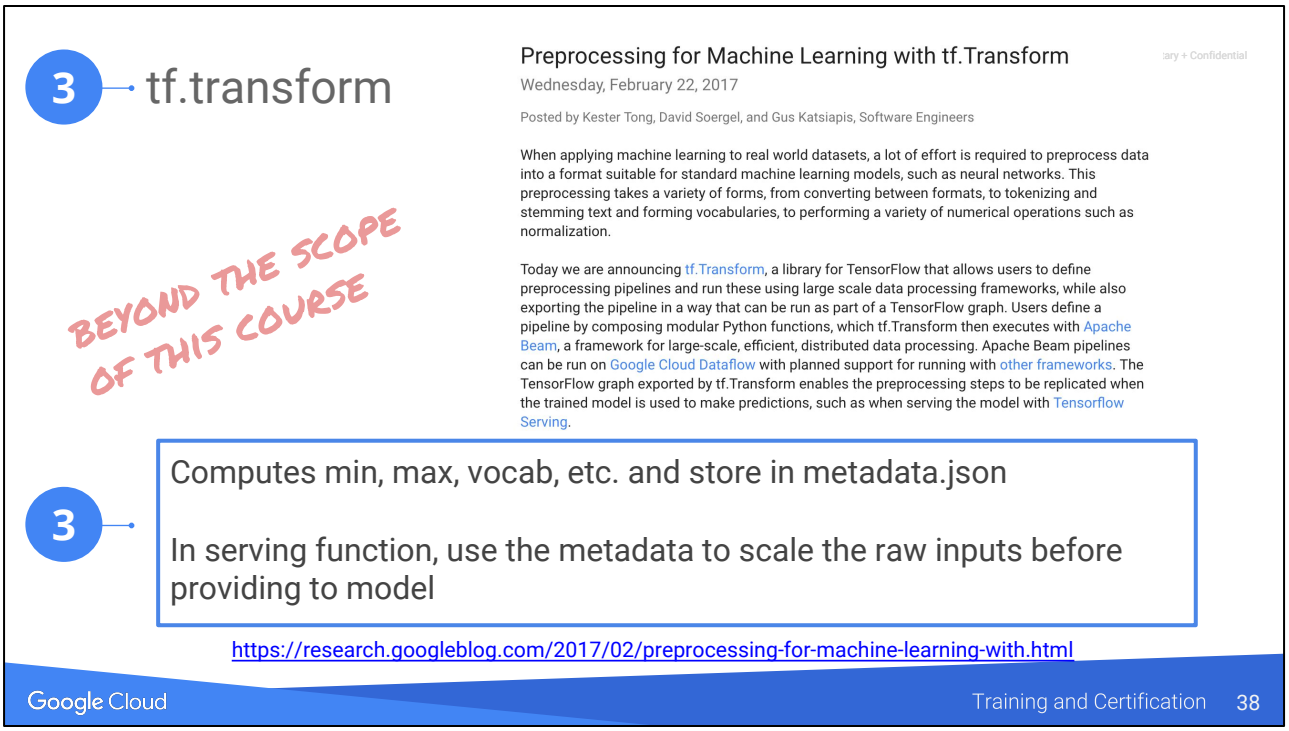
You will have to compute these features in real-time pipeline for predictions (i.e., will have to use Dataflow for predictions also)

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 ...

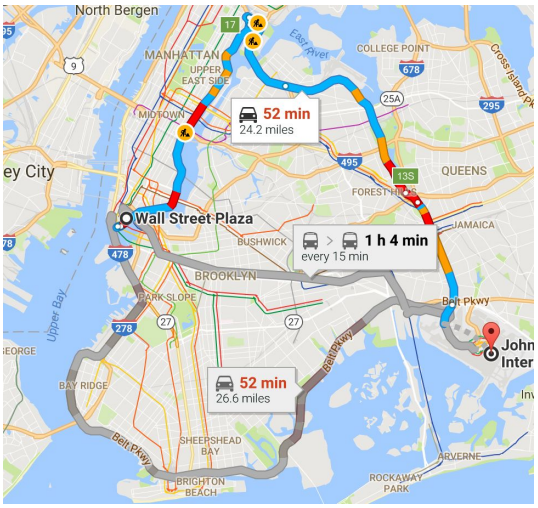


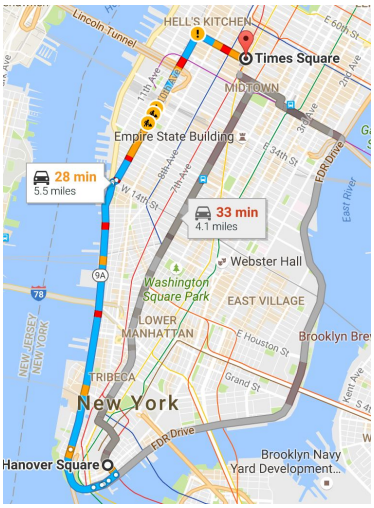
Notes:

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

Goal: To estimate taxi fare





Taxi fares:

$2.50 initial charge
+
50c per ⅕ mile
(or)
50c per minute if stopped
+
Passenger pays tolls
+
Various special charges

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.

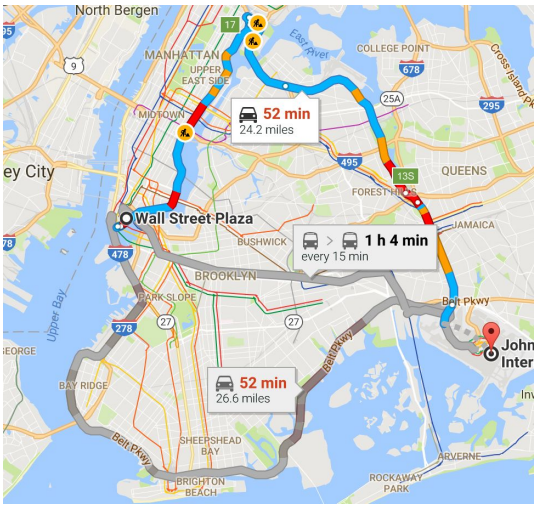
But assume we don't know any of these things. Instead of hardcoding a bunch of rules, let's try to infer the fare amount simply from the data.
https://pixabay.com/en/question-mark-question-faq-ask-1421013/ (cc0)

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.

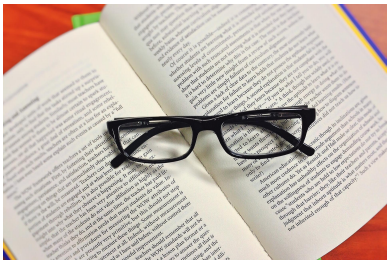
The other goes through Lincoln Tunnel and has different characteristics. You might think of predicting cost as a classification problem, so that you get probabilities for both routes. In this case, you’d have to bin the cost into categories (2.50-7.50, 7.50-12.50, etc.) and predict the likelihood of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

Lab: Serverless Machine Learning

Part 4. Feature engineering

In this notebook, you will learn how to incorporate feature engineering into your 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:

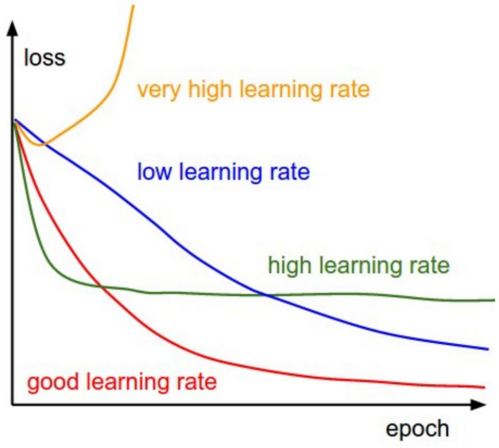
https://pixabay.com/en/book-glasses-read-education-1176256/ (cc0)

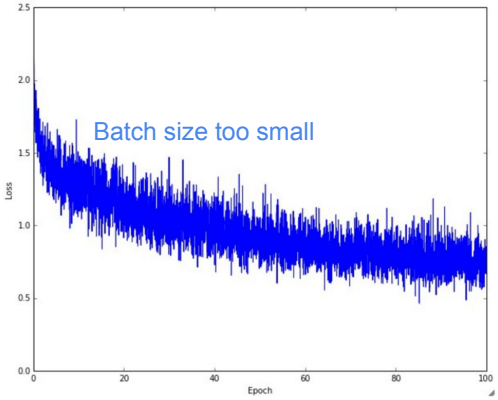
Agenda

Creating good features + Lab

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.

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. Add a special evaluation metric
3. Make sure outputs don't clobber each other
4. Supply hyperparameters to training job

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

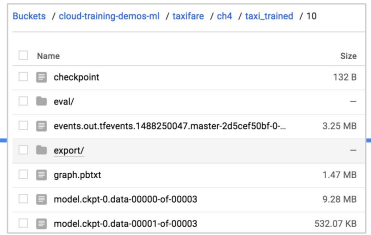
```
parser.add_argument(
    '--nbuckets',
    help='Number of buckets into which to discretize lats and lons',
    default=10,
    type=int
)
parser.add_argument(
    '--hidden_units',
    help='List of hidden layer sizes to use for DNN feature columns',
    default="128 32 4"
)
```

2. Add a special evaluation metric

```
def get_eval_metrics():
  return {
     'rmse':
        tflearn.MetricSpec(metric_fn=metrics.streaming_root_mean_squared_error),
     'training/hptuning/metric':
        tflearn.MetricSpec(metric_fn=metrics.streaming_root_mean_squared_error),
  }
```

3. 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', '')
 )
```



4. Supply hyperparameters to training

```
%writefile hyperparam.yaml
trainingInput:
  scaleTier: STANDARD_1
  hyperparameters:
    goal: MINIMIZE
    maxTrials: 30
    maxParallelTrials: 1
    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 \
   --num_epochs=100
```


Notes:

Minimize the RMSE on the evaluation dataset in this example.

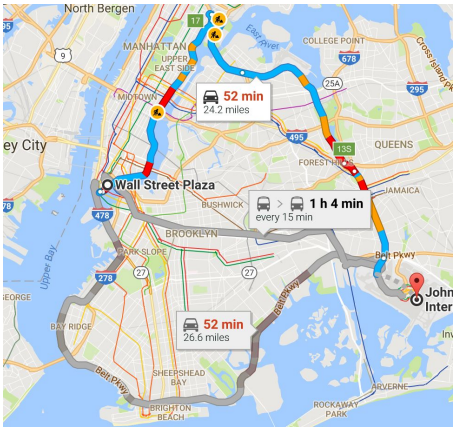
Systematically try various neural network architectures and different processing methods.

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

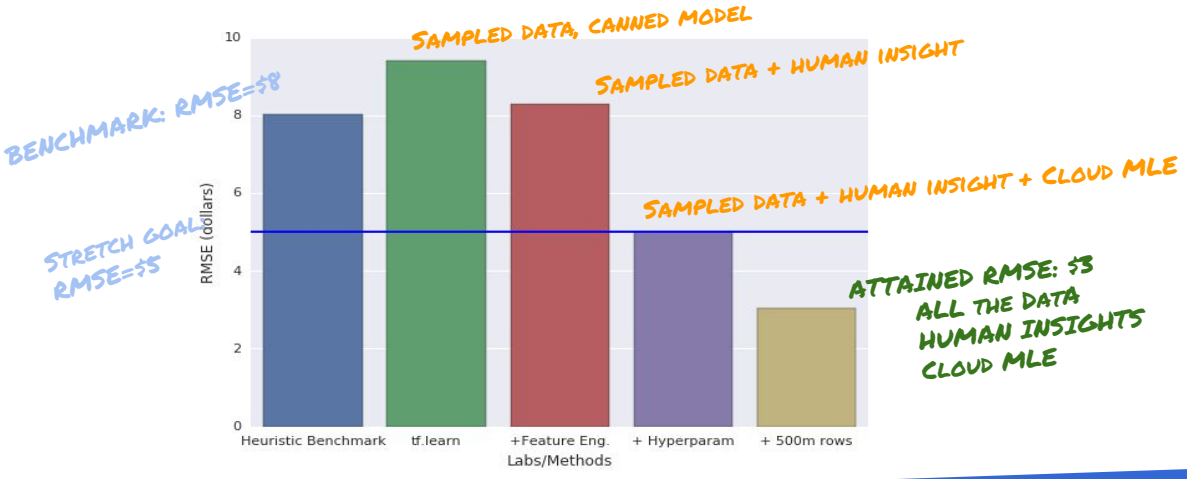


https://github.com/GoogleCloudPlatform/training-data-analyst/tree/master/courses/machine_learning/feateng/hyperparam.ipynb

Notes:

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

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



Resources

Feature Engineering for KDD Cup 2010	http://pslcdatashop.org/KDDCup/workshop/papers/kdd2010ntu.pdf
Stanford CS class on convolutional neural networks	http://cs231n.github.io/


cloud.google.com
Image by Connie Zhou