What is Machine Learning?

Google

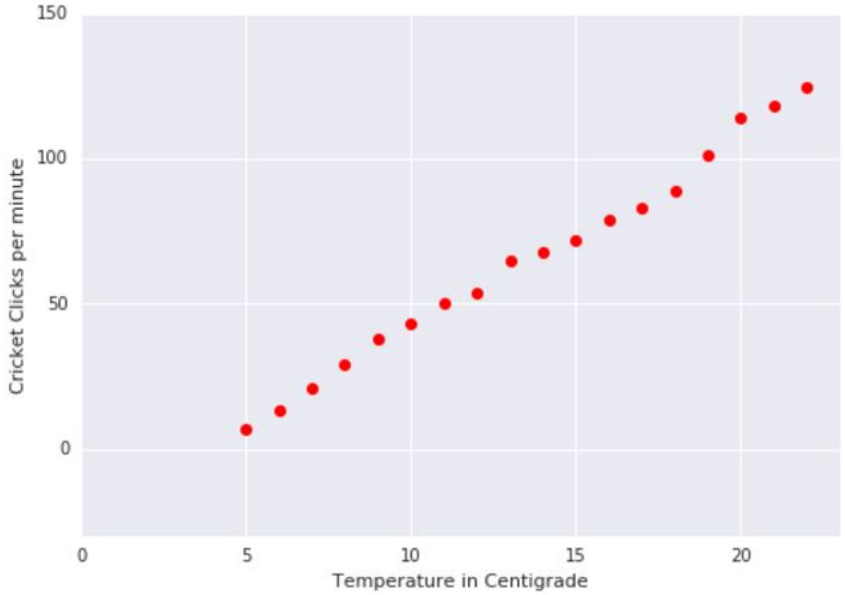
What is Machine Learning (ML)?

There are many ways to define ML.

- *ML systems learn how to combine data to produce useful predictions on never before seen data*
- ML algorithms find patterns in data and use these patterns to react correctly to brand new data.

Sample Machine Learning Problem

- Predict cricket chirps/min from the temperature in centigrade
- To the right is a plot showing cricket chirps per minute (y-axis) vs temperature (x-axis)

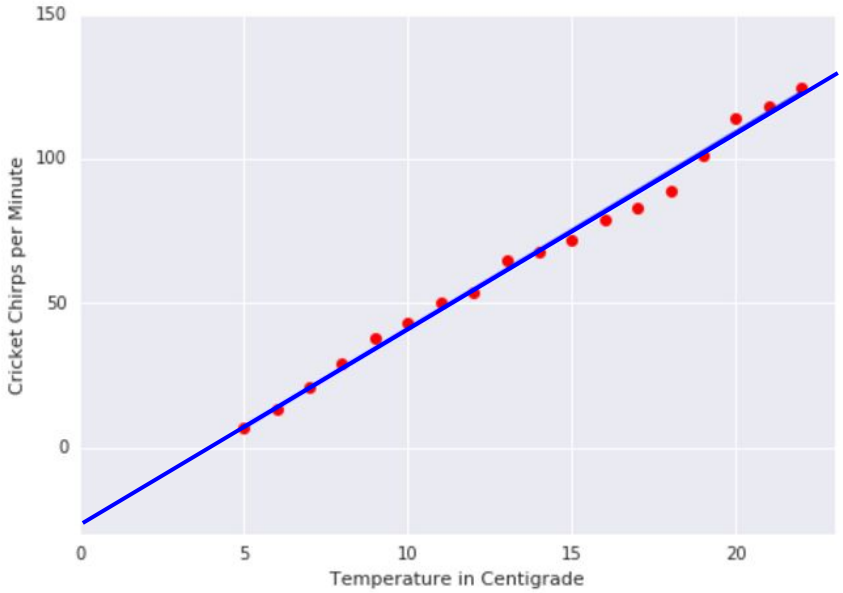


- What do you observe?

Google

Sample Machine Learning Problem (cont)

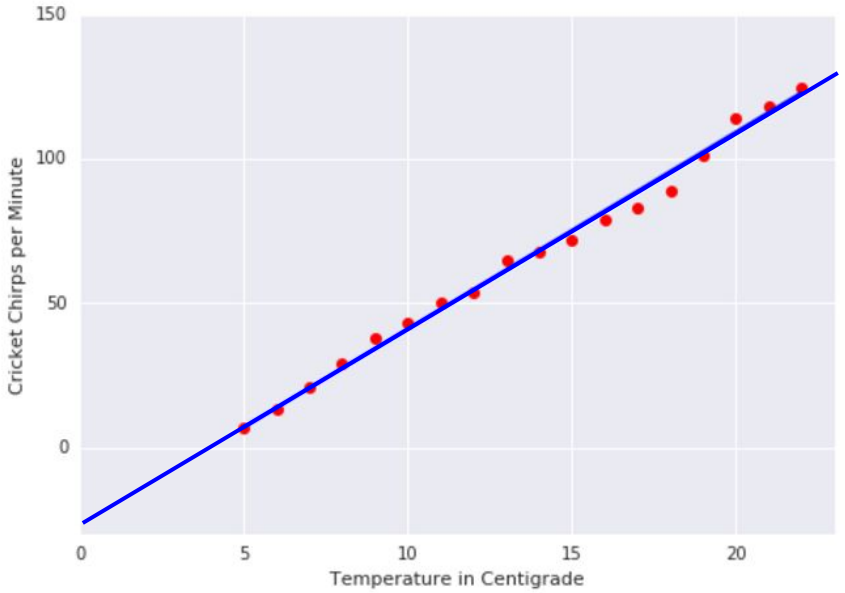
- The line shown in blue fits the data well.
- Recall that the equation for a line is $y = mx + b$ with slope m and y-intercept b.



Google

Sample Machine Learning Problem (cont)

- The line shown in blue fits the data well.
- Recall that the equation for a line is $y = mx + b$ with slope m and y-intercept b.
- In this example: $y = 7x - 30$

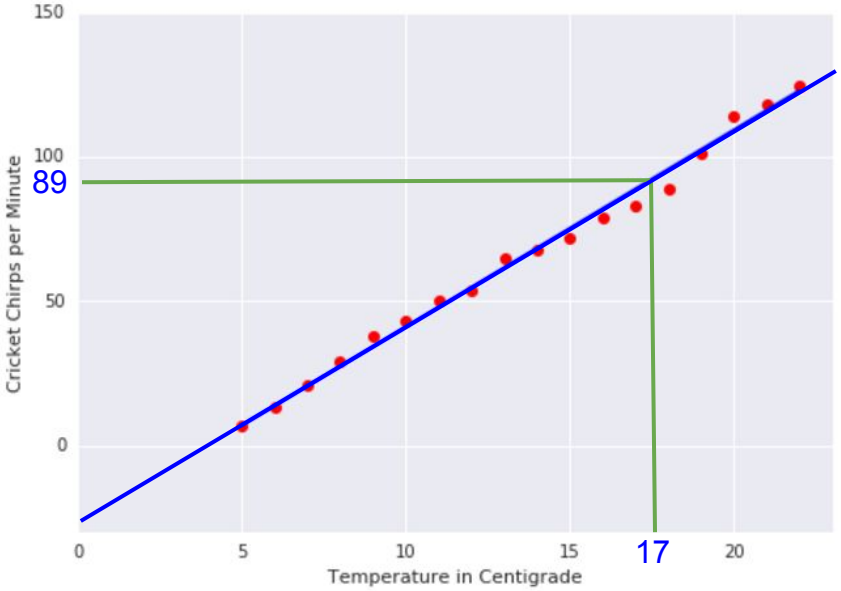


Google

Sample Machine Learning Problem (cont)

- The line shown in blue fits the data well.
- Recall that the equation for a line is $y = mx + b$ with slope m and y-intercept b.
- In this example: $y = 7x - 30$

- If we pick a temp not in the data, we can use this line to predict the chirps. If x = 17, then y = 7 * 17 - 30 = 89

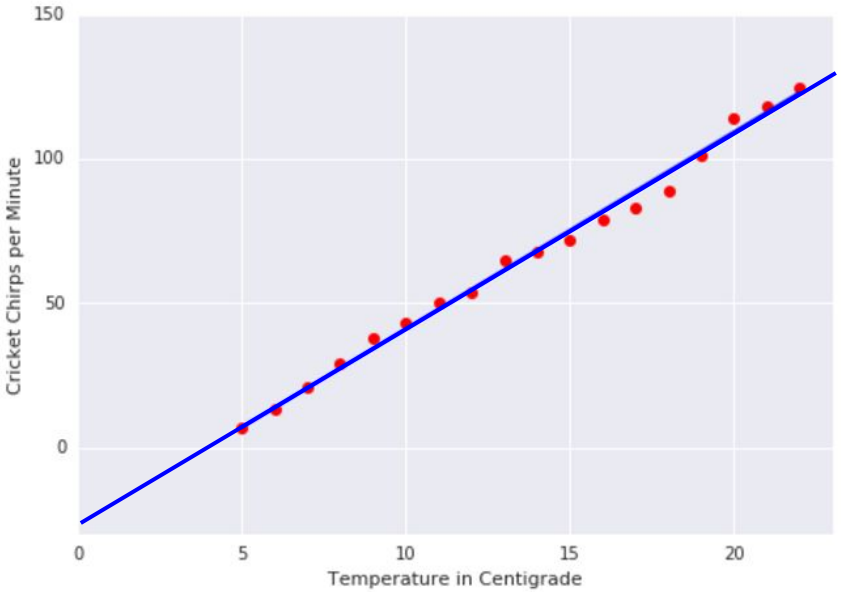


Google

Sample Machine Learning Problem (cont)

- The line shown in blue fits the data well.
- Recall that the equation for a line is $y = mx + b$ with slope m and y-intercept b.

- In machine learning we often call this a model since it is what we use to make our predictions.
- We call this model a linear model since we fit the data with a line



Google

Definition: Linear Regression

- When a learning model fits data by a line, we say it is linear
- Regression is relationship between correlated variables
  - When you have a +/- change in a variable that affects the other variable +/-
- When you have just one input value (feature), linear regression is the problem if fitting a line to the data and using this line to make new predictions.
- By convention in machine learning we write: $y' = b + wx$. So instead of *m, we* use *w*.

Linear Regression with Many Features

- Often you have multiple features that can be used to predict a value.
  - For example suppose you want to predict the rent for an apartment.
  - Two relevant features are: the number of bedrooms and the apt condition from 1 (poor) to 5 (excellent)
- In this case the input ***x*** becomes a pair (x_1, x_2) where x_1 is the number of bedrooms and x_2 is the apt condition.
- By convention we make ***x*** bold since it is a vector.
- Thus our equation to predict the rent becomes:

$$y' = b + w_1x_1 + w_2x_2$$

Linear Regression Notation

When you have *n* features the linear model becomes:

$$y' = b + w_1x_1 + w_2x_2 + \ldots + w_nx_n$$

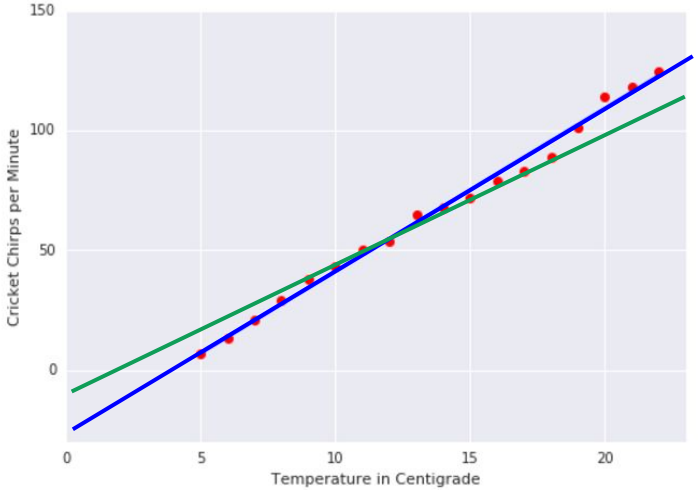
- y' is what we use as our predicted value for the label
- x_k is a feature (one of the input values)
- w_k is the weight (importance) associated with feature *k*
- b is the bias (y-intercept)

Sometimes w_0 is used instead of b with a default that $x_0 = 1$

The model parameters (which are to be learned) are: $b, w_1, w_2, \ldots, w_n$

Loss

- We need a way to say what solution fits the data best
- **Loss** is a penalty for an incorrect prediction
- Lower loss is generally better
- Choice of loss function guides the model we choose



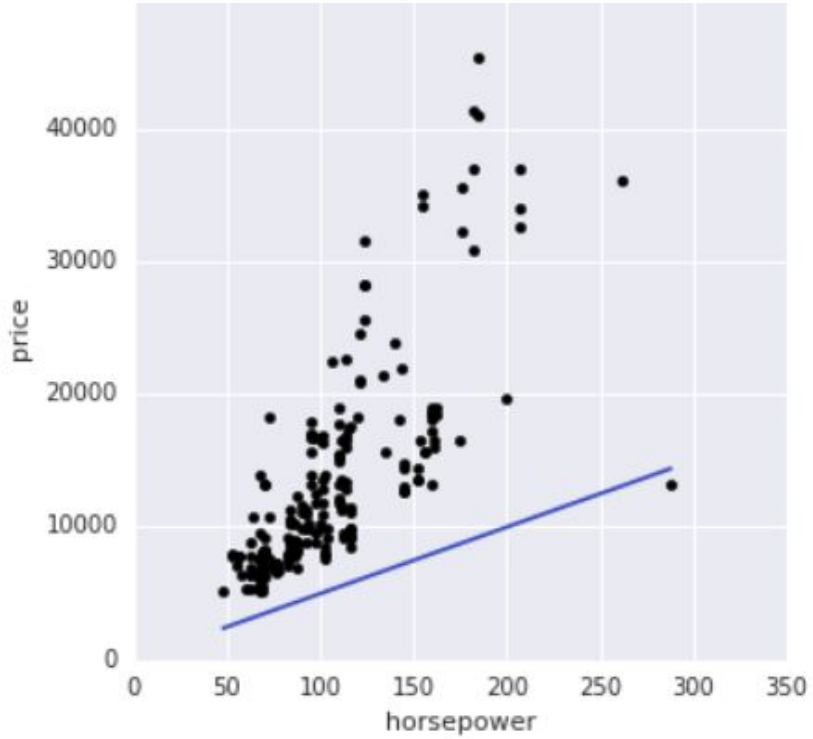
Do you think the green line or blue line will lead to better predictions?

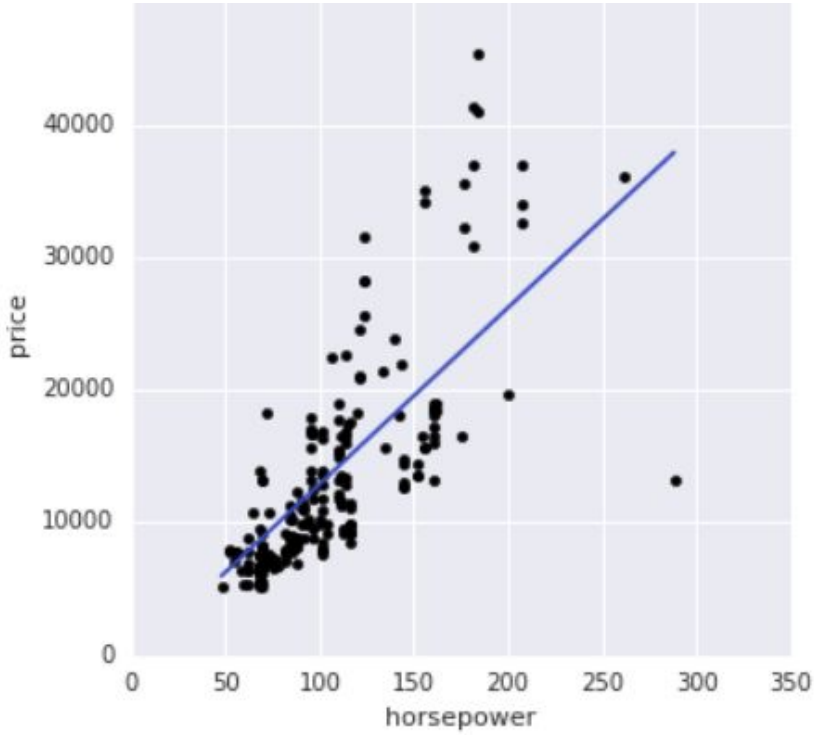
Google

Evaluating If a Linear Model Is Good

- We will eventually look at mathematical ways to evaluate the quality of a model but that should not replace more informal yet intuitive ways to evaluate a model
- When you just have one variable, drawing a line on a scatter plot that shows the model predictions is a good tool.
- Let’s look at two different models to predict the price of a car from its engine’s horsepower.

Which Model is Better?





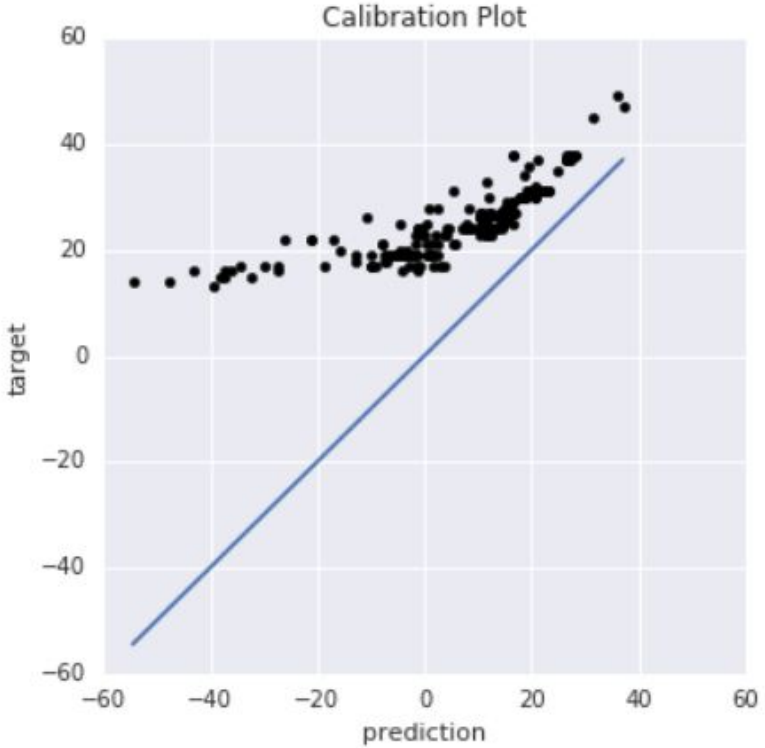
Google

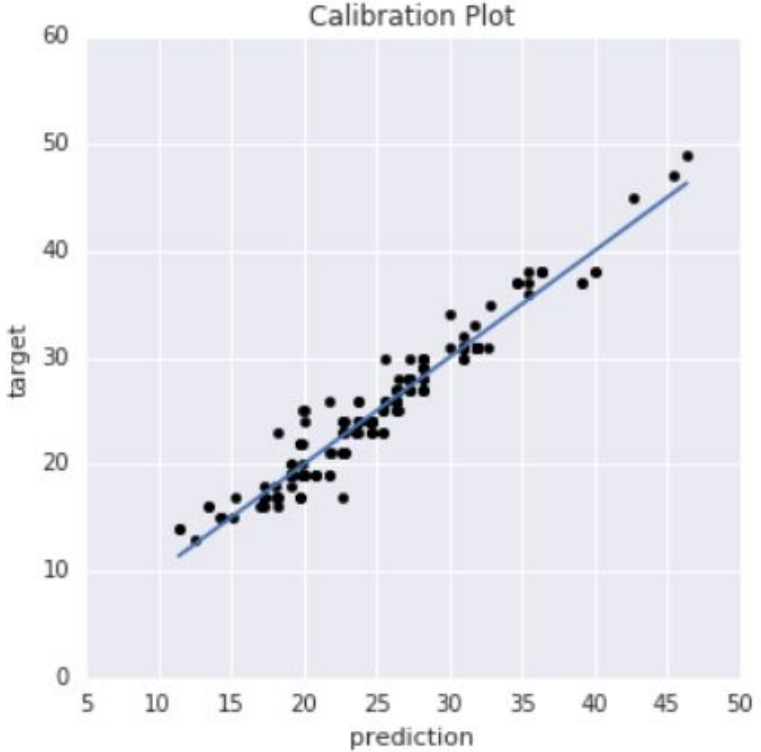
Calibration Plots To Visualize a Model

- We want something that can serve a similar role to a scatter plot when we have more than one input feature.
- For this we can use a calibration plot that shows the prediction (x-axis) versus the target value (y-axis)

- On the next slide we'll look at two calibrations plots from different models trained on the same data set.

Which Model is Better?

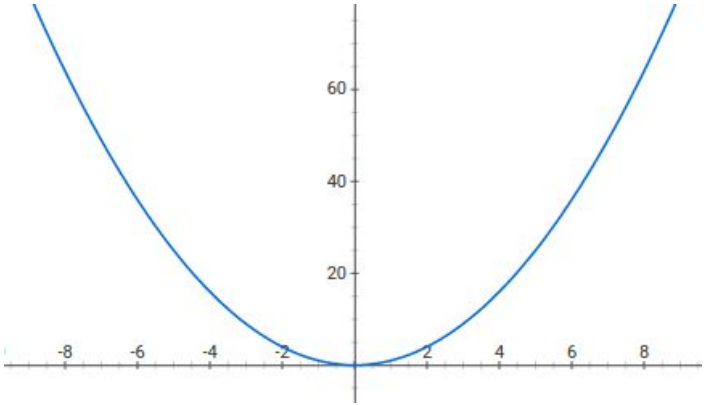




Google

A Convenient Loss Function for Regression

- ***L_2* Loss** for a given example is also called squared error
  = square of the difference between prediction and label
  = (observation - prediction)^2
  = $(y - y')^2$



Google

Computing Squared Error on a Data Set

For training set *D* composed of examples $\boldsymbol{x} = \langle x_0, x_1, x_2, \ldots, x_n \rangle$ and correct label *y*, and the prediction for the current model $\boldsymbol{w}$ of $y' = \boldsymbol{w}^T\boldsymbol{x} = w_0x_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n$, the squared error ($L_2$Loss) is:

$$L_2\text{Loss} = \frac{1}{|D|} \sum_{(\mathbf{x},y)\in D} (y - y')^2$$

Generally average over all examples, so divide by the number of examples in *D* (denoted by |*D*|)

We're summing over all examples in our training set *D*

Google

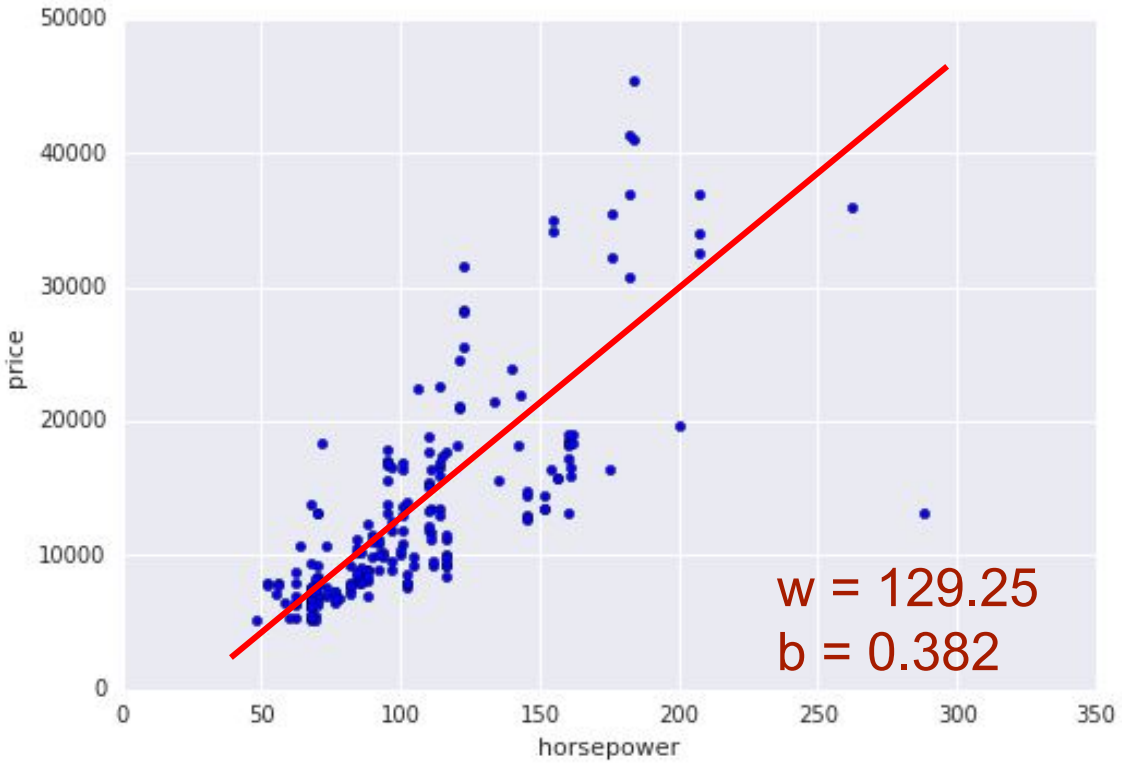
RMSE - Root Mean Squared Error

For training set *D* composed of examples ***x*** = <x_0, x_1, x_2, …, x_n> and correct label *y*, and the prediction for the current model *y'*

$$\text{RMSE} = \sqrt{\frac{1}{|D|} \sum_{(\mathbf{x},y)\in D} (y - y')^2}$$

Our goal is to train a model that minimizes RMSE (which is the same as minimizing the mean squared error).

Summary of Linear Regression



- $y' = wx + b$
- Minimize mean squared error (or RMSE)
- In general have:

$y' = b + w_1x_1 + \ldots + w_nx_n$

- Learn model weights b, w_1, w_2, … w_n from data

Google

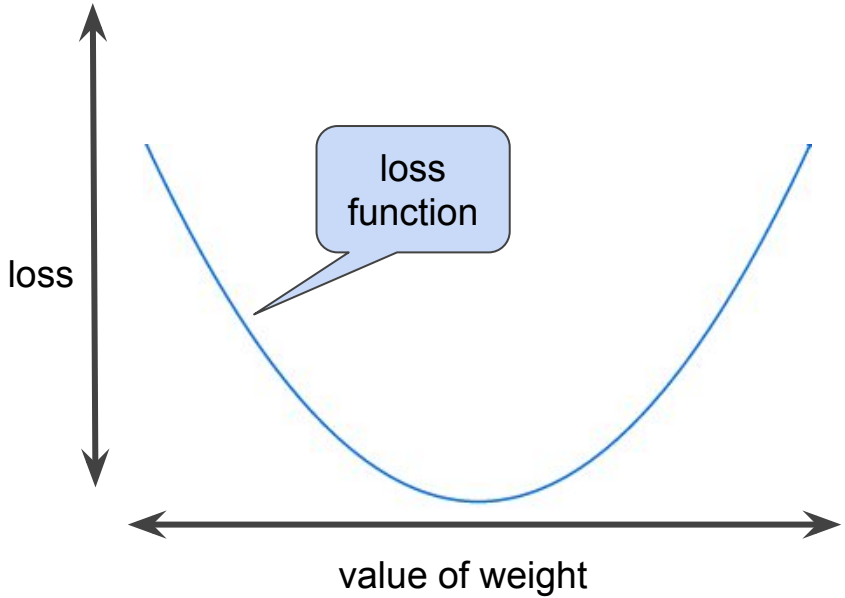
Learning Model Weights

Gradient Descent: High Level View

- Derivative of $(y - y')^2$ w.r.t. the model weights ***w*** tells us how loss changes in the neighborhood of weights we're currently using.
- A gradient is the generalization of a derivative when you have more than one variable.
- We can take small steps in the negative gradient direction to modify the weights so that the loss on that example is lower.
- This optimization strategy is called Gradient Descent.

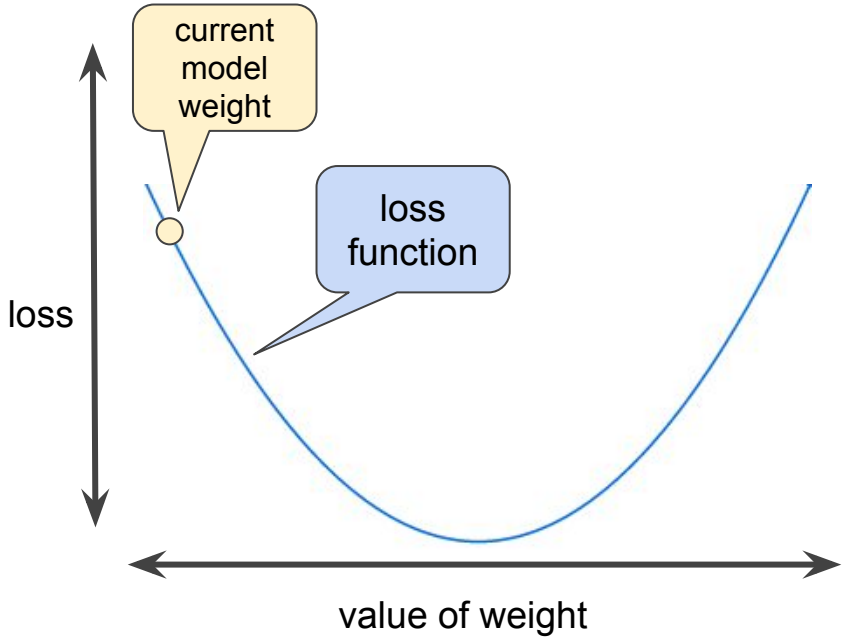
Pictorial View of Gradient Descent

- As a simple illustration assume our model has a single weight and we are using squared loss
- To the right we plot the squared loss in blue



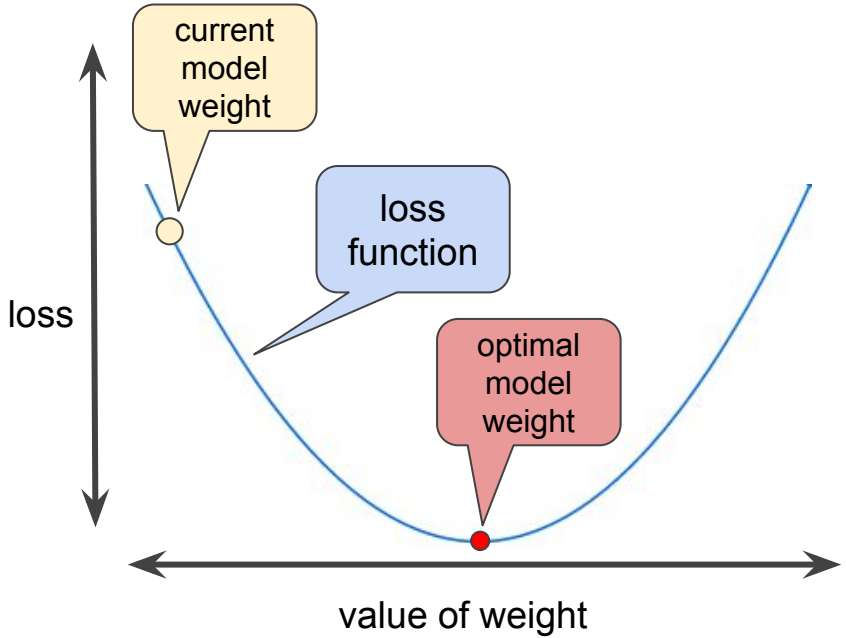
Pictorial View of Gradient Descent

- As a simple illustration assume our model has a single weight and we are using squared loss
- To the right we plot the squared loss in blue
- The yellow dot represents our current model weight

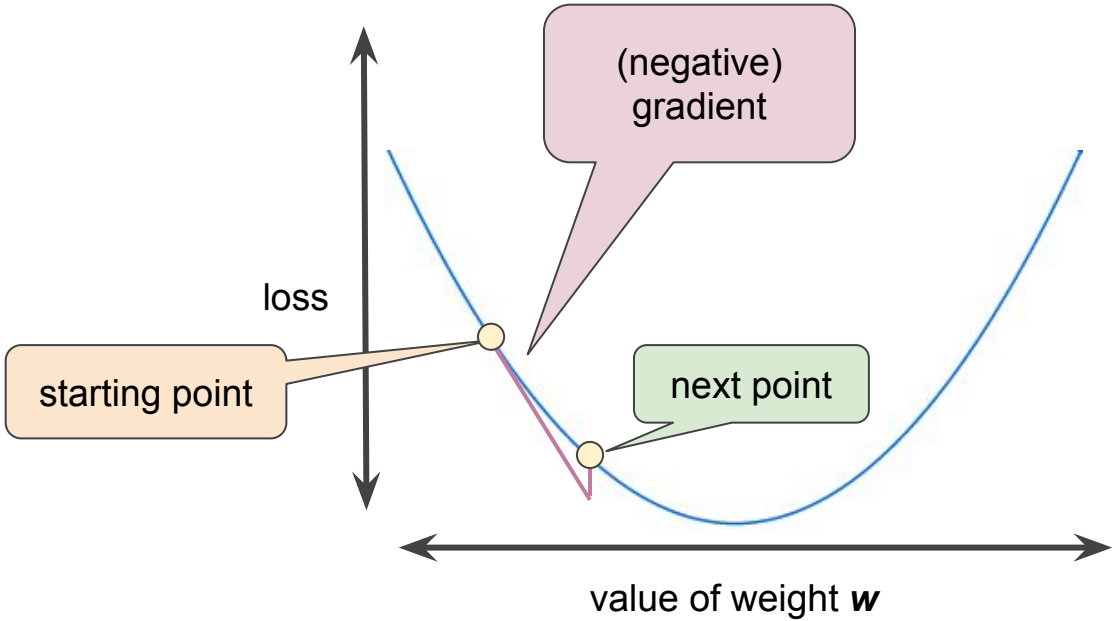


Pictorial View of Gradient Descent

- As a simple illustration assume our model has a single weight and we are using squared loss
- To the right we plot the squared loss in blue
- The yellow dot represents our current model weight
- The red dot is the model weight minimizing squared loss

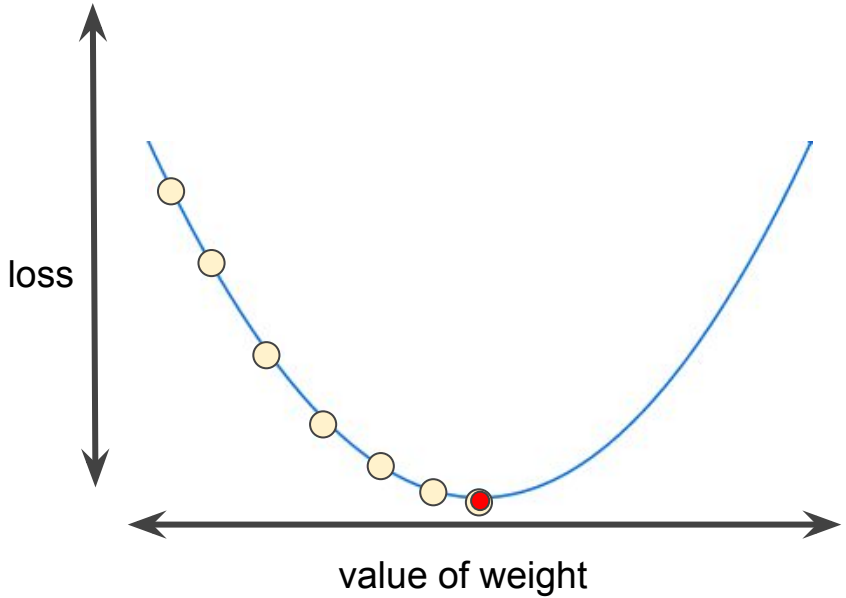


Training: A Gradient Step



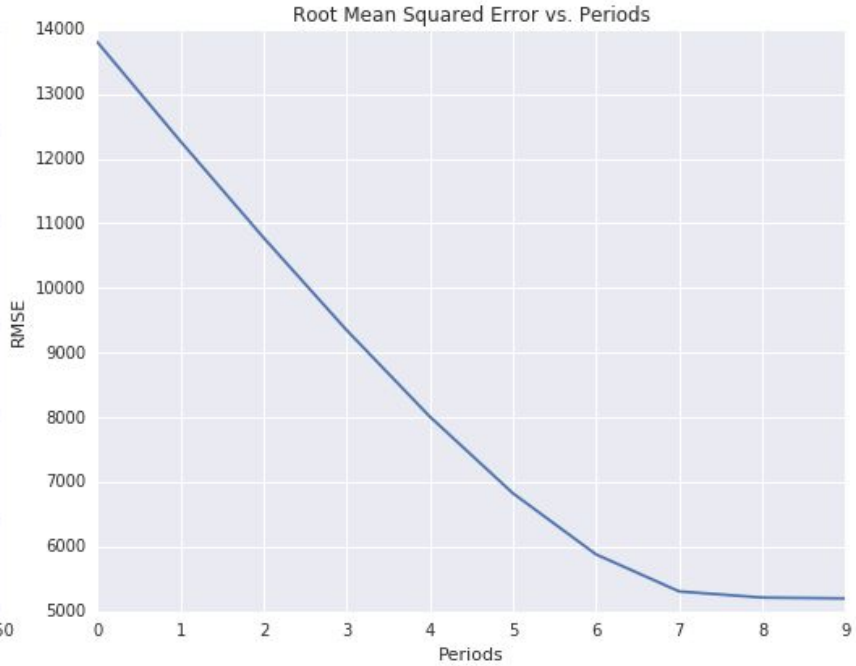
Learning Rate: Size of Step to Make

- The sequence of yellow dots shows how the model evolves with the weight updates
- Gradient descent with a good learning rate gives small but noticeable steps towards the optimal value
- We want to end with a model very close to the optimal model as seen here



Training a Model with Gradient Descent

In the scatter plot we see the model evolve (from the blue to red line)



The Learning Curve shows how the loss changes over time

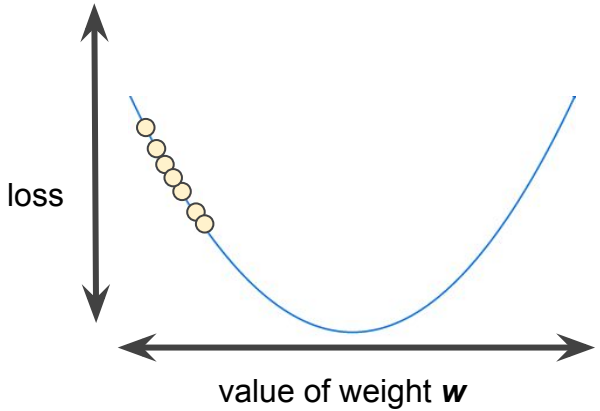
Google

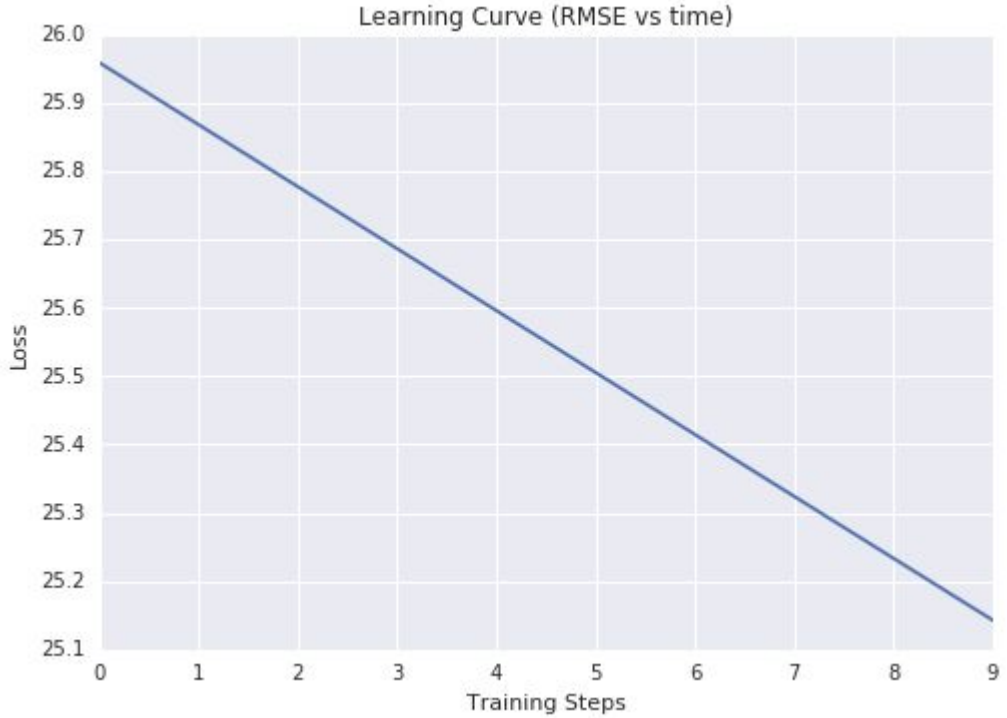
Things You Need to Decide

- Learning Rate
  - Very important since this is the size of the step to take. Typically change by powers of 10 until the model is training reasonably well and then fine tune
- Number of Steps to Train
  - Time to train is proportional to this (for fixed set of features). You want to make this is as small as you can while still getting to the minimum loss possible.
- What Features to Use
  - This is very important and will be our next main topic

Learning Rate Way Too Low

- Moving towards optimal model but too slowly

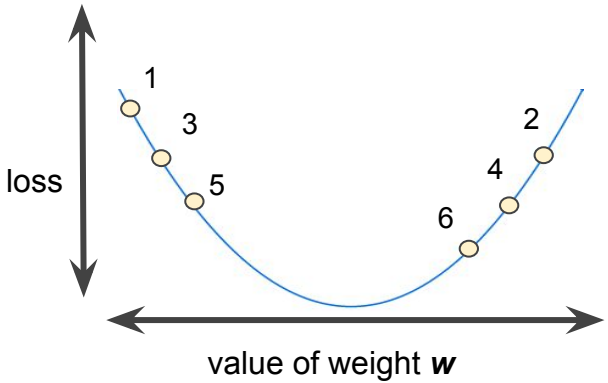




Google

Learning Rate Too High

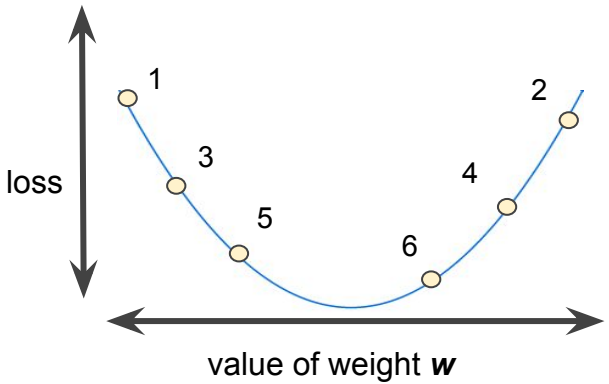
Though loss is decreasing, going back and forth overshooting optimal weights

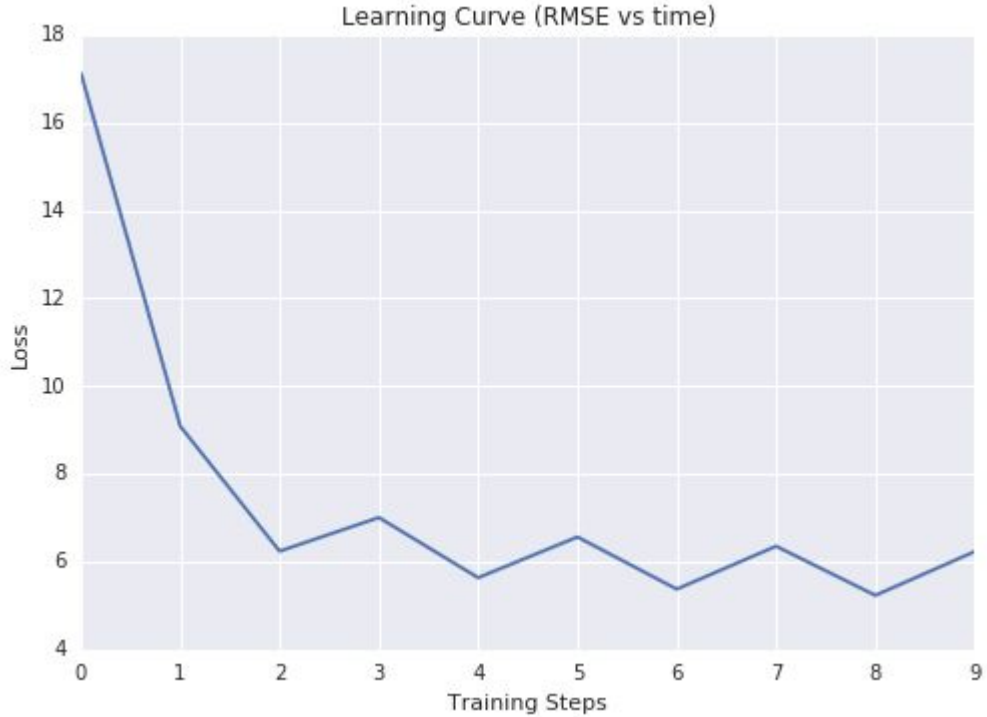


Google

Learning Rate Still Too High

Still overshooting as seen in loss oscillating



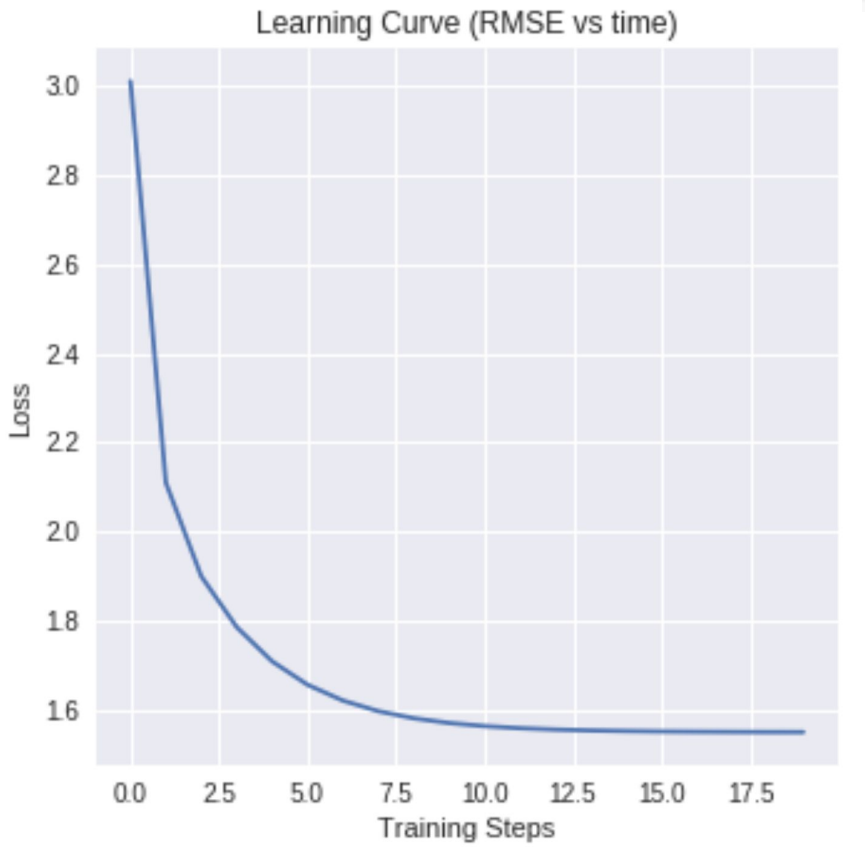


Google

Need More Steps (loss still going down)

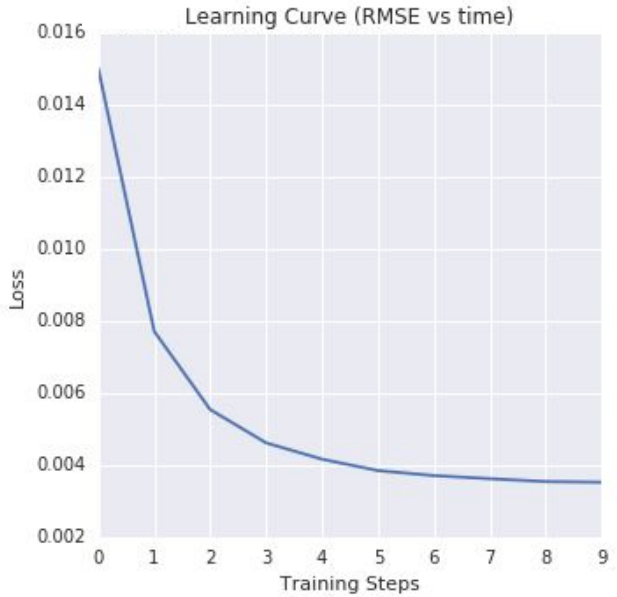
Google

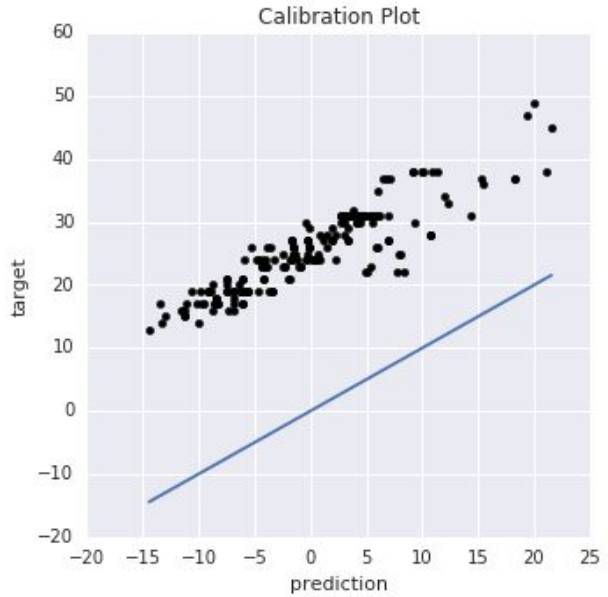
Good Learning Rate and Number of Steps



Google

Converging to a Poor Model





Though the model appears to have converged, we can see from the calibration plot it is a poor model. Try a different learning rate or better features. Very important to evaluate if model is any good!

Google

Transforming Features

Google

Start By Exploring Your Data

- **Gather Statistics**
  - Average values
  - Min and max values
  - Standard deviation
- **Visualize**
  - Plot histograms
  - Are there missing values?
  - Are there outliers?

	weight	horsepower	peak-rpm	city-mpg	highway-mpg	price
count	205.00	203.00	203.00	205.00	205.00	201.00
mean	2555.57	104.26	5125.37	25.22	30.75	13207.13
std	520.68	39.71	479.33	6.54	6.89	7947.07
min	1488.00	48.00	4150.00	13.00	16.00	5118.00
25%	2145.00	70.00	4800.00	19.00	25.00	7775.00
50%	2414.00	95.00	5200.00	24.00	30.00	10295.00
75%	2935.00	116.00	5500.00	30.00	34.00	16500.00
max	4066.00	288.00	6600.00	49.00	54.00	45400.00

Google

Feature Engineering

Process of creating features from raw data is Feature Engineering

This process of using domain knowledge and a high-level understanding of the ML model you are using to create good features

Feature Engineering is important and often a very big part of whether a ML system working well in practice.

Why Transform Features

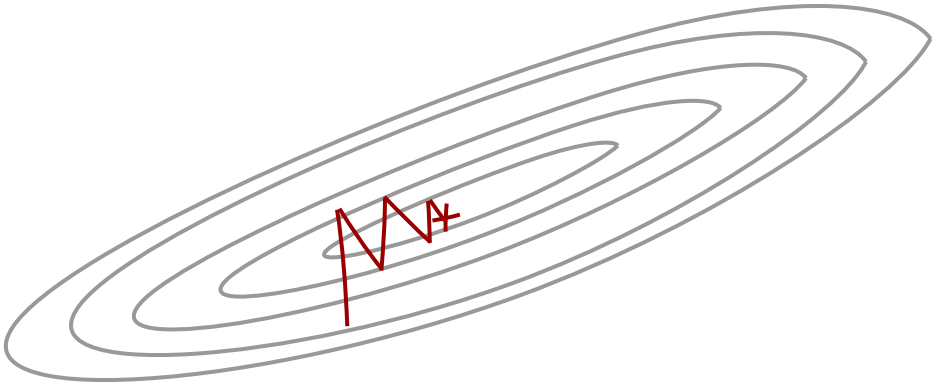
- Linear models can only use a feature values via taking linear combinations of them ($y' = b + w_1x_1 + \ldots + w_nx_n$)
- Can learn better with normalized numeric features so all are equally important initially
- We must handle missing data, outliers,....
- Convert non-numeric features into numeric values
  - can't multiply a weight times a string

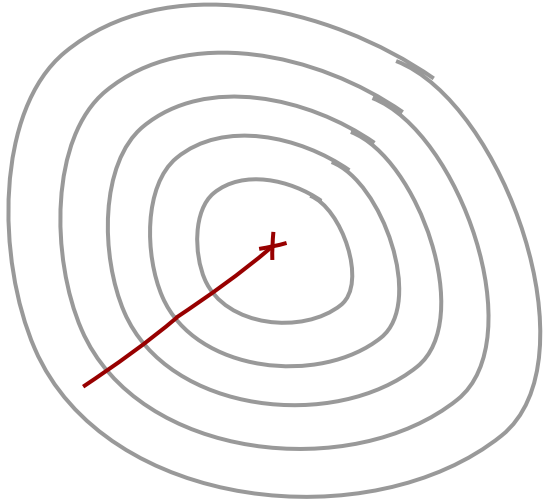
Why Transform Features (cont)

- Linear models can represent a lot more problems when we add **non-linearities** to the features
- We can often introduce new features using domain knowledge. For example, given the width and length of a room, we might add the feature: area = width * length.
- Tokenize or lower-casing of text features
- ...

Transforming Numeric Features

Models tend to converge faster if features on similar scale





Transforming Numeric Features (cont)

- As an example:
  - What if you are creating a linear model to predict city mpg of a car from highway mpg (x_1) and price (x_2)
  - What happens if you directly use these features?

Transforming Numeric Features (cont)

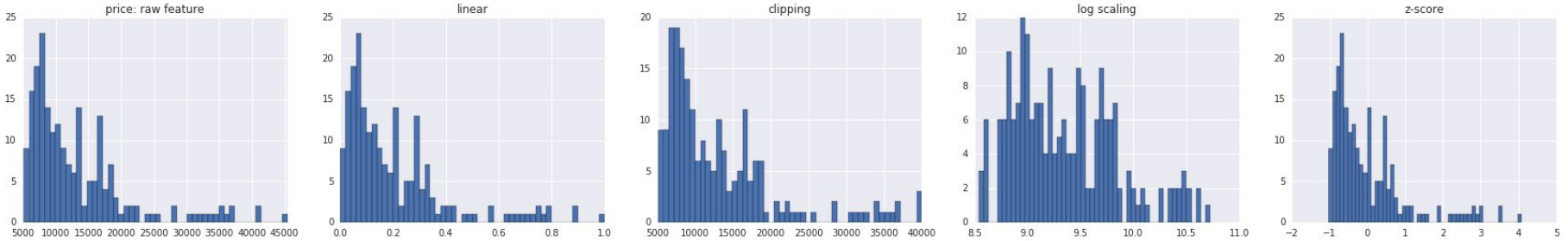
- As an example:
  - What if you are creating a linear model to predict city mpg of a car from highway mpg (x_1) and price (x_2)
  - What happens if you directly use these features?
  - The price is a much larger number and so dominates in the linear function $w_1x_1 + w_2x_2$ thus making it hard for the model to learn that highway mpg is much more important.

Transforming Numeric Features

Common scaling methods:

- **linear:** $x' = (x - x_{min}) / (x_{max} - x_{min})$ to bring to [0,1]
- **clipping:** set min/max values to avoid outliers
- **log scaling:** $x = \log(x)$ to handle very skewed distributions
- **z-score:** $x' = (x - \mu) / \sigma$ to center to 0 mean and stddev $\sigma = 1$

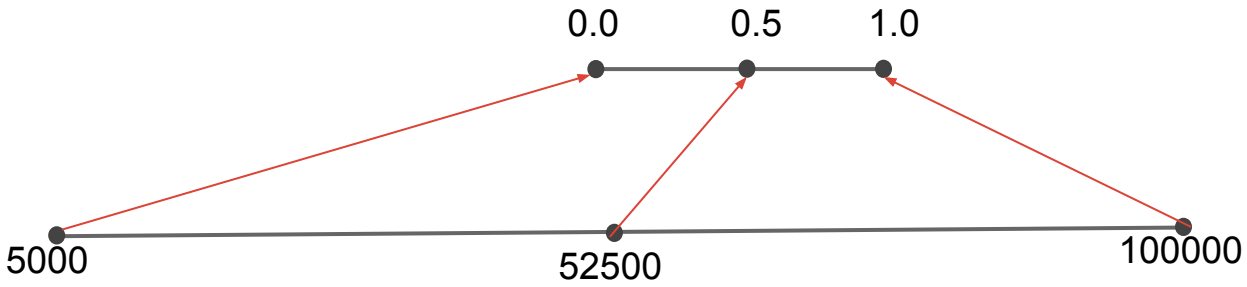
As always, experiment to see which works best in practice



Google

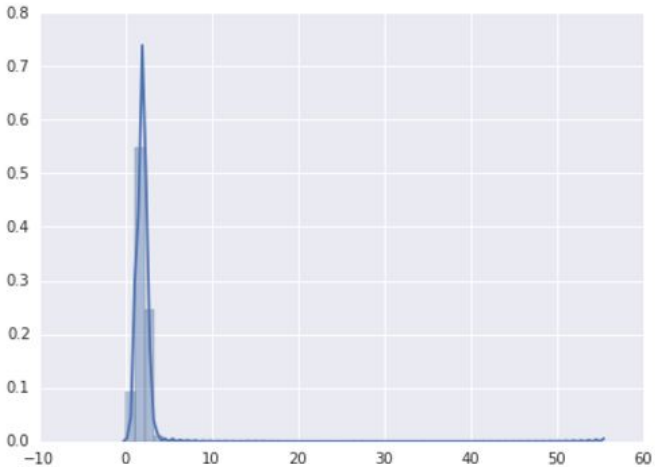
Linear Scaling

- Linearly scaling just stretches/shrinks the feature values to fall between 0 and 1 (or sometimes -1 to +1 is used).
- Transformed value $x' = (x - a) / (b - a)$
- For example, a feature that ranges from a=5000 to b=100000 get scaled between 0 to 1 as:

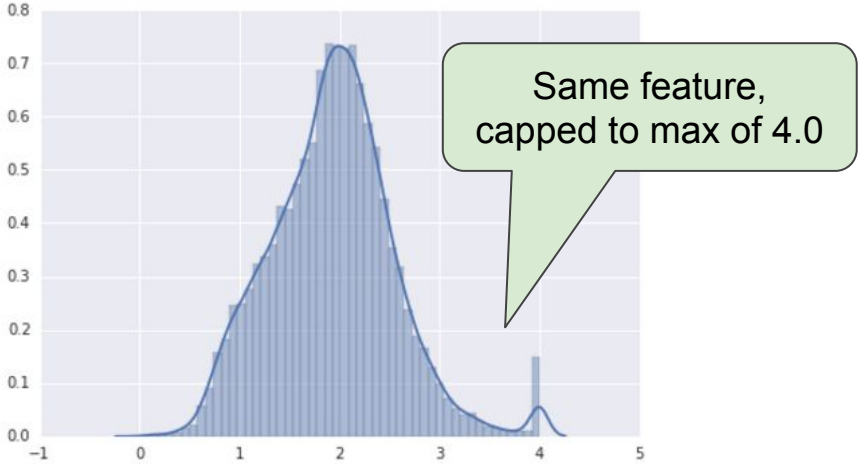


Feature Clipping

- It helps to get rid of extreme values (outliers) by capping all features above (or below) some value to a fixed value. This can be done before or after other normalizations.



Rooms Per Person

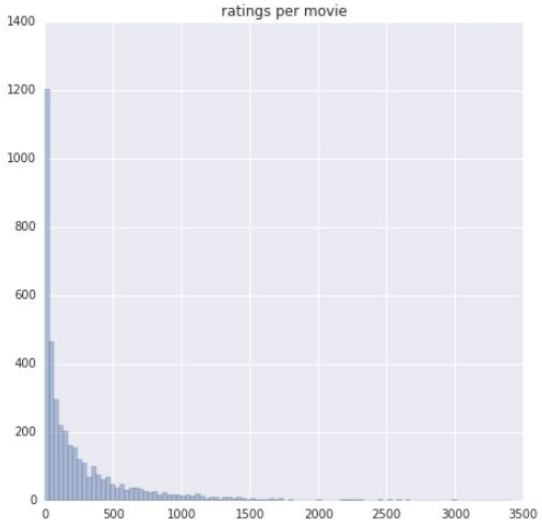


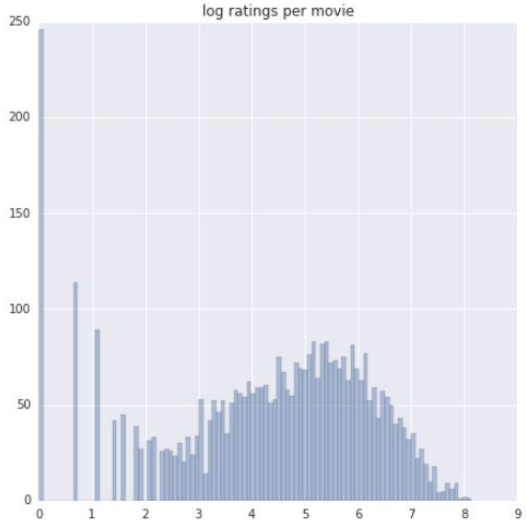
Capped Rooms Per Person

Google

Log Scaling

- Common data distribution: power law shown on the left
  - Most movies have very few ratings (often called the tail)
  - A few have lots of ratings (often called the head)
- Log scaling improves linear model performance

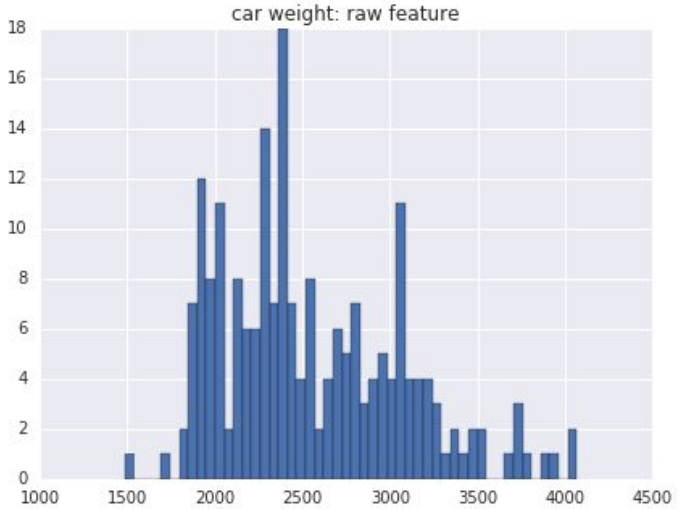


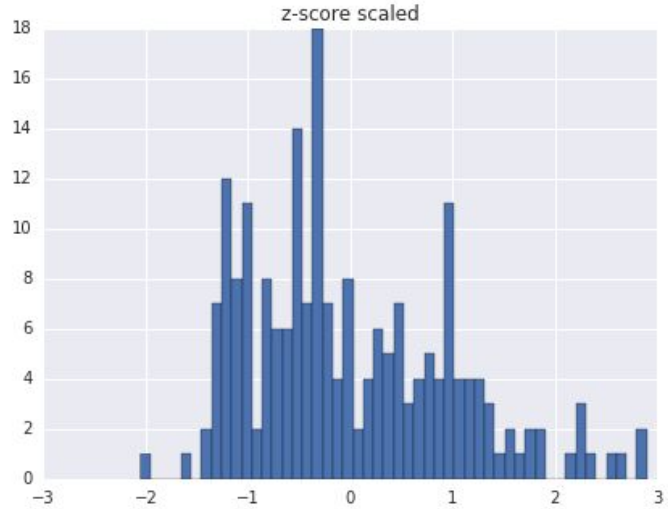


Google

Z-Score Scaling

- This is a variation of linear scaling that normalized that data to have mean 0 and standard deviation of 1. It's useful when there are some outliers but not so extreme you need clipping





Google

Add Non-Linearities To a Linear Model

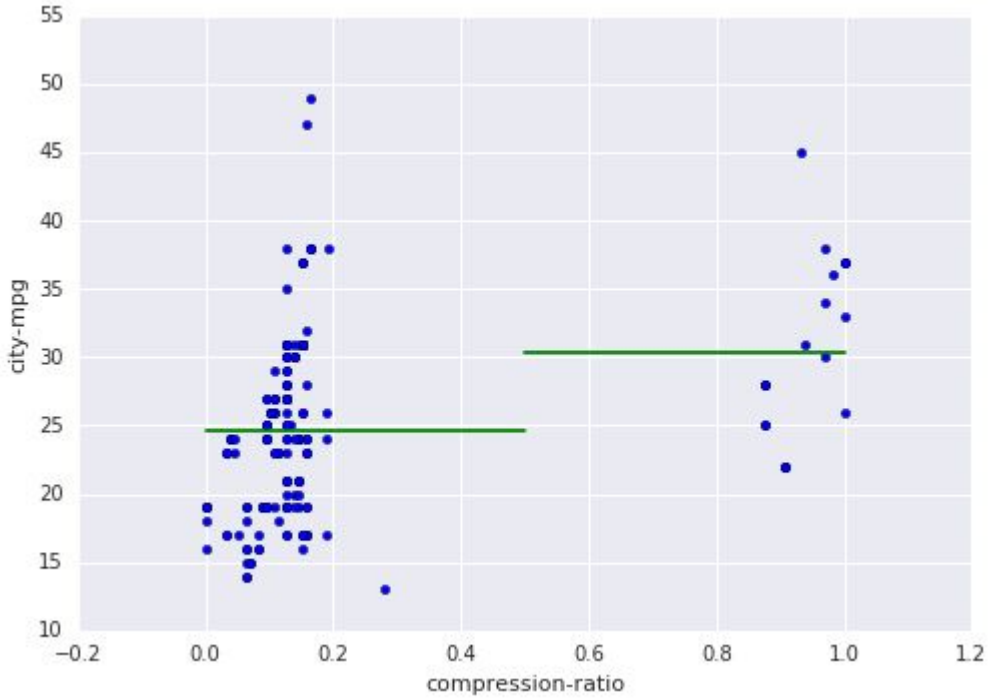
- Linear models can represent a lot more problems when we add **non-linearities** to the features
- One way to do this is to Bucketize numerical features

Motivation: Bucketizing Numeric Features

- Can try to use a linear model, but can't capture feature behavior well.
- RMSE 9.065
- Problem is there is a different behavior for the two ranges of compression ratio

Google

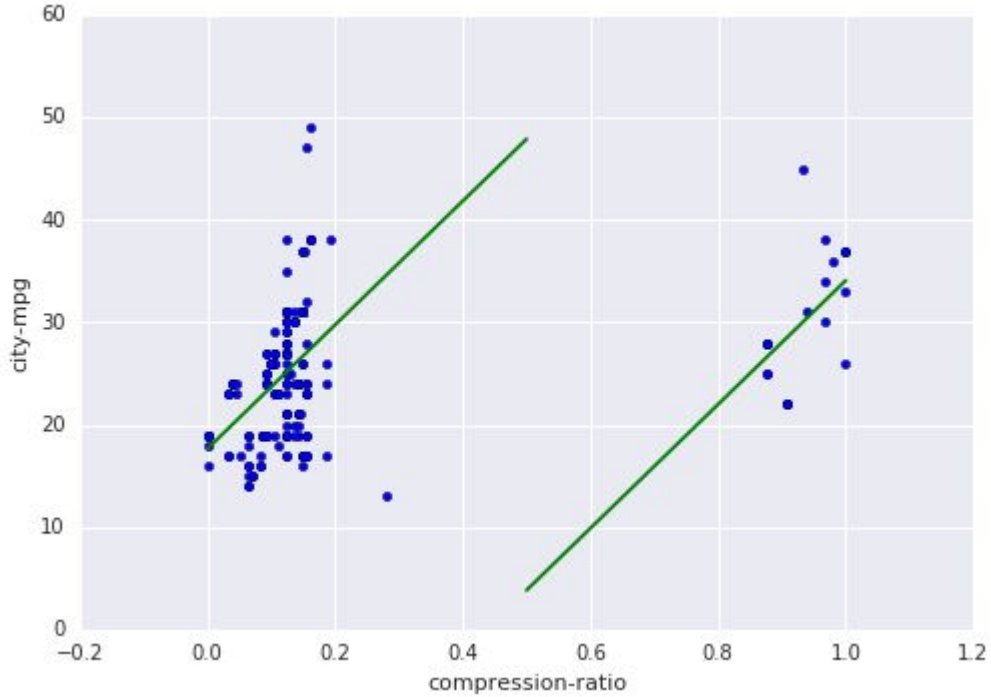
Bucketizing Compression Ratio



- Each binary bucketized feature has its own weight.
- RMSE 6.3
- Without also using the raw feature, we only get a single value (the bias) per bucket.

Google

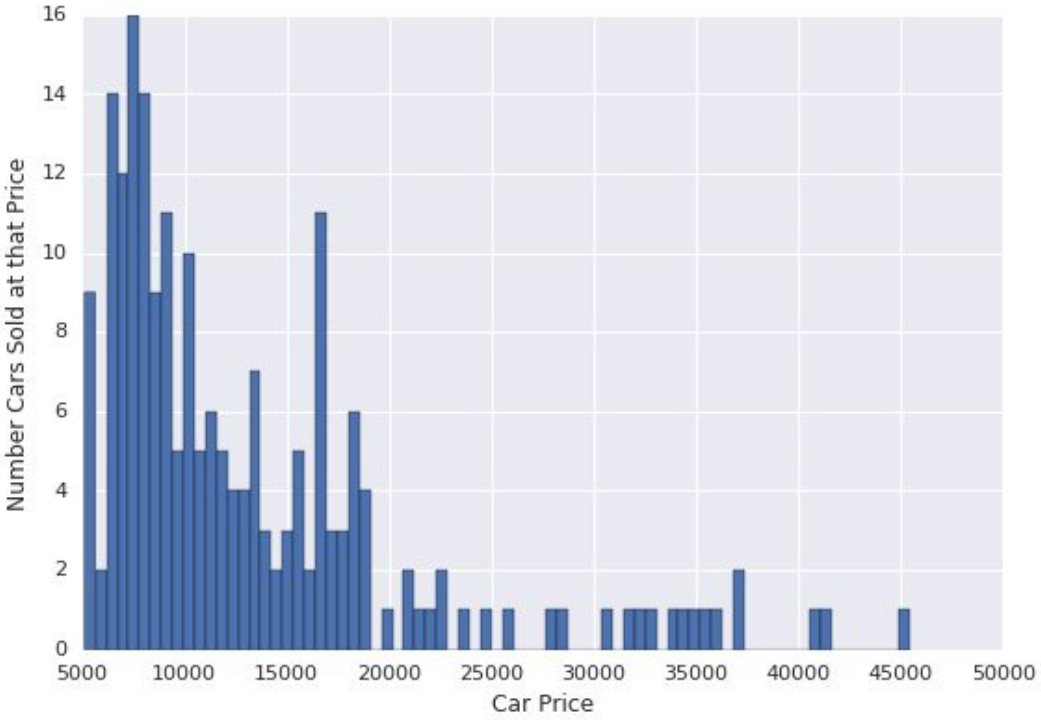
Using Raw and Bucketized Features



- Use raw feature and bucketized feature
- Linear + step function
- Shared slope, independent biases
- RMSE 5.7

Google

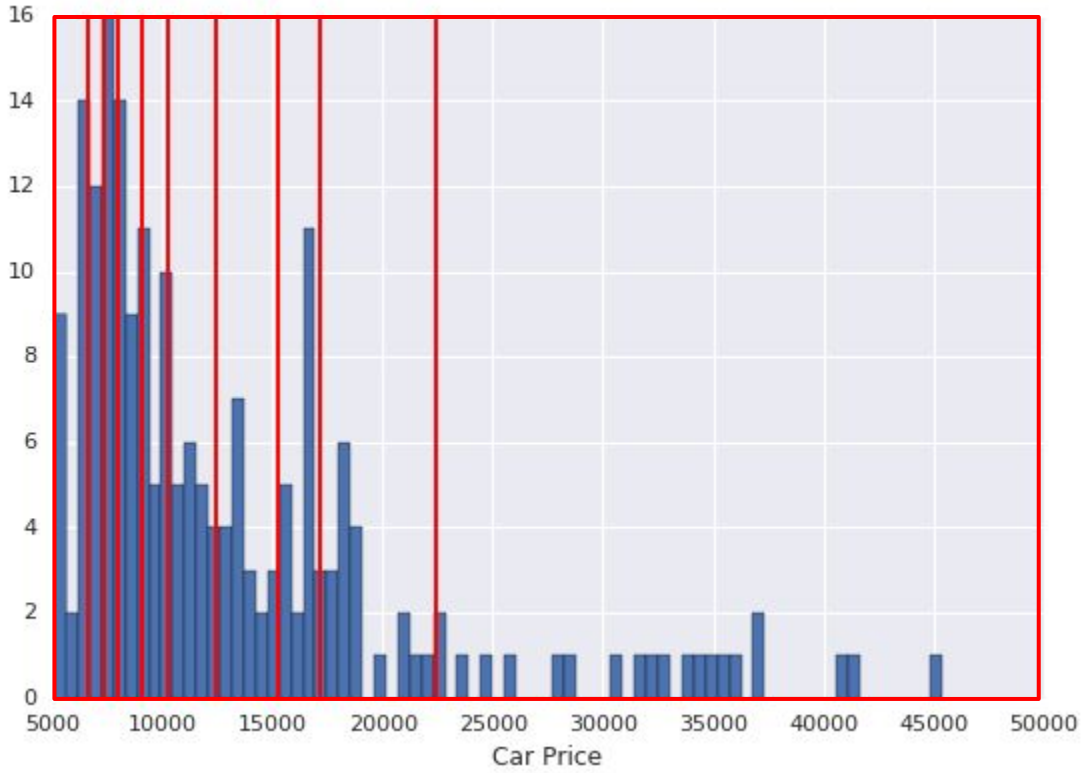
Selection of Bucket Boundaries



- We want a way to automate the selection of bucket boundaries
- Quantiles: Each bucket has roughly the same number of points.
- Another option is equal width bins.

Google

Creating Buckets by Quantiles



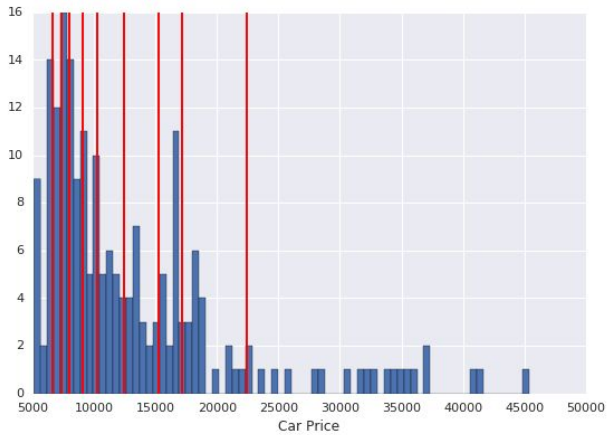
Each bucket is a Boolean variable with each car price falling into a single bucket.

car_price_under_6649
car_price_6649_to_7349
car_price_7349_to_7999
car_price_7999_to_9095
car_price_9095_to_10295
car_price_10295_to_12440
car_price_12440_to_15250
car_price_15250_to_17199
car_price_17199_to_22470
car_price_22470_and_up

Google

Creating Bins by Quantiles

By using quantiles all “expensive” cars get put in a bucket together allowing more resolution for cars in the price range of 90% of the cars in the data set.



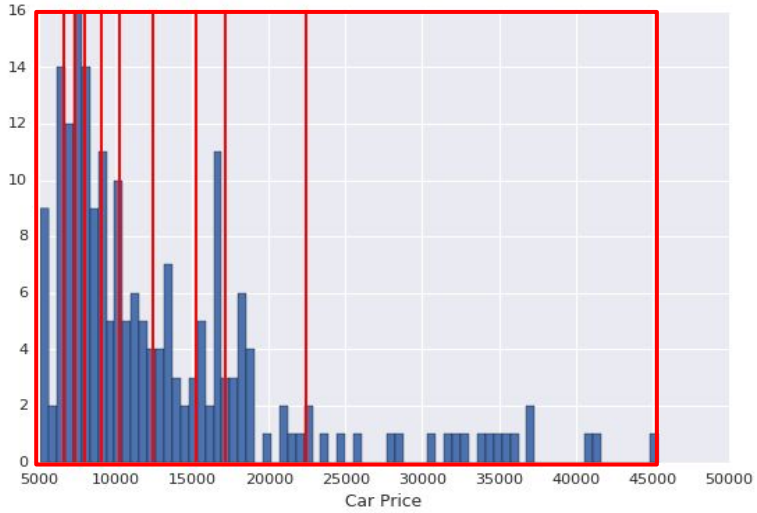
Bucket values for car of price $9000:

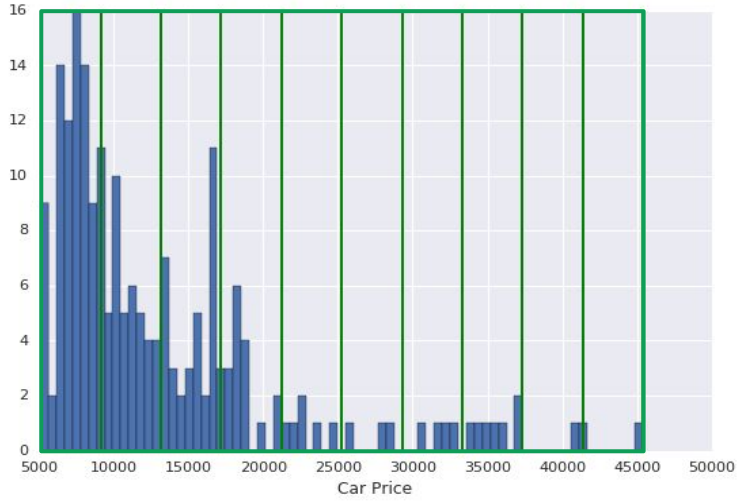
0	car_price_under_6649
0	car_price_6649_to_7349
0	car_price_7349_to_7999
1	car_price_7999_to_9095
0	car_price_9095_to_10295
0	car_price_10295_to_12440
0	car_price_12440_to_15250
0	car_price_15250_to_17199
0	car_price_17199_to_22470
0	car_price_22470_and_up

Google

Quantiles vs Equal Width Bins

What is the advantage of using quantiles (left) versus equal width bins (right)?





Google

Quantiles vs Equal Width Bins

- Both forms of binning provide non-linear behavior since the feature weight for each bin can be independently set
- Using quantiles gives more resolution in areas where there is more data
- For both techniques, you can adjust the number of bins to vary the amount of resolution versus the number of features introduced.

Representing Categorical Features

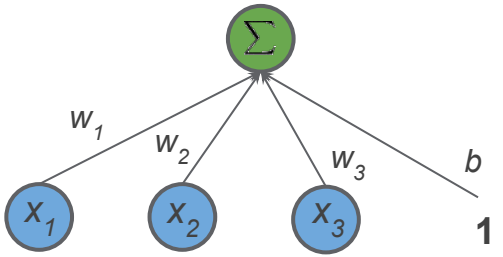
How can we represent features such as:

- The day of the week
- A person’s occupation(s)
- The words in an advertisement
- The movies a person has rated

Remember a linear model can only take a weighted combination of features.

Graphical View of a Linear Model

- In the graphical view below inputs are at the bottom and the output is at the top. Each input (or the constant 1 for the bias) is multiplied by the edge weight and they are summed together at the output node.



Output: $y' = w_1x_1 + w_2x_2 + w_3x_3 + b$

Input: $\boldsymbol{x} = (x_1, x_2, x_3)$

Transforming Categorical Features

The features can be mapped to numerical values. For example for the days of the week we could use:

Mon	Tues	Wed	Thur	Fri	Sat	Sun
0	1	2	3	4	5	6

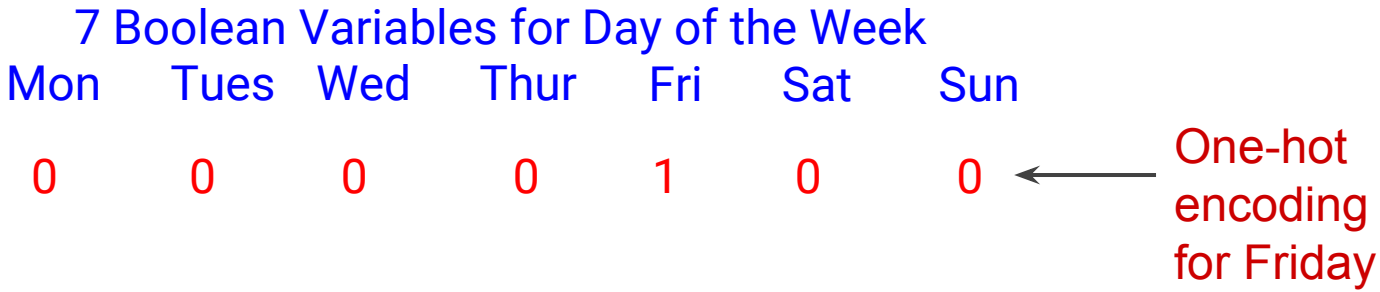
If we the value we are predicting increases linearly from Mon to Sun then we could directly use this, but typically you want to learn an independent weight for each value.

Categorical Features: One-Hot Encoding

- Introduce a Boolean variable for each feature value
- Independent weight is learned for each feature value.
- Example: For days of the week, introduce 7 Boolean variables each with its own learned weight.
- Sample one-hot encodings:

	7 Boolean Variables for Day of the Week						
	Mon	Tues	Wed	Thur	Fri	Sat	Sun
Encoding Tuesday	0	1	0	0	0	0	0
Encoding Friday	0	0	0	0	1	0	0
Encoding Sunday	0	0	0	0	0	0	1

Efficiently Representing One-Hot Encoding



- Map each feature value to a unique index

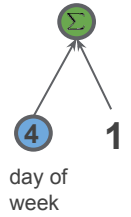
Efficiently Representing One-Hot Encoding

- Map each feature value to a unique index
- Represent a one-hot encoding as sparse vector using the index of the feature value that is not zero
- Encode Friday as <4> using the indices shown in purple.

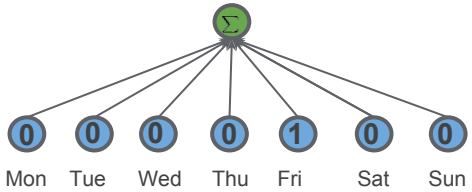
Google

One-Hot Encoding vs Single Numeric Feature

- Single numeric feature and weight
  - Represent Friday as the integer 4 that is multiplied by single model weight
- One-hot encoding with 7 Boolean variables for day of week
  - Represent Friday as the sparse vector <4> meaning the Boolean variable for Friday is 1 and the rest are 0.
- If the difference is not clear, please ask.



learns a weight and bias



Learns one bias per day of week since one input is 1 and the rest are 0

Google

Encoding Features that are Sets/Phrases

Sample ad: “1995 Honda Civic good condition”

Set Boolean variable for each word in ad to 1 and rest to 0:

- Sparse representation of ad: <102, 151, 200, 225, 350>

Google

Encoding Features that are Sets/Phrases

Sample ad: “1995 Honda Civic good condition”

Set Boolean variable for each word in ad to 1 and rest to 0:

indices ...	100	101	102	103	104		150	151	152		200	201		225	226		350
...	1993	1994	1995	1996	1997	…	poor	good	excellent	…	Honda	Toyota	...	Civic	Accord	….	condition
	0	0	1	0	0		0	1	0		1	0		1	0		1

- Sparse representation of ad: <102, 151, 200, 225, 350>
- We call this a sparse encoding since in the dense representation there are only a few 1s. A one-hot encoding is a special case with a single 1.

Google

Vocabulary for Categorical Features

Vocabulary: The mapping from the feature value to its unique index.

- Boolean variable is 1 if the feature has that value and 0 otherwise
- Represent as a one-hot or sparse vector using a set of the non-zero indices

... but sometimes we can't fit all possible feature values into the vocabulary

Vocabulary

0	Mon
1	Tues
2	Wed
3	Thur
4	Fri
5	Sat
6	Sun

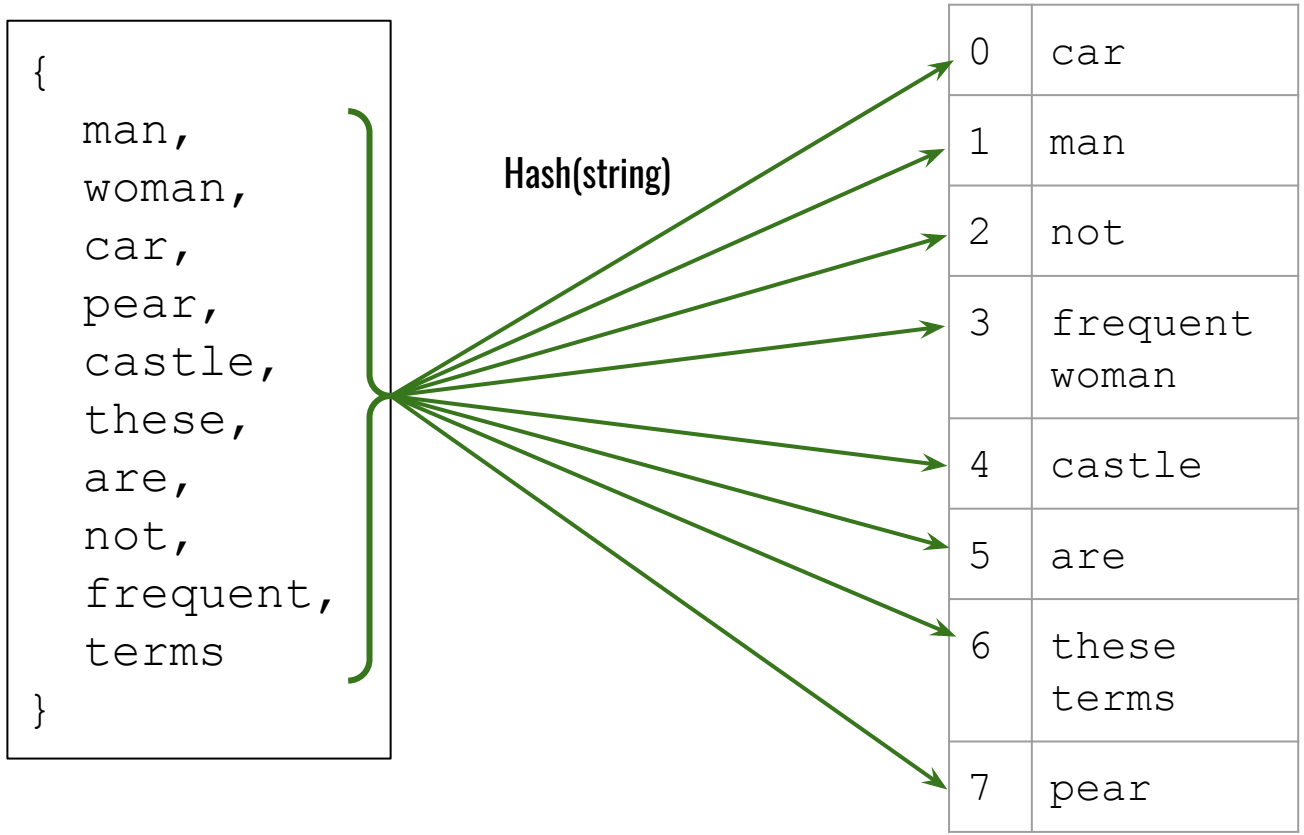
Vocabulary - Out of Vocab

Out of Vocabulary (OOV)

- Give unique indices to frequent terms/values
- Add one extra index to the vocabulary, OOV
- For items not in the vocabulary, assign to OOV

0	Super
1	Common
2	Terms
	...
N-1	Important
N	OOV

Hashing to Define Vocabulary Mapping



Google

Categorical vs Numerical Features

- How would you encode a feature such as zipcode?
- Though it could be used directly as a numerical feature, you don't want to multiply it by a weight.
- Best to treat zipcode as categorical data.
- You could use domain knowledge to group zipcodes that are geographically nearby into a single bucket.
- You need to think about raw features and how to best use them.

Feature Engineering: Missing Features

Often data sets are missing features. If this is extremely rare we could just skip those examples, but otherwise what do we do?

- For non-numerical data?

Feature Engineering: Missing Features

Often data sets are missing features. If this is extremely rare we could just skip those examples, but otherwise what do we do?

- For non-numerical data?
  - A common solution is to just introduce a feature value for “missing”
- How do we handle this for numerical data?

Feature Engineering: Missing Features

Often data sets are missing features. If this is extremely rare we could just skip those examples, but otherwise what do we do?

- For non-numerical data?
  - A common solution is to just introduce a feature value for “missing”
- How do we handle this for numerical data?
  - Use the average value (or some common value)
  - Bin the data and introduce a “missing” bin

Add Non-Linearities To a Linear Model

- Linear models can represent a lot more problems when we add **non-linearities** to the features
- We've already seen one way to do this:
  - Bucketizing Numerical Features
- Other ways to introduce non-linearities:
  - Feature Crosses
  - Adding Artificial Variables

Preview: Feature Crosses

- We will study feature crosses in more depth later.
- In a linear model the contribution captured for each feature is independent of the others and this is often not the case in data.
- Feature Crosses introduce non-linear behavior between a set of two or more features by capturing dependencies between the features.

Feature Cross Example

Rainfall

None	Low	Med	High
w1	*w2*	*w3*	*w4*

Temperature

Cold	Cool	Warm	Hot
w5	*w6*	*w7*	*w8*

Rainfall x Temperature

	Cold	Cool	Warm	Hot
None	*w9*	*w10*	*w11*	*w12*
Low	*w13*	*w14*	*w15*	*w16*
Med	*w17*	*w18*	*w19*	*w20*
High	*w21*	*w22*	*w23*	*w24*

- There is one weight for each feature and for all 16 possible combinations of these feature values.
- Color encodes the value of the weight with red being low and green being high

Google

Feature Crosses: Some Examples

- **Housing market price predictor:**
  `[latitude X longitude x num_bedrooms]`

- **Predictor of pet owner satisfaction with pet:**
  `[pet behavior type X time of day]`

- **Tic-Tac-Toe predictor:**
  `[pos1 x pos2 x ... x pos9]`

Visualization of Weights with a Cross

Rainfall

None	Low	Med	High
w1	*w2*	*w3*	*w4*

Temperature

Cold	Cool	Warm	Hot
w5	*w6*	*w7*	*w8*

Rainfall x Temperature

	Cold	Cool	Warm	Hot
None	*w9*	*w10*	*w11*	*w12*
Low	*w13*	*w14*	*w15*	*w16*
Med	*w17*	*w18*	*w19*	*w20*
High	*w21*	*w22*	*w23*	*w24*

- There is one weight for each feature and for all 16 possible combinations of these feature values.
- Color encodes the value of the weight with red being low and green being high

Google

Feature Crosses: Why would we do this?

- Linear learners scale well to massive data
- But without feature crosses, the expressivity of these models would be limited
- Using feature crosses + massive data is one efficient strategy for learning highly complex models
  - Foreshadowing: Neural nets provide another
- Are there downsides to adding more and more crosses?

Feature Engineering: Be Creative

- We've just seen some examples but the key is to think about your data, what a linear model can do and then how to best capture that.
- As one more example, if you wanted to predict the rental price for an apartment and you had a street address, how might you represent that?
- Remember, there's not one right answer. Sometimes you need to try a few things and see what gives you the best predictions.

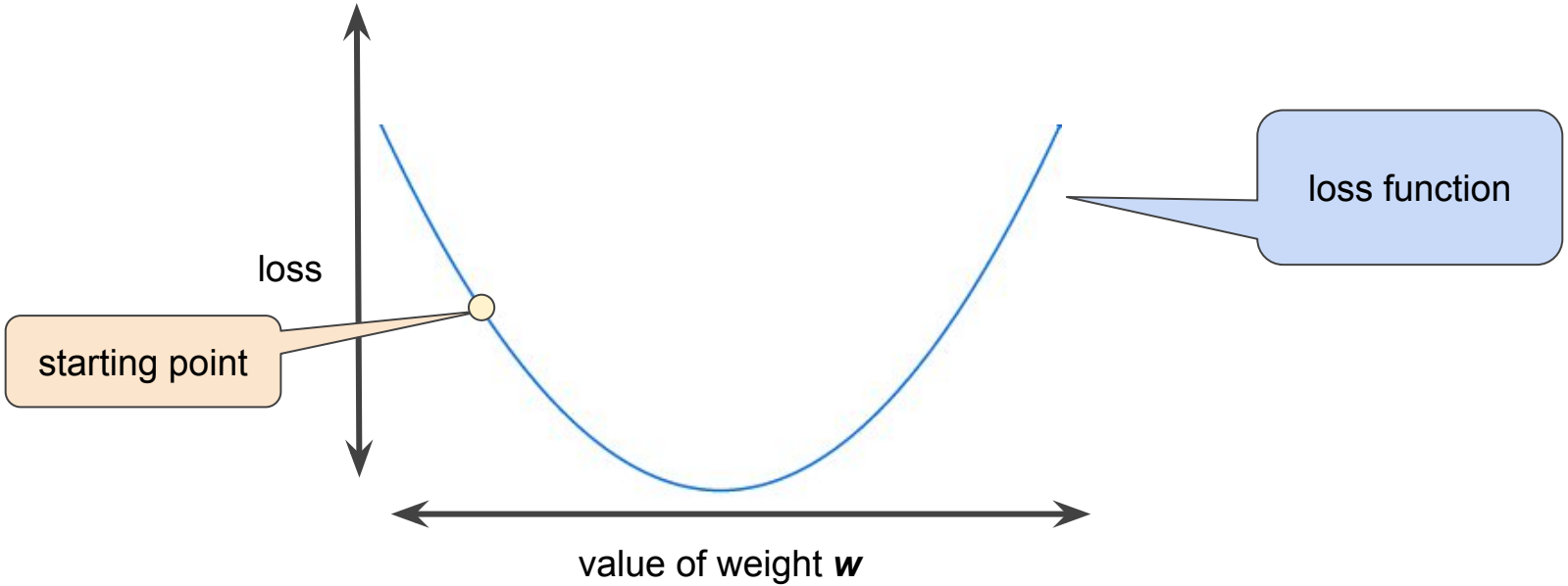
Stochastic Gradient Descent: A Closer Look

Google

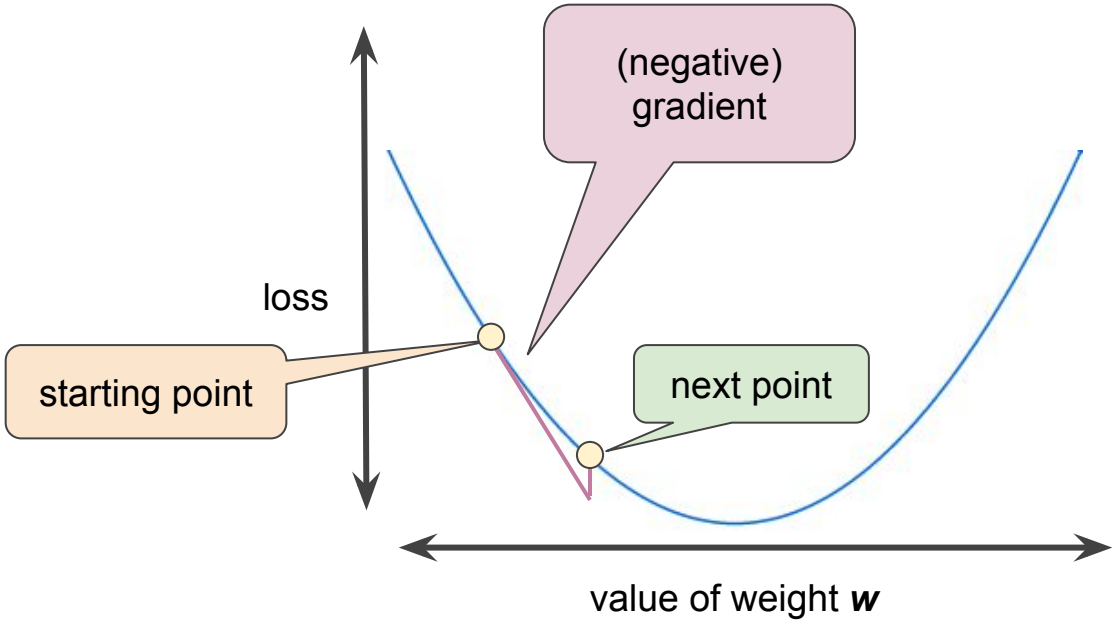
Review of Overview of Gradient Descent

- Derivative of $(y - y')^2$ w.r.t. the model weights ***w*** tells us how loss changes in the neighborhood of weights we're currently using.
- A gradient is the generalization of a derivative when you have more than one variable.
- We can take small steps in the negative gradient direction to modify the weights so that the loss on that example is lower.
- This optimization strategy is called Gradient Descent.

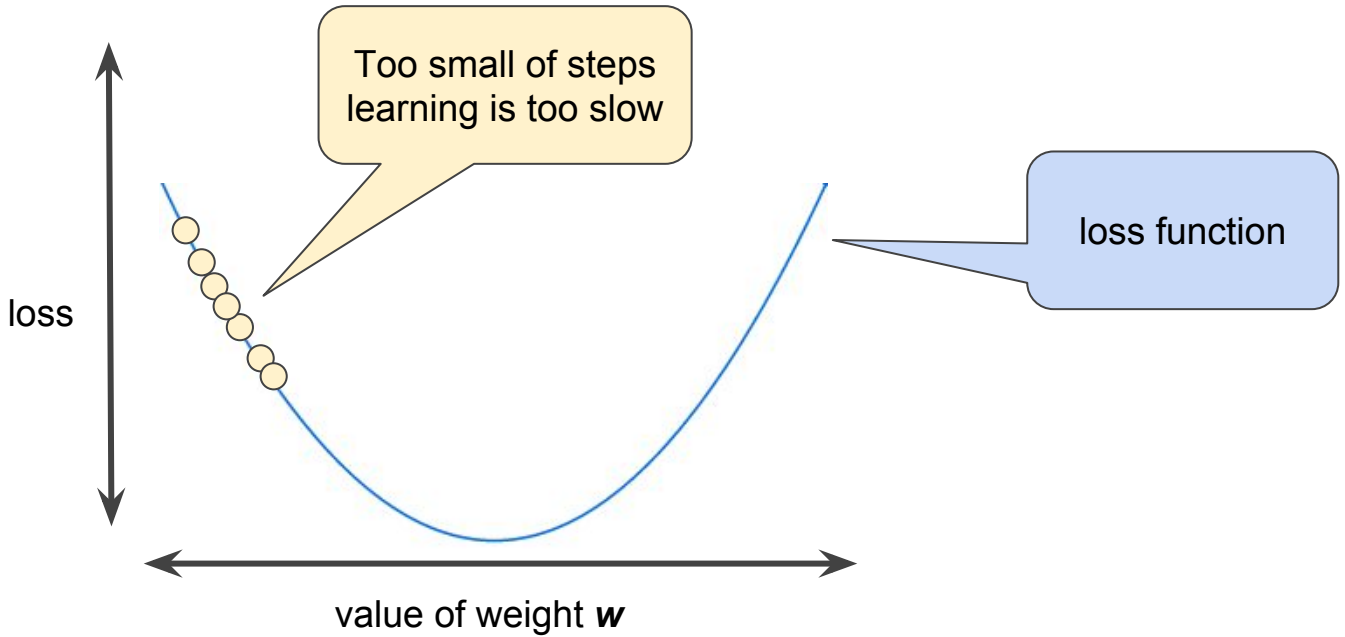
Training: Decreasing the Loss



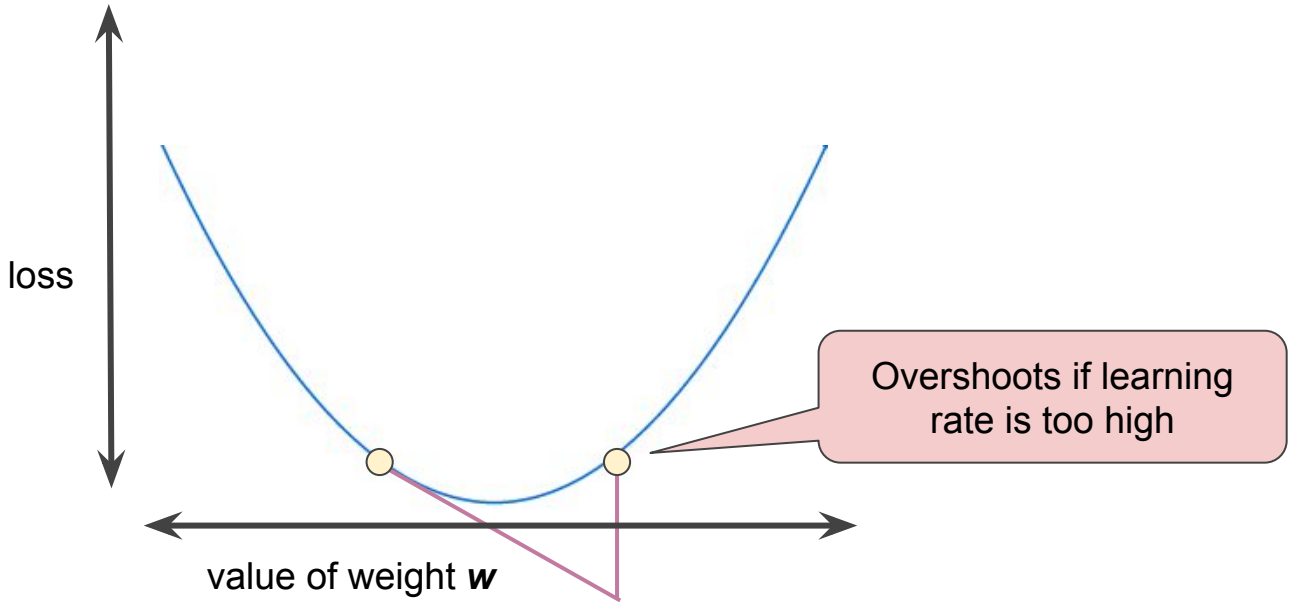
Training: A Gradient Step



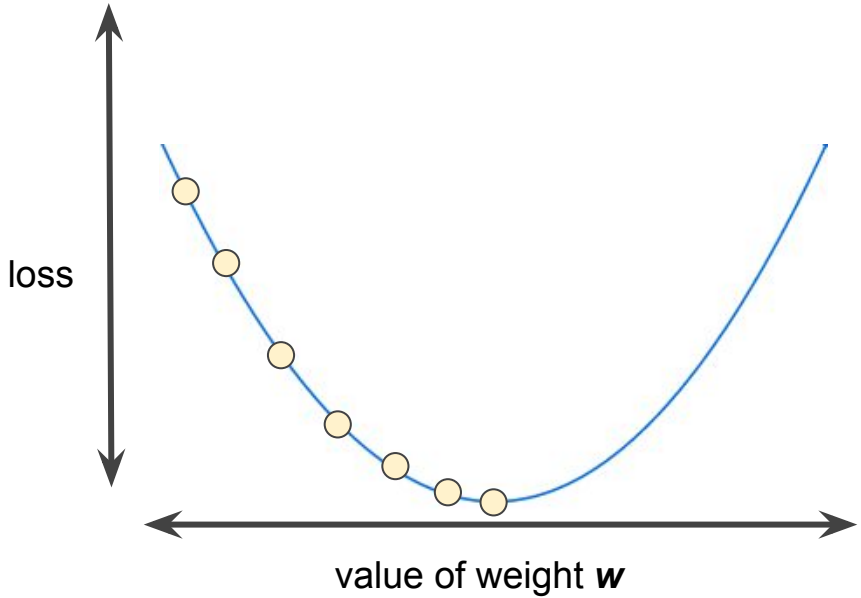
Learning Rate (Size of Step) Too Small



Learning Rate (Size of Step) Too Large

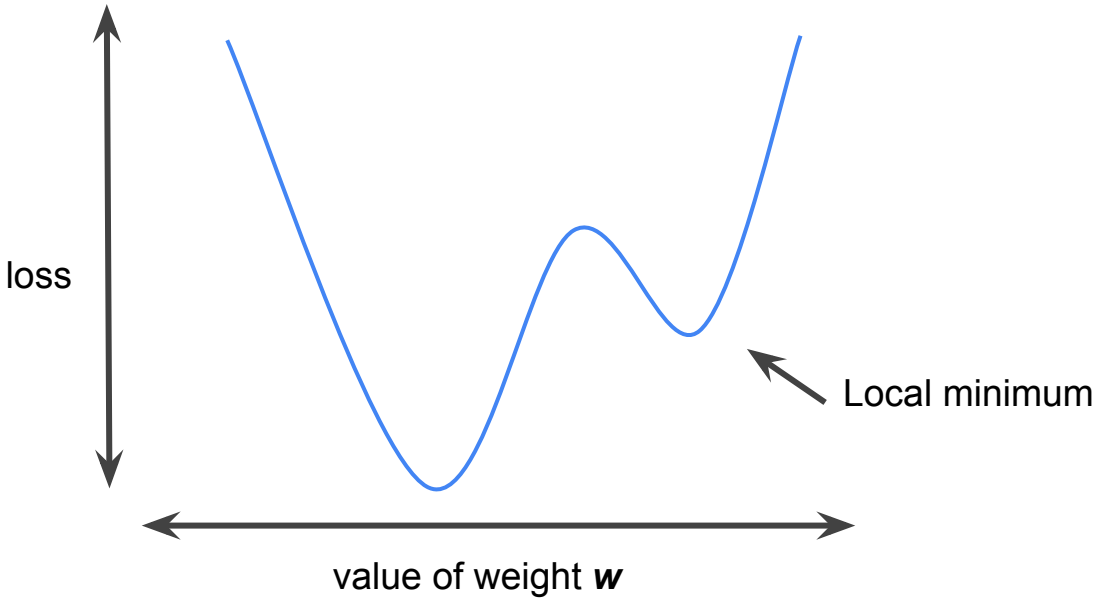


Well-Tuned Learning Rate



Local Minimum

When the loss function we are optimizing is not convex we can get stuck in local minimum



Mini-Batch Gradient Descent

- Computing gradient over the entire dataset for every step is expensive and unnecessary
- **Mini-Batch Gradient Descent**: Compute gradient on batches typically of 10-1000 samples

Stochastic Gradient Descent (SGD)

- Mini-Batch Gradient Descent using batches that are random samples of examples
- Random samples allow the gradient estimate to be more accurate.
- Many data sets are arranged in some order (e.g. imagine if the automobile data set was sorted by price).
- Easy way to get random samples is to randomly “shuffle” the data and then just take first batch-size examples, then next batch-size examples, ...

Update Step in SGD

- At a high level what happens in each step is

$$w_{\text{new}} = w_{\text{old}} - \bigtriangledown \cdot \text{learning\_rate}$$

This is the gradient for the weight being updated as estimated from the labeled examples in the batch

- Remember *w* and *b* are instance variables that are part of the internal state of the estimator object that are updated via this formula when the fit method is called.

Update Step in SGD (cont)

- To make this concrete let's consider when $y' = wx + b$ and we are minimizing $(y' - y)^2$. So for w
- ∇ is $\frac{d}{dw}(y'-y)^2 = 2(y'-y)\frac{d}{dw}(y'-y) = 2(y'-y)\frac{d}{dw}(wx+b-y) = 2(y'-y)x$
- So at each step: $w = w - 2(y'-y)\cdot x \cdot \text{learning\_rate}$

Update Step in SGD (cont)

- To make this concrete let's consider when $y' = wx + b$ and we are minimizing $(y' - y)^2$. So for w
- ∇ is $\frac{d}{dw}(y'-y)^2 = 2(y'-y)\frac{d}{dw}(y'-y) = 2(y'-y)\frac{d}{dw}(wx+b-y) = 2(y'-y)x$
- So at each step: $w = w - 2(y'-y)\cdot x \cdot \text{learning\_rate}$
- Similarly, at each step: $b = b - 2(y'-y) \cdot \text{learning\_rate}$

Update Step in SGD (cont)

- To make this concrete let's consider when $y' = wx + b$ and we are minimizing $(y' - y)^2$. So for w
- ∇ is $\frac{d}{dw}(y' - y)^2 = 2(y' - y)\frac{d}{dw}(y' - y) = 2(y' - y)\frac{d}{dw}(wx + b - y) = 2(y' - y)x$
- So at each step: $w = w - \boxed{2(y' - y) \cdot x} \cdot \text{learning\_rate}$
- Similarly, at each step: $b = b - \boxed{2(y' - y)} \cdot \text{learning\_rate}$
- The gradient highlighted in yellow with a red outline is computed by taking the average value over the examples in the batch.
- When $y' < y$ (prediction too small), $y' - y$ is negative so w gets bigger bringing the prediction closer y. If $y' > y$, w gets smaller.

Generalization

Will Our Model Make Good Predictions?

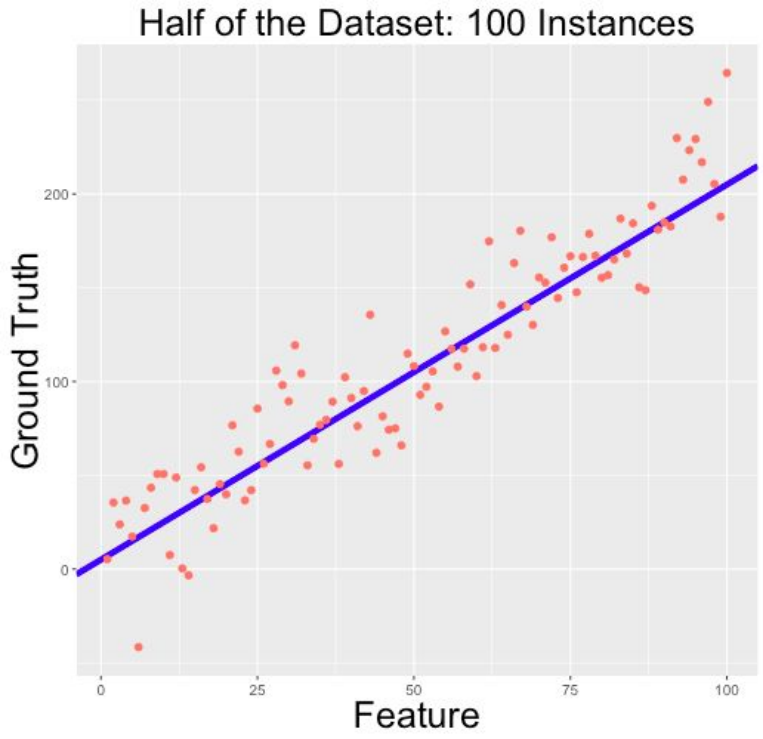
- **Our Key Goal**: predict well on new unseen data.
- **Problem**: We only get a sample of data D for training
- We can measure the loss for our model on the sample D but how can we know if it will predict well on new data?
- Performance Measure: A measure of how well the model predicts on unseen data. This is usually related to the loss function.

Example Performance Measure

For regression a common performance metric is root mean squared error (RMSE)

$$\text{RMSE} = \sqrt{L_2\text{Loss}} = \sqrt{\frac{1}{|D|}\sum_{(\mathbf{x},y)\in D}(y - y')^2}$$

On this dataset, for the best linear model: RMSE = 22.24



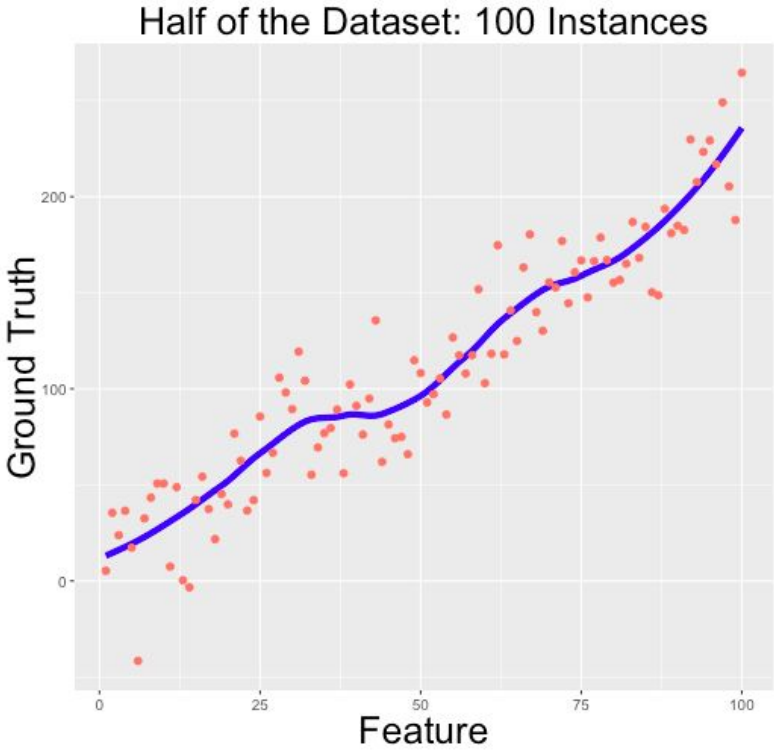
Google

Non-Linear Model

What if we use a model that is not a line (called *non-linear*)

For the model on the right:
RMSE = 21.44

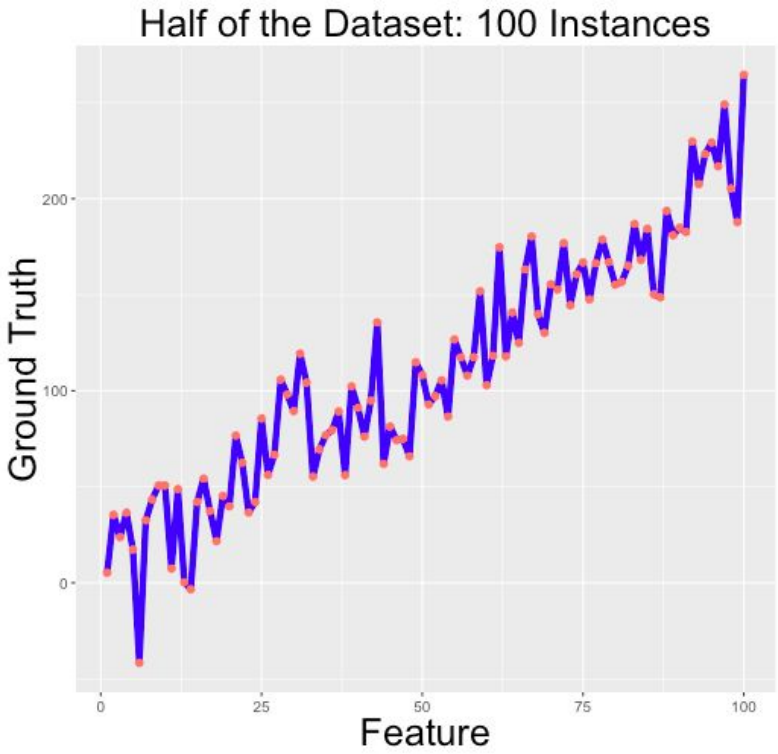
This is better than the linear model with RMSE = 22.24



“Perfect” Model

If we make a very complex model then we can perfectly fit the data, so RMSE = 0

Why might you not want to do this?

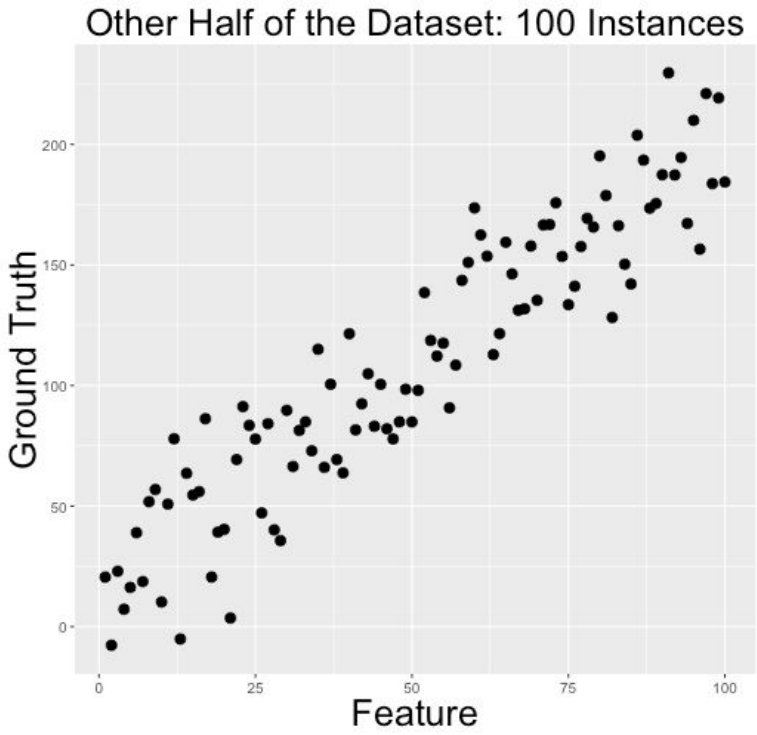


Google

Measuring Generalization Ability

Here's the other half of the data.

We will discuss this more but for now, let's refer to this "other half" as the test data and the original points as the training data since they were used to train the model.



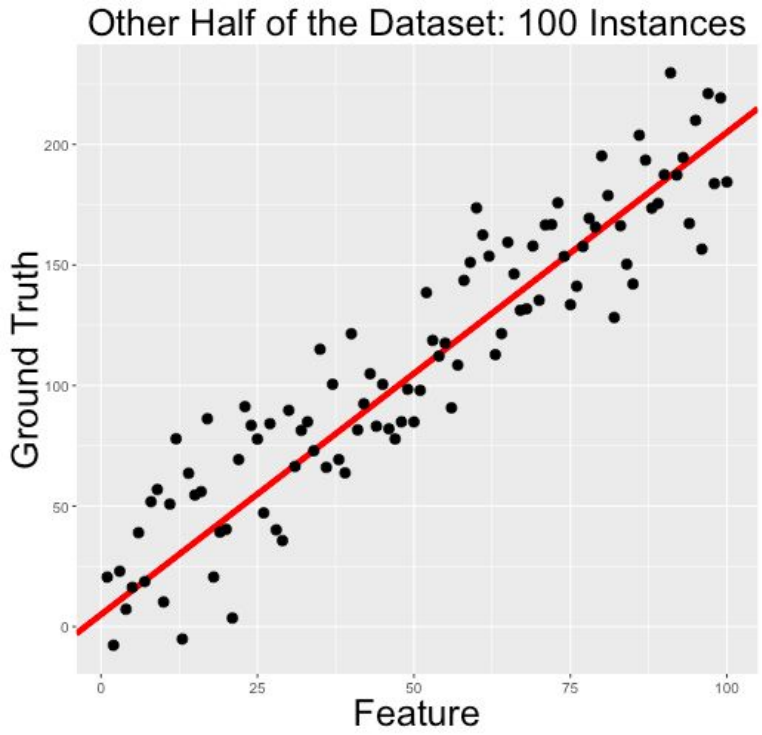
Google

Generalization for Linear Model

We'll look at our performance using the best linear model.

Training RMSE = 22.24
Test data RMSE = 21.98

Pretty similar.



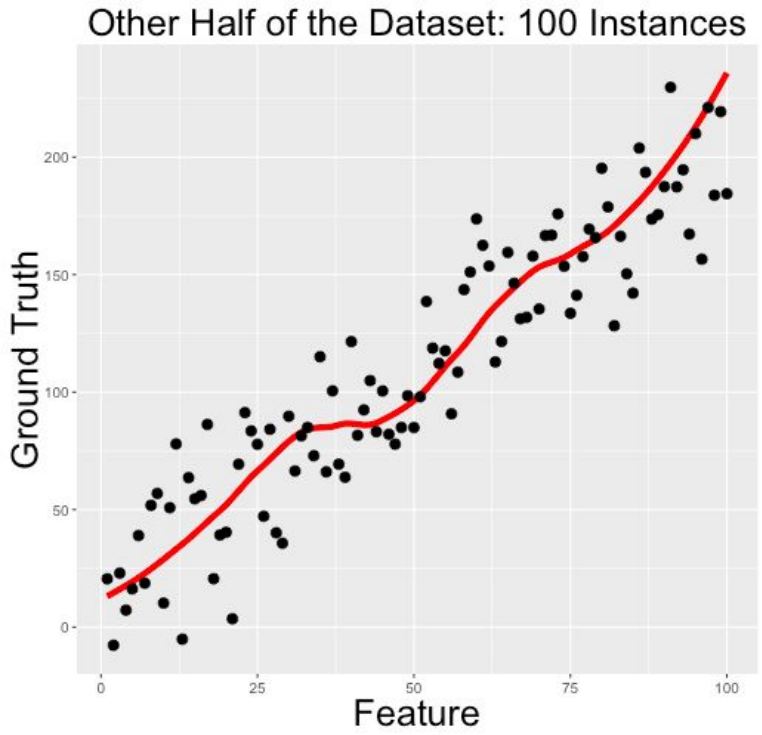
Google

Generalization for Non-Linear Model

We'll look at our performance using the non-linear model.

Old RMSE = 21.44
New RMSE = 22.74

Still pretty similar.



Google

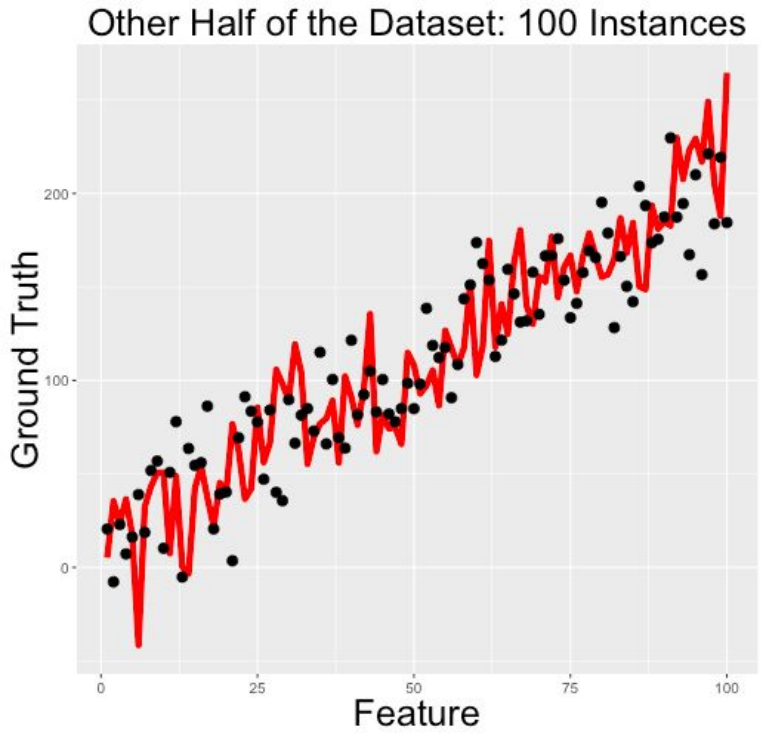
Generalization for “Perfect” Model

Training data RMSE = 0

Test data RMSE = 32

This did not generalize to new data!

This is called overfitting.

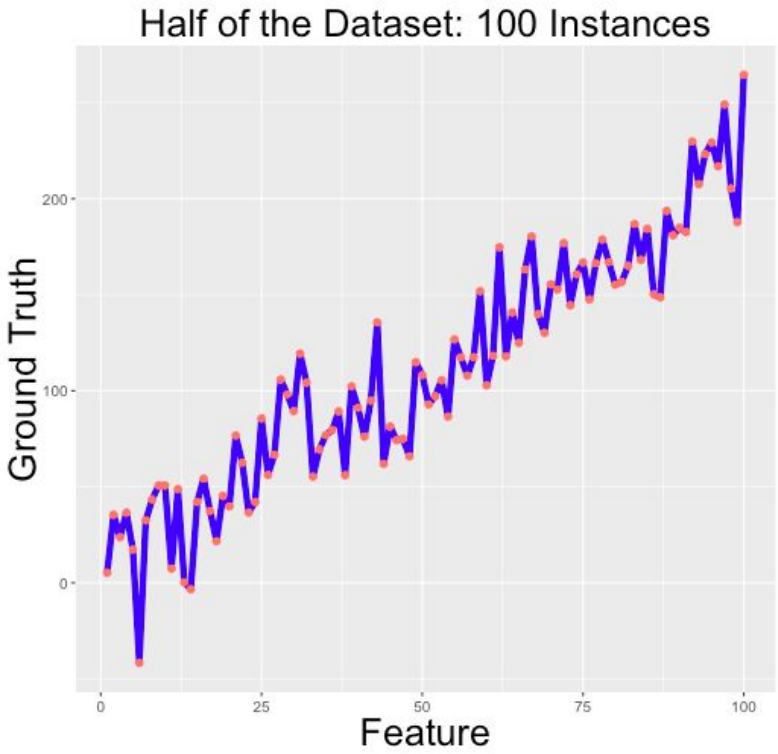


Google

Overfitting

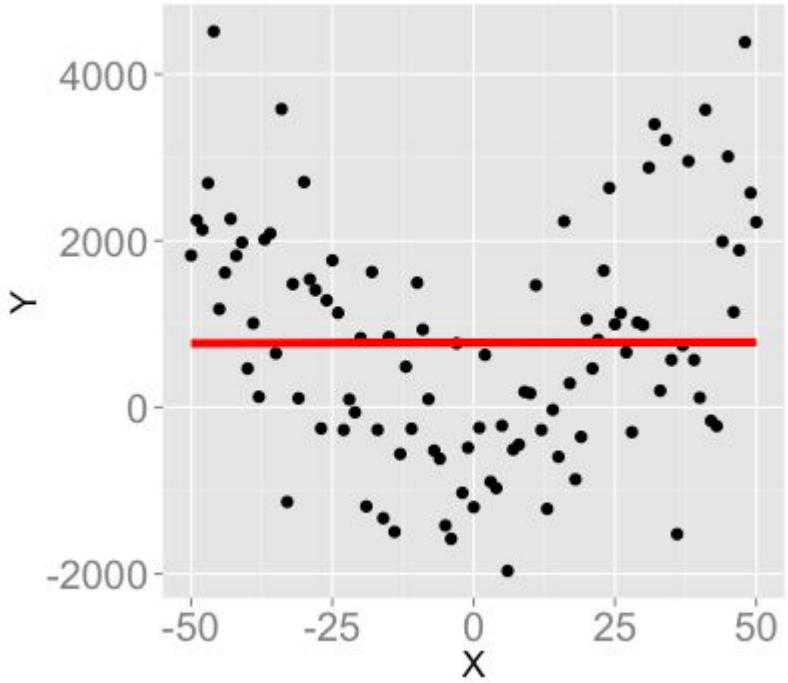
If we make a very complex model then can perfectly (or near perfectly) fit the training data we just memorize versus the goal of generalizing.

Remember our goal is to build a system to deal with new data!



Google

An Underfit Model



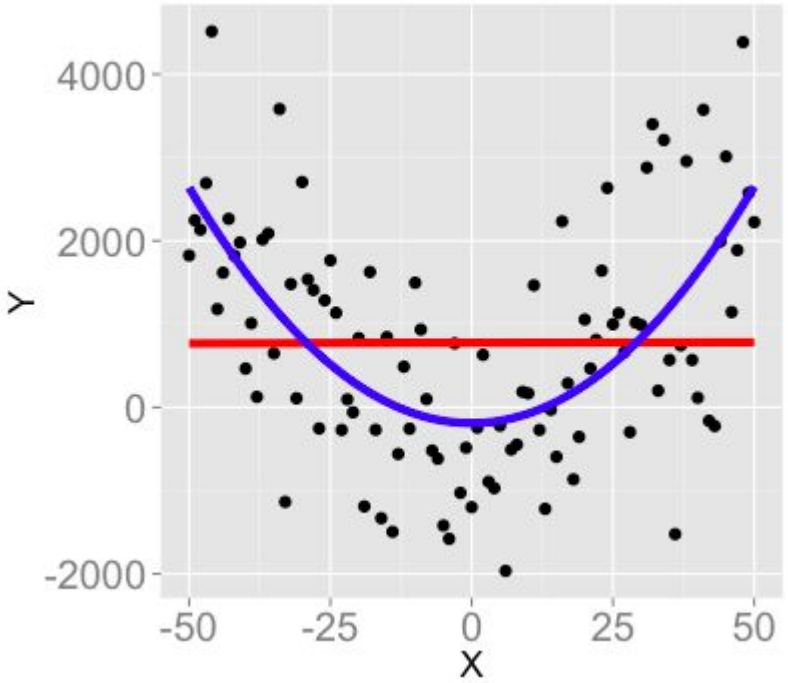
Underfitting happens when you try to use a model that is too simple.

Google

How do we know if our model is good?

- William of Occam (back in 14th century) argued that simple explanations of nature are better
- Occam's Razor principle: the less complex a model is, the more likely to predict new data well
- How can we define how complex a learning model is?
- How can we measure how well our model generalizes?

Model Complexity



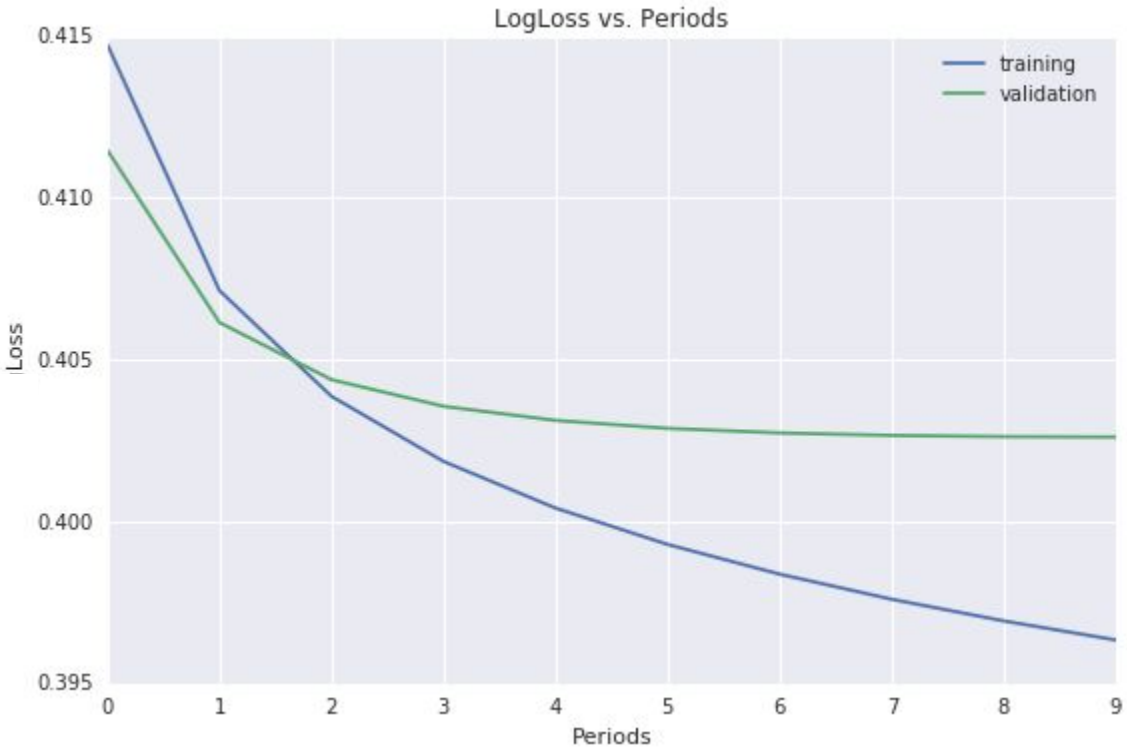
Our goal is to determine what model complexity is most appropriate.

We'll discuss ways to do this.

How do we know if our model is good?

- In practice to determine if our model do well on a new sample of data we use a new sample of data that we call the **test set**
- Good performance on the test set is a useful indicator of good performance on the new data as long as:
  - The test set is large enough
  - The test set is independent of the training set so it truly represents new data
  - We don’t cheat by using the test set over and over

Generalization Curve and Overfitting



A learning curve that is showing overfitting beginning to occur.

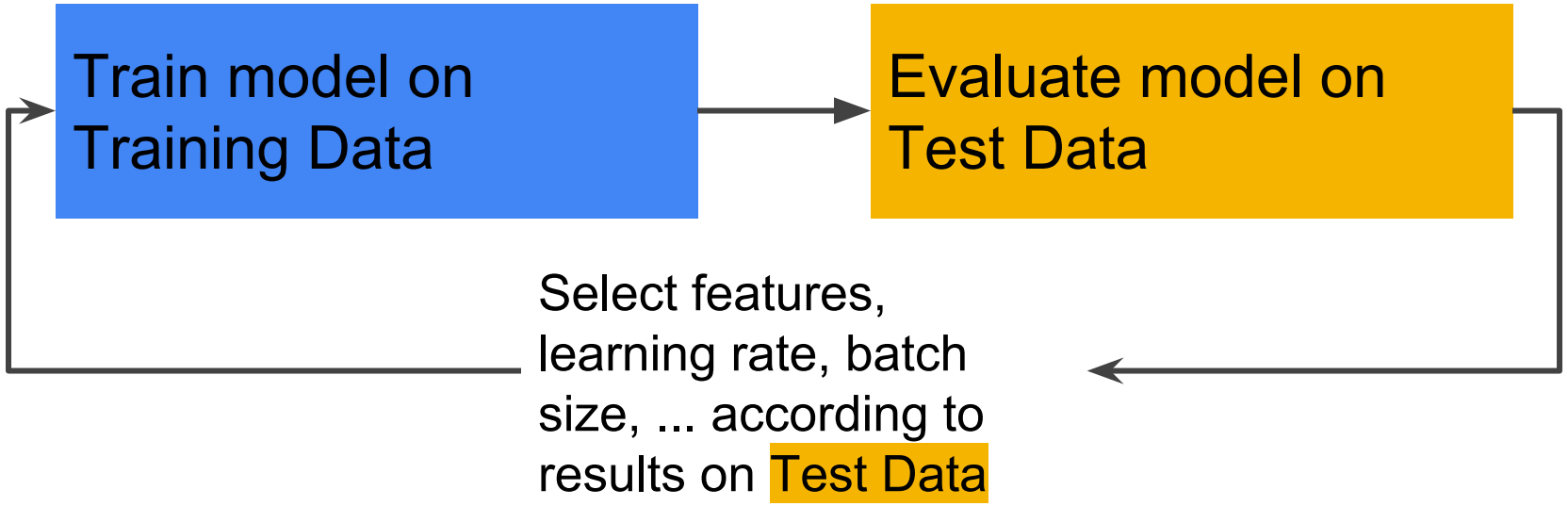
Google

Training, Validation, and Test Data Sets

Partitioning Data Sets

- Set aside some of the data as test data
  - Often just do at random
  - Sometimes use most recent data as test data
  - You need to be very careful here

A Possible Workflow?



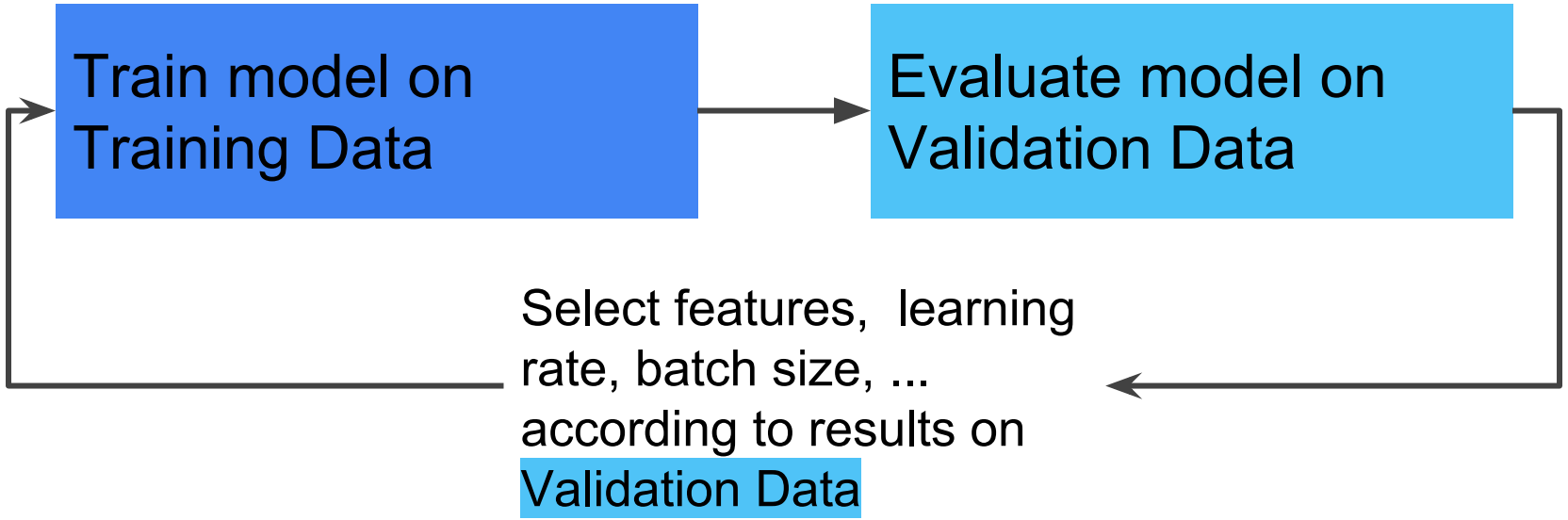
Pick the model that does best on Test Data.

Any issues here?

A Solution to “Polluting” Test Data

- Divide the data we are provided for training our model into two datasets
  - Most of it will be in our Training Data
  - A portion of it (typically 5-10%) will be used as a Validation Data.

Better Workflow: Use a Validation Data



Pick model that does best on Validation Data
Check for generalization ability on Test Data

Partitioning Data Sets

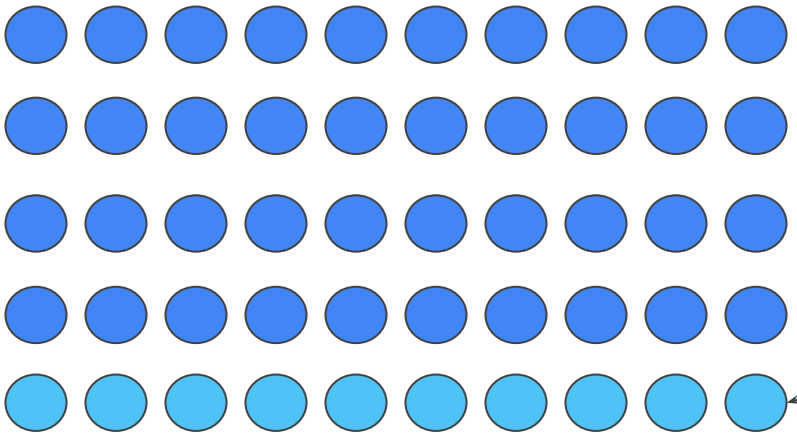
- You need to be very careful when partitioning your data into training, validation and test sets.
- You will explore this some in your next lab.
- Let’s look at some real-life examples showing some subtle but significant errors you can make that would lead you to believe you have a really good model but in practice one that will not generalize to new data.

k-fold Cross Validation

- Test data must be set aside unless there is no concern about overfitting
- What if we don’t have enough data to set aside enough for validation data?
- For these cases k-fold cross validation is often used
- Basic idea is to divide the data into k roughly even size pieces and in each of k training phases uses 1 piece as validation and the other k-1 as training data

Illustration of k-fold Cross-Validation

Demonstrate with k=5 where each row represents ⅕ of the data.



Training Set

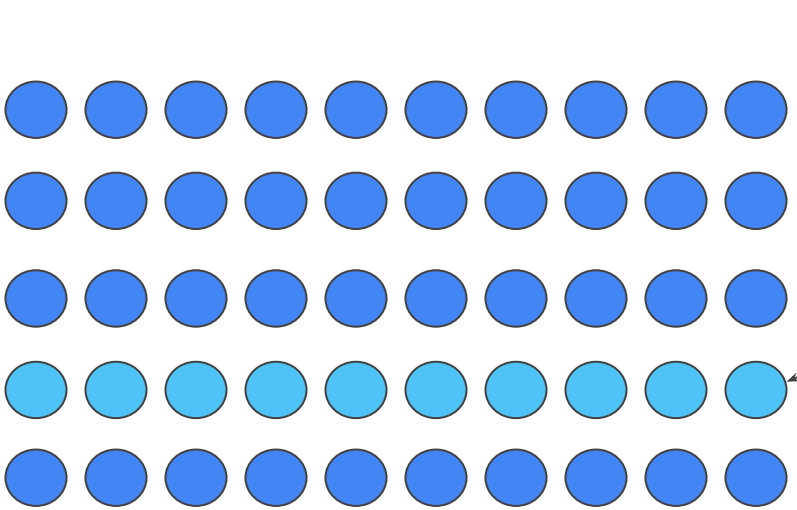
Validation Set

Performance measure m_1 computed just on the validation set for this fold

Google

Illustration of k-fold Cross-Validation

In second training phases shift which points are used for validation.



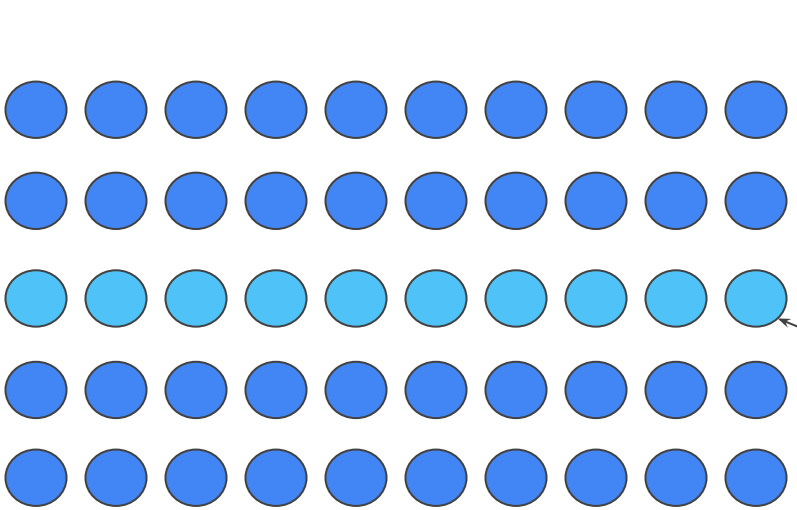
Training Set

Validation Set

Performance measure m_2 computed just on the validation set for this fold

Illustration of k-fold Cross-Validation

Continue shifting which points are used for validation.



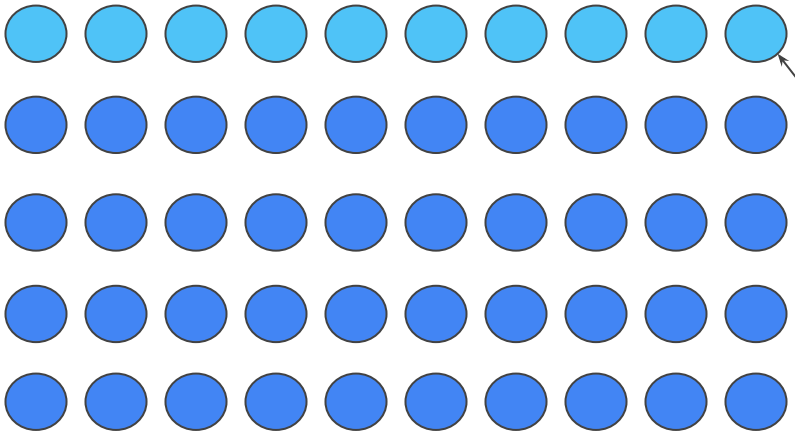
Training Set

Validation Set

Performance measure m_3 computed just on the validation set for this fold

Google

Illustration of k-fold Cross-Validation



Training Set

Validation Set

Performance measure m_5 computed just on the validation set for this fold

Cross-Validation: Compute Metric

- For each of the k training phases, compute the performance metric (over the validation set for that phase). This gives us $m_1, m_2, \ldots, m_k$
- Average $m_1, m_2, \ldots, m_k$ to get an aggregate performance metric.
- You can also check model stability by comparing the performance across the *k* runs and also compute standard statistical measures such as standard deviation and error bars over the *k* folds.

Cross-Validation: Train Final Model

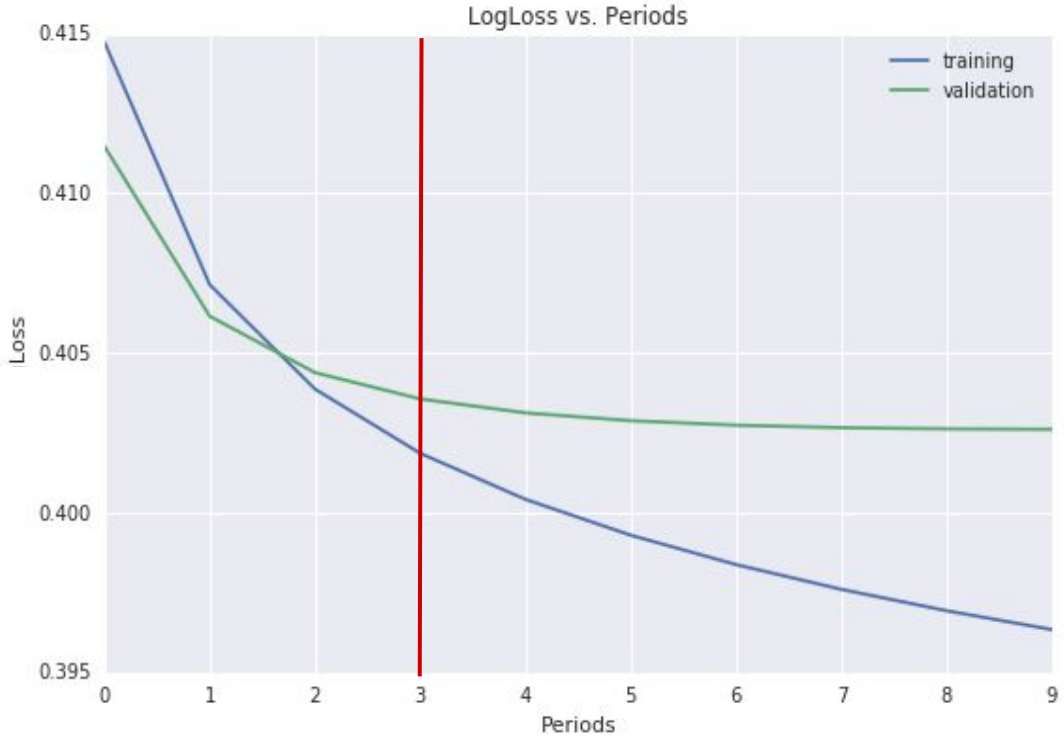
- To train the final model, choose the hyperparameter setting that gives you best aggregated performance over the k runs.
- Now run the algorithm with the chosen hyperparameters using all examples (other than those set aside as test data throughout) as the training data to obtain the final model.
- Use the test data, which has not been used during cross validation to check for any issues with overfitting.

k-fold Cross-Validation: Pros and Cons

- The advantage of k-fold cross validation is that we can train on *(k-1)/k* of the data each phase and that all points are used for validation.
- The disadvantage is that *k* different models need to be trained for every set of hyperparameters being considered, which is slow.
- Only use k-fold cross-validation if you don't have enough labeled data to split into independent train, validate and test sets.

Regularization for Simplicity

Ways to Prevent Overfitting



Early stopping: Use learning curve to detect overfitting. Stop training at somewhere around the red line.

Are there better approaches?

Google

Penalizing Model Complexity

- We want to avoid model complexity where possible.
  - remember Occam's razor
- We can introduce this idea into the optimization we do at training time.
- Empirical Risk Minimization:

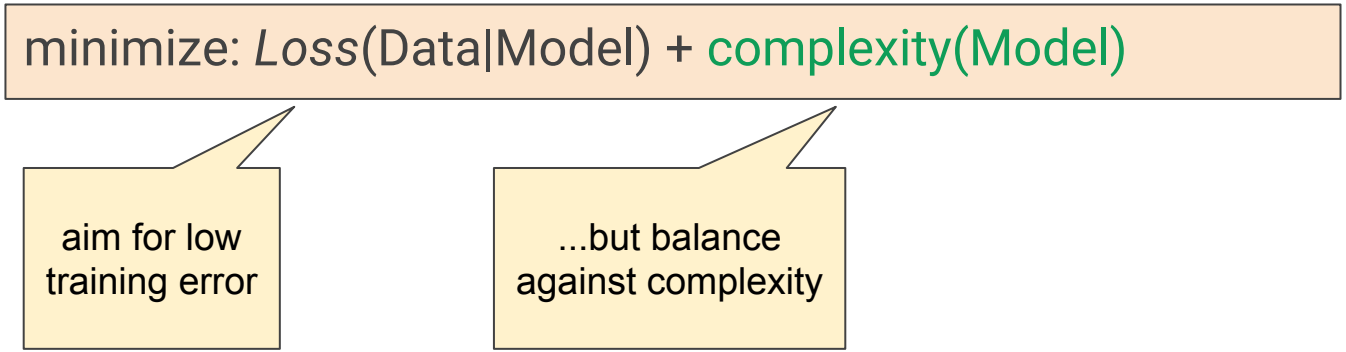
minimize: *Loss*(Data|Model)

aim for low training error

Google

Penalizing Model Complexity

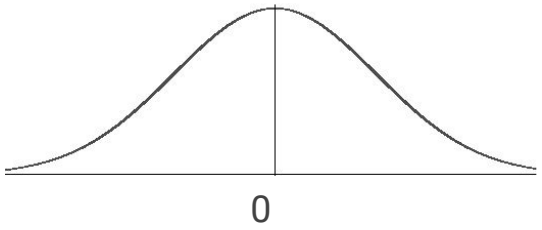
- We want to avoid model complexity where possible.
  - Occam's razor
- We can introduce this idea into the optimization we do at training time.
- Structural Risk Minimization:



Google

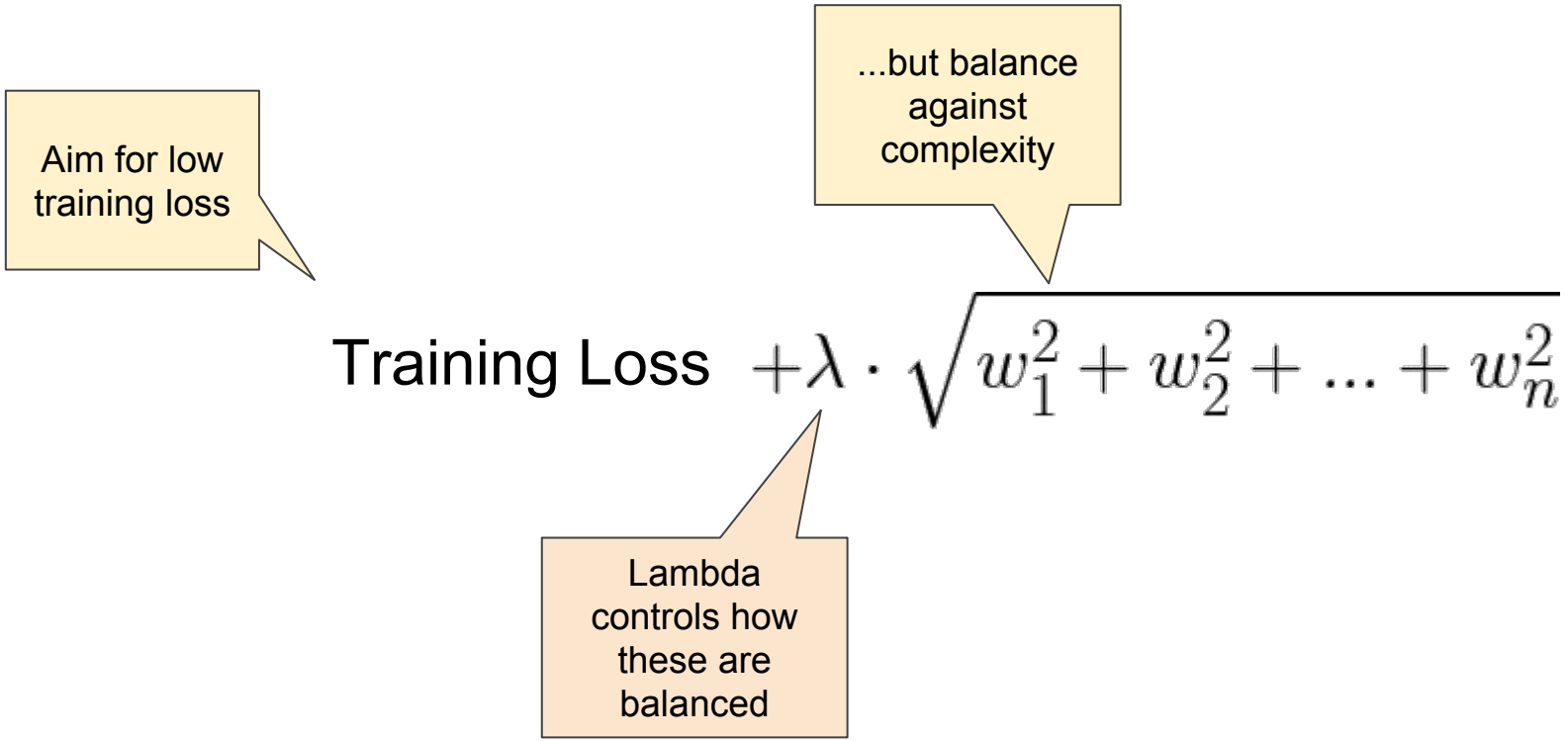
Regularization

- How to define *complexity(model)*?
- One possible "prior belief" is that weights should be normally distributed around 0
- Diverging from this should incur a cost



- Can encode this idea via **L_2 regularization**
  - *complexity(model) = sum of squares of the weights*
  - a.k.a. square of the L_2 norm of the weight vector
  - Encourages small weights centered around zero

L_2 Regularization



Linear Classifier

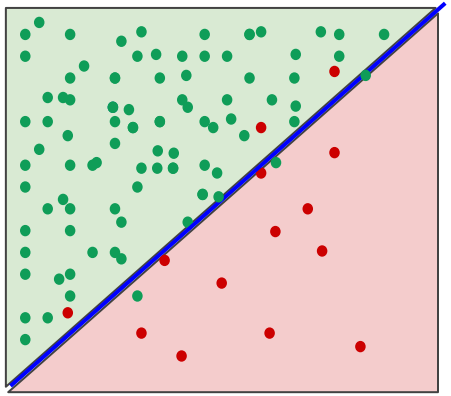
Logistic Regression

- Many problems require a probability estimate as output.
- **Logistic Regression** is focused on such problems.
- Handy because the probability estimates are **calibrated**.
  - for example, prob(click) * bid = expected revenue
- Also useful for when we need a binary classification
  - click or not click? → prob(click)
  - spam or not spam? → prob(spam)

Logistic Regression For Classification

- Suppose we have a linear classifier to predict if email is spam or not-spam.

- The goal is to modify how we define and train a linear model in a way that we can estimate a probability that an email is spam versus just give a classification.

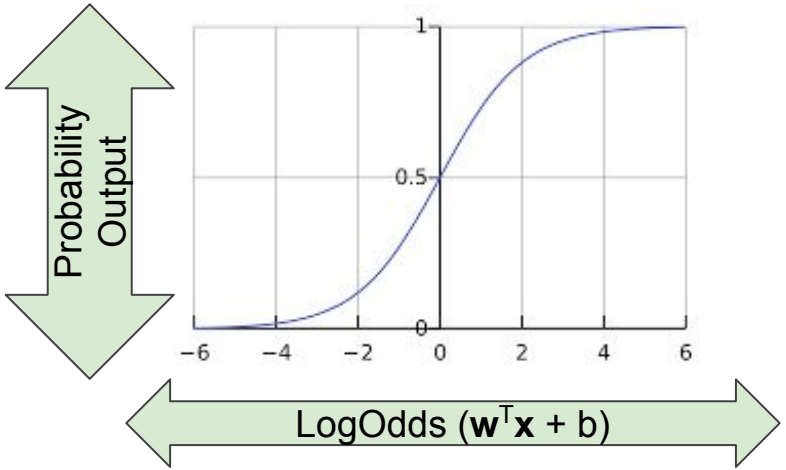


Logistic Regression: Predictions

$$y' = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x}+b)}}$$

linear model

pass linear model through a sigmoid (pictured to the right)

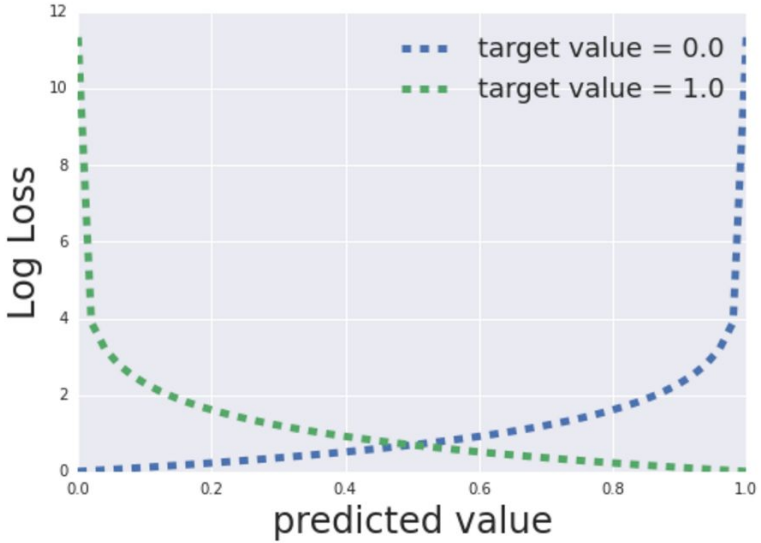


Recall that: $\mathbf{w}^T\mathbf{x} = w_1x_1 + w_2x_2 + \cdots + w_nx_n$

Google

LogLoss For Predicting Probabilities

Close relationship to Shannon's Entropy measure from Information Theory



$$LogLoss = \sum_{(\mathbf{x},y)\in D} -y\log(y') - (1-y)\log(1-y')$$

Google

Logistic Regression and Regularization

Regularization is super important for logistic regression.

- Remember the asymptotes
- It'll keep trying to drive loss to 0 in high dimensions

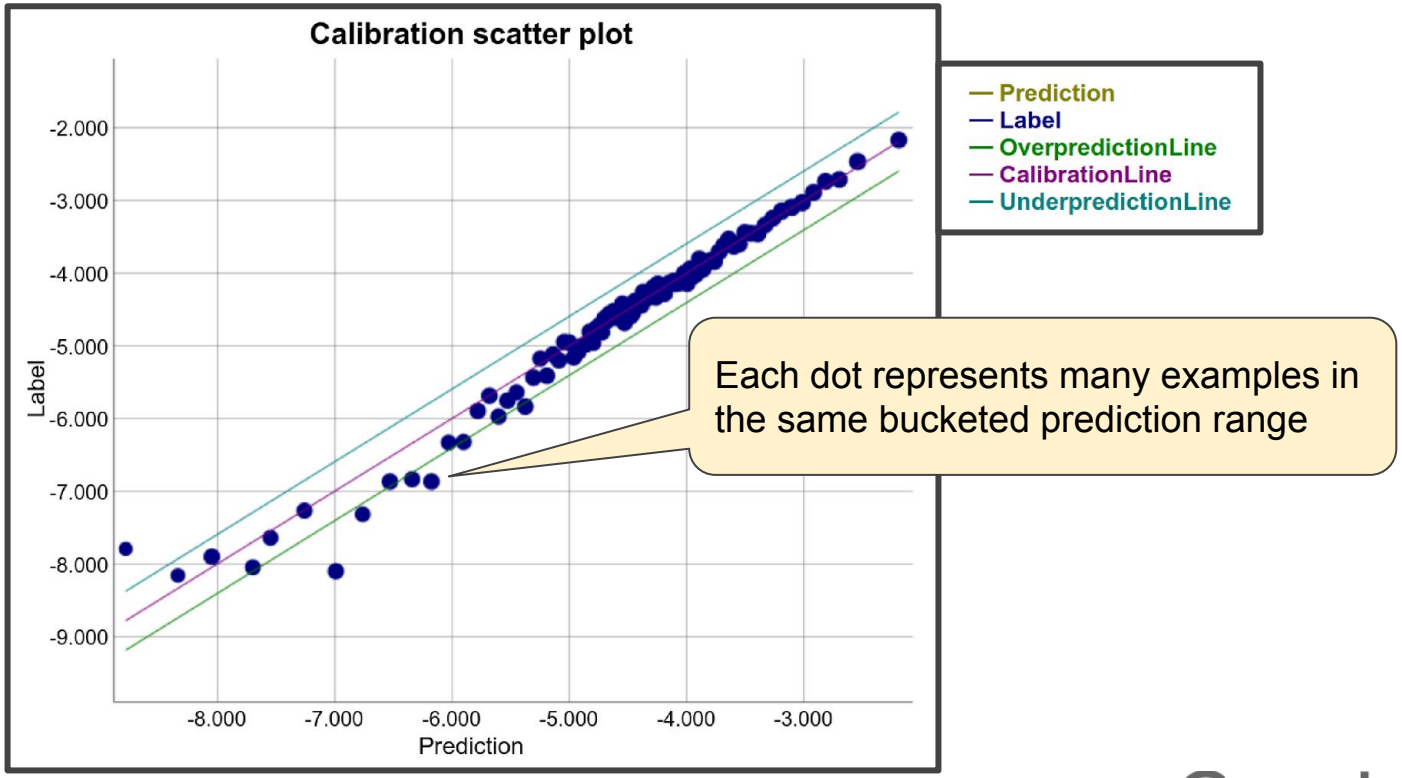
Two strategies are especially useful:

- L_2 **regularization** -- penalizes huge weights.
- **Early stopping** -- limiting training steps or learning rate.

Prediction Bias

- Logistic Regression predictions should be unbiased meaning that: average of predictions ≅ average of observations
- Bias is a sign that something is wrong.
  - Zero bias alone does not mean everything is working.
  - But it's a great sanity check.
- If you have bias, you have a problem (e.g. incomplete feature set, biased training sample, …)
- Don't fix bias with a calibration layer, fix it in the model.
- Look for bias in slices of data, this can guide improvements.

Calibration Plots show Bucketed Bias



Google

Evaluation Metrics for Linear Classification

Google

Classification via Thresholding

- Sometimes, we use the output from logistic regression to predict the probability an event occurs.
- Another common use for logistic regression is to use it for binary classification by introducing a threshold.

- Choice of threshold is a very important choice, and can be tuned.

Evaluation Metrics: Accuracy

- How do we evaluate classification models?
- One possible measure: Accuracy
  - the fraction of predictions we got right

Group Chat: Accuracy

- Devise a scenario in which accuracy would be a *misleading* metric of model quality.

Accuracy Can Be Misleading

- In many cases, accuracy is a poor or misleading metric. Two common situations that case this are:
  - Class imbalance, when positives or negatives are extremely rare
  - Different kinds of mistakes have different costs

Confusion Matrix

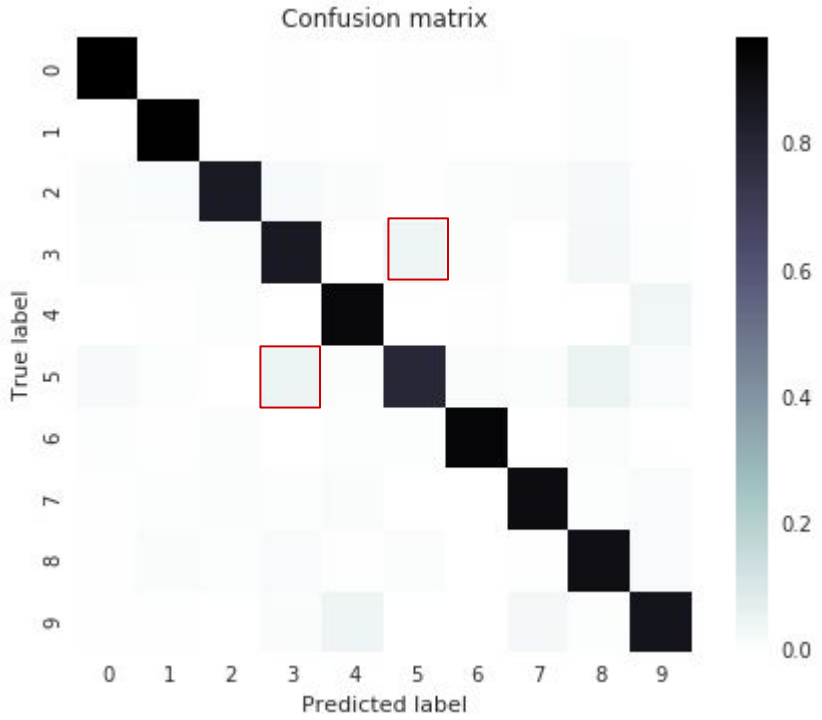
Useful to separate out different kinds of errors. Let's use spam detection as an example where spam(1), and not spam(0)

This is called a confusion matrix.		ML System Says	
		Spam	Not Spam
Truth	Spam	True Positive TP	False Negative FN
	Not Spam	False Positive FP	True Negative TN

Google

Confusion Matrix for Multi-Class Problems

- The confusion matrix to the right is for the task of digit prediction
- The strong weights along the diagonal indicate this model is very good.
- Two regions outlined in red illustrate 3 and 5 being confused.



Google

Evaluation Metrics: Precision and Recall

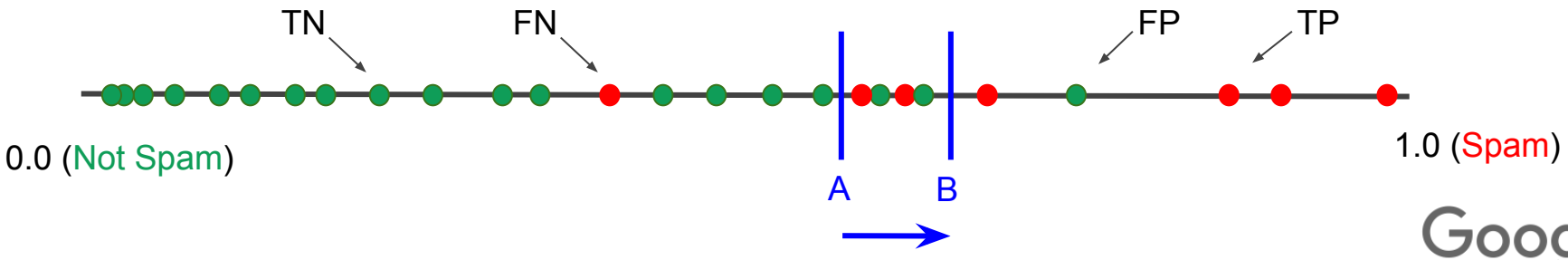
- We return to binary classification (True/False labels)
- Precision: (True Positives) / (All Positive Predictions)
  - When model said “positive” class, was it right?
  - Intuition: Did the model classify as “spam” too often?
- Recall: (True Positives) / (All Actual Positives)
  - Out of all the possible positives, how many did the model correctly identify?
  - Intuition: Did it classify as “not spam” too often?

Precision vs Recall Trade-offs

- A system with high precision might leave out some good email, but it is very unlikely to let spam through
- A system with high recall might let through some spam, but it also is very unlikely to miss any good email.
- The trick is to balance them -- and which kind of error is more problematic depends a lot on the application. What are some examples of this?

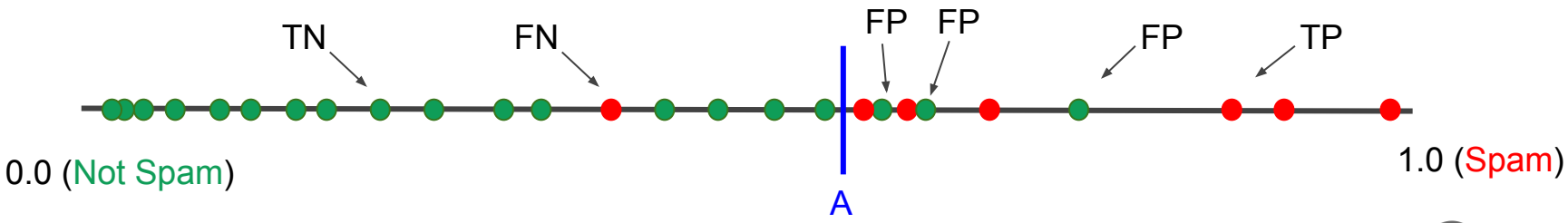
Selecting a Threshold

- True label shown by color
- Prediction from model shown as location on the line
- What happens with a larger decision threshold (B vs A)?
  - Prec = TP / (TP + FP)
  - Recall = TP / (TP + FN)



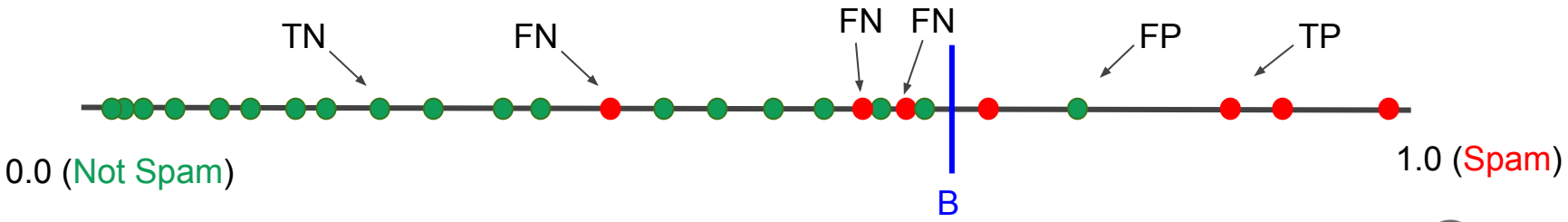
Selecting a Threshold

- True label shown by color
- Prediction from model shown as location on the line
- What happens with a larger decision threshold (B vs A)?
  - Prec = TP / (TP + FP)
  - Recall = TP / (TP + FN)



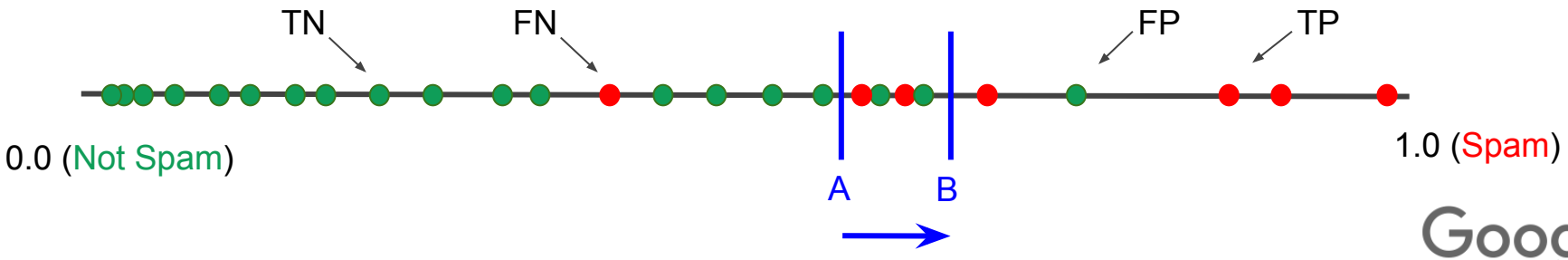
Selecting a Threshold

- True label shown by color
- Prediction from model shown as location on the line
- What happens with a larger decision threshold (B vs A)?
  - Prec = TP / (TP + FP)
  - Recall = TP / (TP + FN)

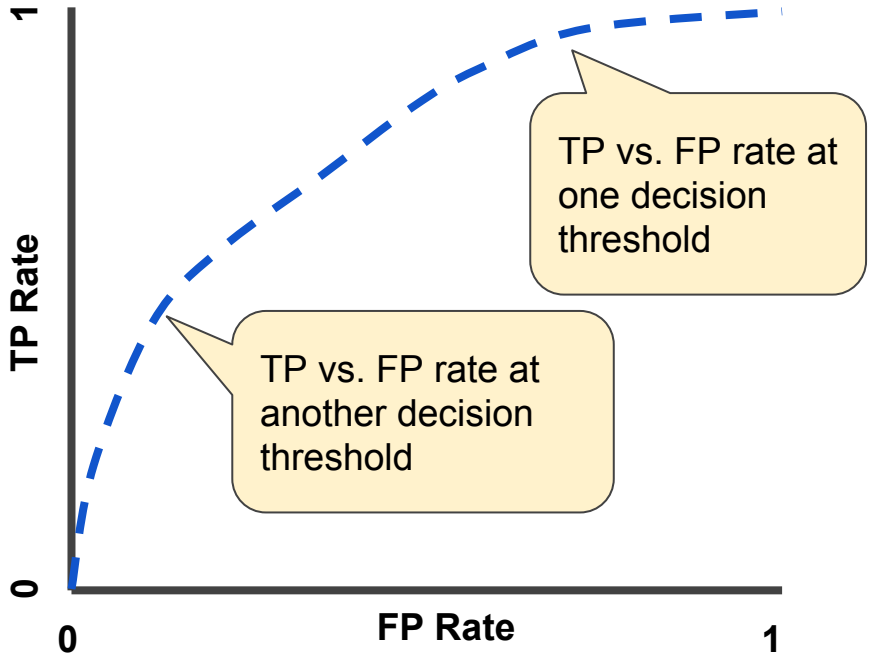


Selecting a Threshold

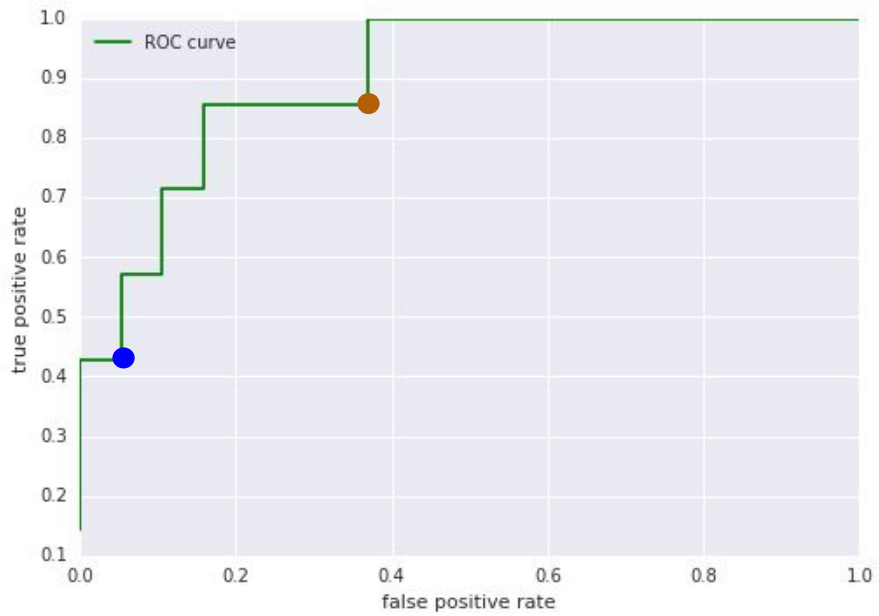
- True label shown by color
- Prediction from model shown as location on the line
- What happens with a larger decision threshold (B vs A)?
  - Prec = TP / (TP + FP) -- Increases since less FP
  - Recall = TP / (TP + FN) -- Decreases since more FN

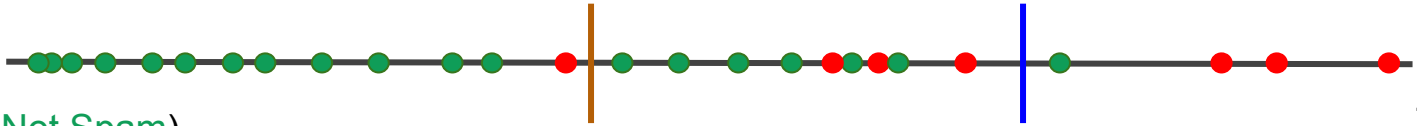


An ROC Curve



Sample ROC Curve





0.0 (Not Spam)

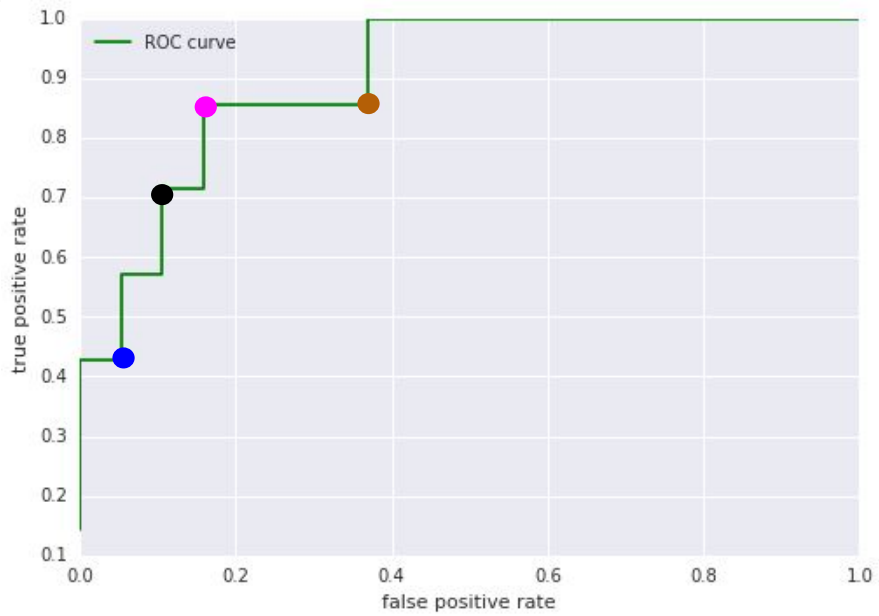
FPR = 7/19, TPR = 6/7

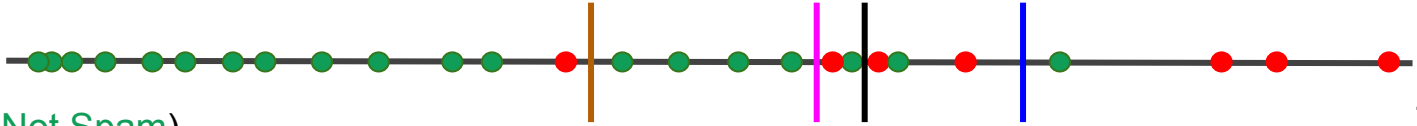
FPR = 1/19, TPR = 3/7

1.0 (Spam)

Google

Sample ROC Curve



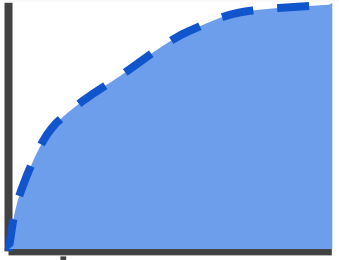


0.0 (Not Spam)

1.0 (Spam)

FPR = 2/19, TPR = 5/7

Evaluation Metrics: AUC



- AUC: “Area under the ROC Curve”
- Interpretation:
  - If we pick a random positive and a random negative, what’s the probability my model scores them in the correct relative order?
- Intuition: gives an aggregate measure of performance aggregated across all possible classification thresholds

Group Chat: AUC

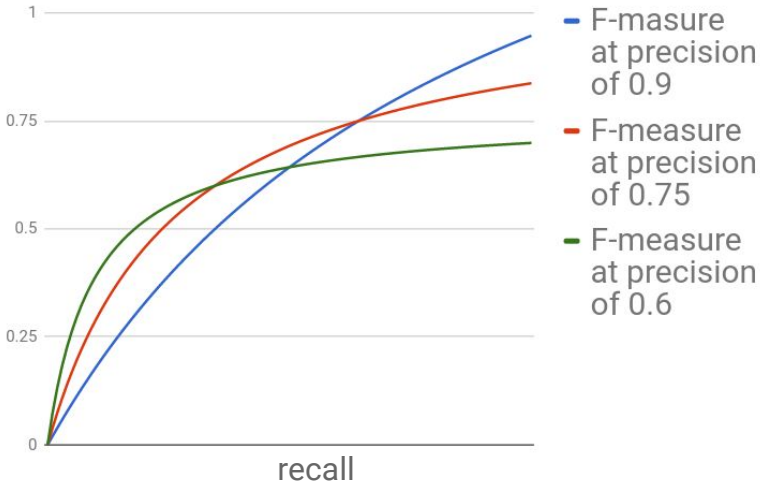
What will happen to the AUC if multiply each of the predictions (the *y'*) for a given model by 2.0?

F-measure

- Another commonly way to balance recall and precision is F-measure which is the harmonic mean of precision and recall (so ranges from 0 to 1).

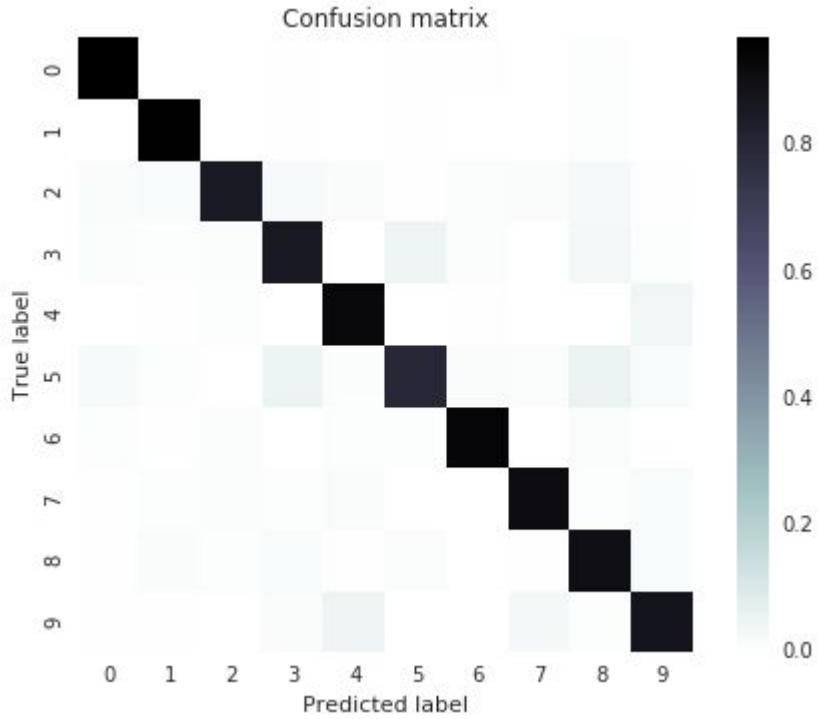
$$F = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

- Observe that if precision = recall = *x* then the F-measure is *x*. Just to give a feel here are a few sample values.



Google

Confusion Matrix for Multiclass Data



This confusion matrix provides a visualization for the task of digit recognition (so possible labels are 0, 1, …, 9). The bar on the right shows the mapping from color to the probability. The high values along the diagonal indicate a strong model.

Google

Regularization for Sparsity

Model Complexity

- **Remember Occam's Razor:** Increasing the model complexity increases the chances of overfitting.

- **Start simple** and build up the complexity only when needed. Your final solution might be sophisticated, but your first attempt shouldn't be.

Let’s Go Back to Feature Crosses

Caveat: Sparse feature crosses may significantly increase feature space

Possible issues:

- Model size may become huge
- Model requires more data and longer to train
- Overfitting becomes more of a problem

Reasons to Want Sparsity

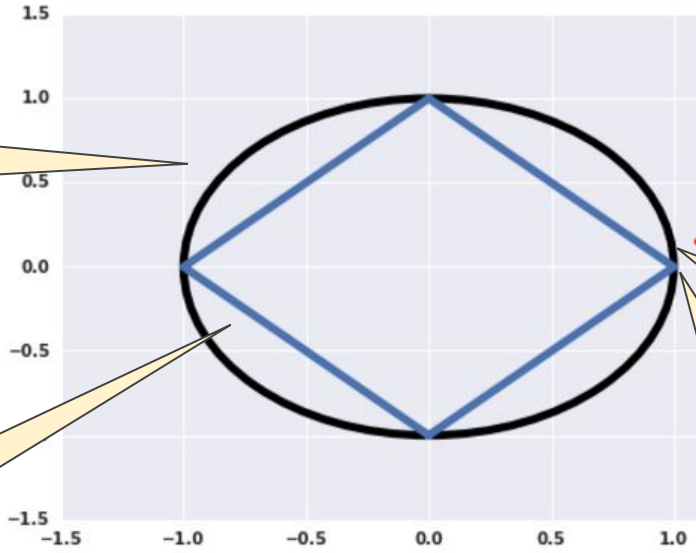
- Would be nice to encourage weights to exactly 0 where possible
  - Saves RAM, may reduce overfitting
  - Also those features can be removed from the model and no longer gathered and stored

L_1 Regularization

- L_0 regularization: penalize a weight for being non-zero
  - Computationally hard and too expensive to compute

- Relax to L_1 regularization:
  - Penalize sum of abs(weights)
  - Convex problem so computationally feasible
  - Encourages sparsity (unlike L_2)

Visualizing L_1 and L_2 Regularization in 2D



L2 ball looks like a circle

Here's a point we want to get back to the edge of a ball

Nearest point on L2 ball has two non-zero coordinates

L1 ball looks like a diamond

Nearest point on L1 ball is a corner: has a coordinate with value exactly 0.0

Google

Intro to Neural Networks

Google

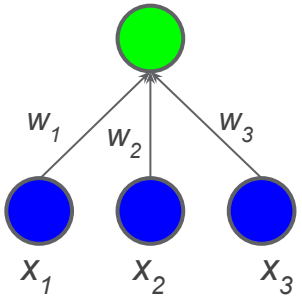
Other Ways to Learn Nonlinearities

- One alternative to feature crosses:
  - Structure the model so that features are used together as in feature crosses
  - Then combine those combinations to get even more intricate features.

- How can we choose these combinations?
  - Let the machine learning model discover them

A Linear Model

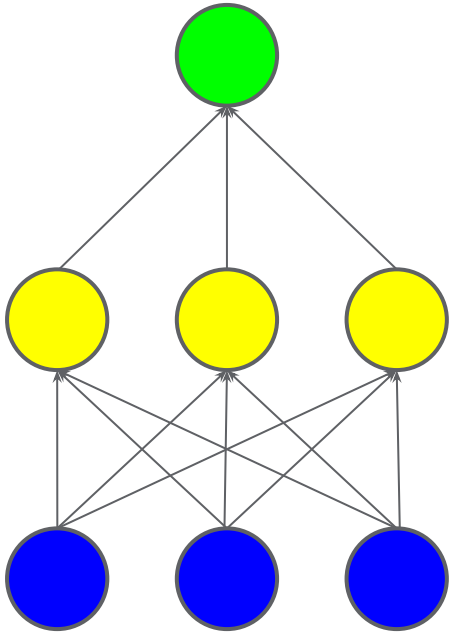
Output: $y' = w_1x_1 + w_2x_2 + w_3x_3 + b$



Input: $\boldsymbol{x} = (x_1, x_2, x_3)$

Note: the bias b is part of the model though not pictured

Adding Another Linear Layer

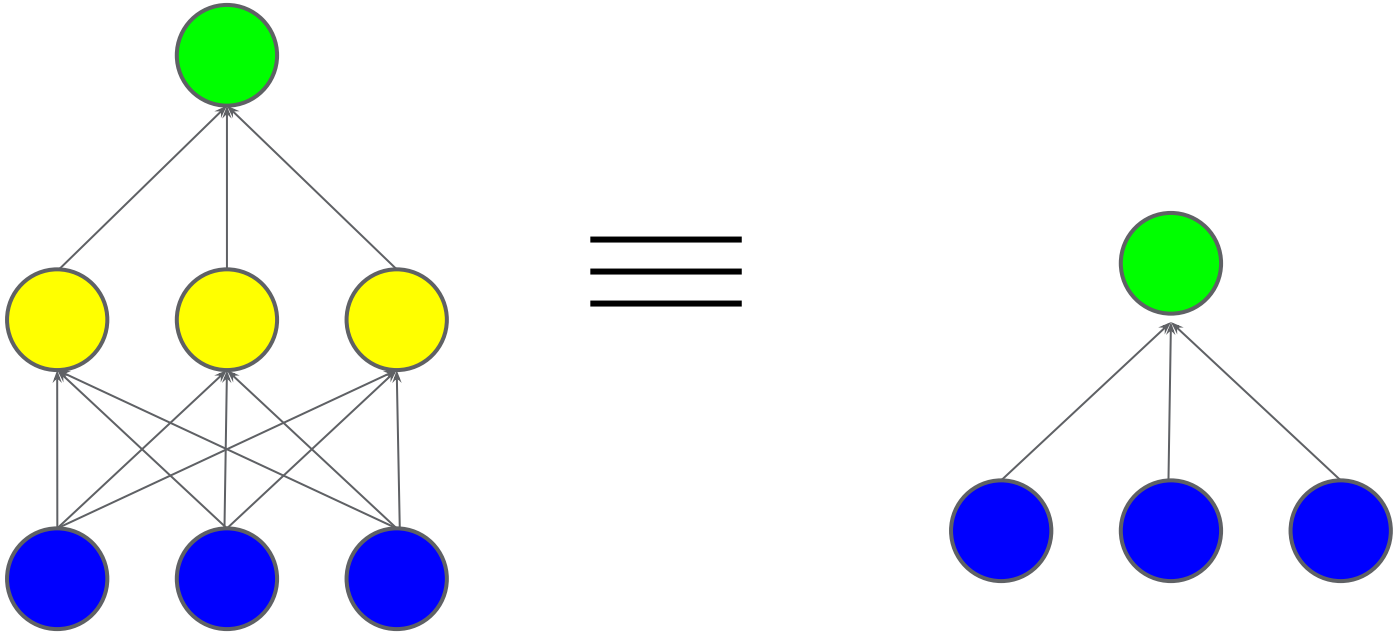


Output

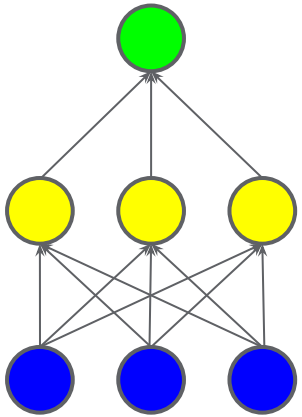
Hidden Layer

Input

Adding Another Linear Layer



Adding Another Linear Layer



Output

$$y' = h_1 * w_{4,1} + h_2 * w_{4,2} + h_3 * w_{4,3} + b_4$$

Hidden Layer

$$h_1 = x_1 * w_{1,1} + x_2 * w_{2,1} + x_3 * w_{3,1} + b_1$$
$$h_2 = x_1 * w_{1,2} + x_2 * w_{2,2} + x_3 * w_{3,2} + b_2$$
$$h_3 = x_1 * w_{1,3} + x_2 * w_{2,3} + x_3 * w_{3,3} + b_3$$

Input

$$y' = (x_1 * w_{1,1} + x_2 * w_{2,1} + x_3 * w_{3,1} + b_1) * w_{4,1} + (x_1 * w_{1,2} + x_2 * w_{2,2} + x_3 * w_{3,2} + b_2) * w_{4,2} + (x_1 * w_{1,3} + x_2 * w_{2,3} + x_3 * w_{3,3} + b_3) * w_{4,3} + b_4$$

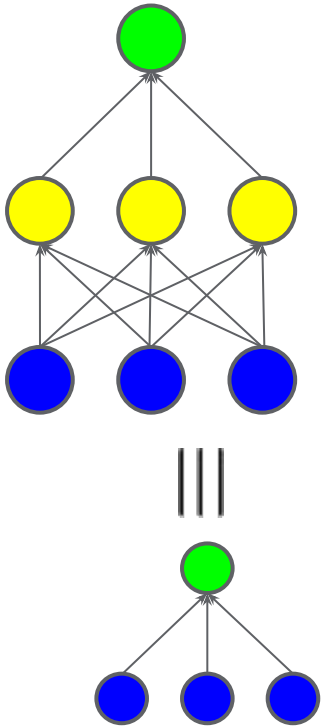
$$= x_1 (w_{1,1} * w_{4,1} + w_{1,2} * w_{4,2} + w_{1,3} * w_{4,3}) + x_2 (w_{2,1} * w_{4,1} + w_{2,2} * w_{4,2} + w_{2,3} * w_{4,3}) + x_3 (w_{3,1} * w_{4,1} + w_{3,2} * w_{4,2} + w_{3,3} * w_{4,3}) + b_1 * w_1 + b_2 * w_2 + b_3 * w_3 + b_4$$

w_1 w_2 w_3 b

$$y' = x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + b$$

Google

Adding Another Linear Layer



Output

Hidden Layer

Input

$$y' = h_1 * w_{4,1} + h_2 * w_{4,2} + h_3 * w_{4,3} + b_4$$
$$h_1 = x_1 * w_{1,1} + x_2 * w_{2,1} + x_3 * w_{3,1} + b_1$$
$$h_2 = x_1 * w_{1,2} + x_2 * w_{2,2} + x_3 * w_{3,2} + b_2$$
$$h_3 = x_1 * w_{1,3} + x_2 * w_{2,3} + x_3 * w_{3,3} + b_3$$

$$\begin{aligned} y' = {} & (x_1 * w_{1,1} + x_2 * w_{2,1} + x_3 * w_{3,1} + b_1) * w_{4,1} + \\ & (x_1 * w_{1,2} + x_2 * w_{2,2} + x_3 * w_{3,2} + b_2) * w_{4,2} + \\ & (x_1 * w_{1,3} + x_2 * w_{2,3} + x_3 * w_{3,3} + b_3) * w_{4,3} + \\ & b_4 \end{aligned}$$

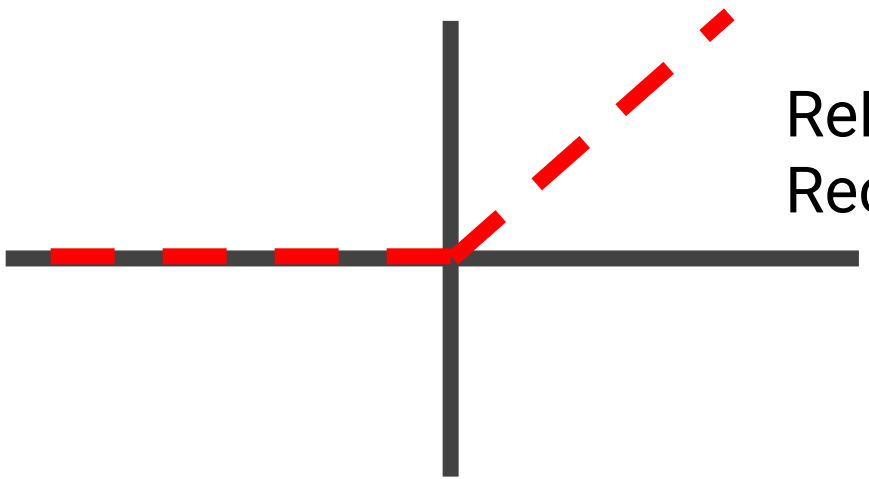
$$\begin{aligned} = {} & x_1 (w_{1,1} * w_{4,1} + w_{1,2} * w_{4,2} + w_{1,3} * w_{4,3}) + \\ & x_2 (w_{2,1} * w_{4,1} + w_{2,2} * w_{4,2} + w_{2,3} * w_{4,3}) + \\ & x_3 (w_{3,1} * w_{4,1} + w_{3,2} * w_{4,2} + w_{3,3} * w_{4,3}) + \\ & b_1 * w_1 + b_2 * w_2 + b_3 * w_3 + b_4 \end{aligned}$$

w_1 w_2 w_3 b

$$y' = x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + b$$

Google

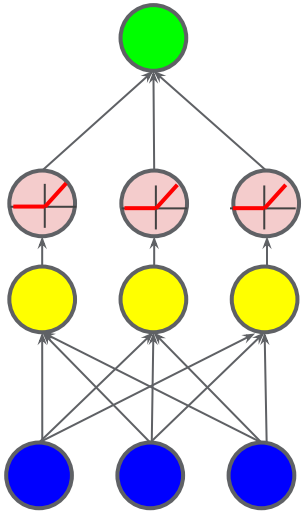
ReLu: A Simple Non-Linear Function



ReLu
Rectified Linear Unit

$$F(x) = \max(0, x)$$

Adding a Non-Linearity



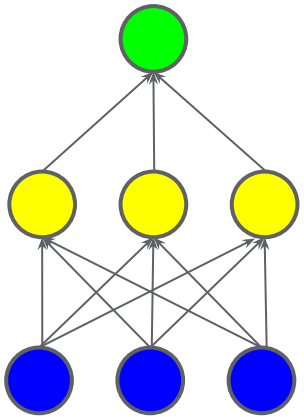
Output

Non-Linear Transformation Layer
(a.k.a. Activation Function)

Hidden Layer
(Linear)

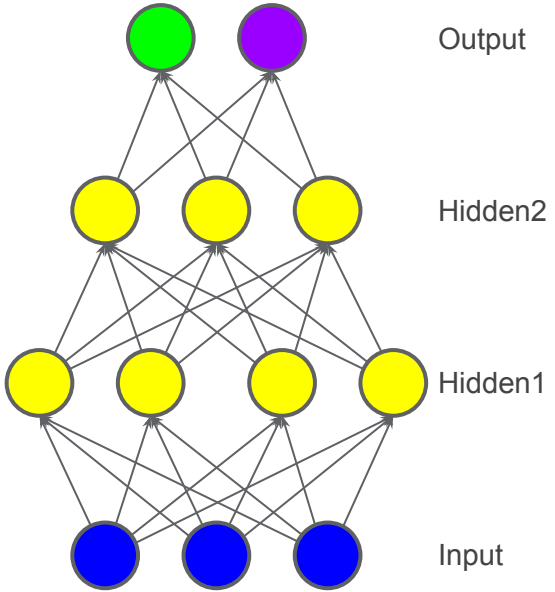
Input

By convention we combine the Activation function into the hidden layer making it a non-linear function. So the network to the left is draw as:



Google

Neural Nets Can Be Arbitrarily Complex



Training done via BackProp algorithm: gradient descent in very non-convex space. We will describe this soon.

Demo 1
Demo 2
Demo 3

Google

Training Neural Nets

Google

Intro to Backpropagation

Neural Networks are trained with an algorithm called backpropagation that generalizes gradient descent to deeper networks.

Backpropagation algorithm visual explanation

Backprop: What You Need To Know

- Gradients are important
  - If it’s differentiable, we can probably learn on it
- Gradients can vanish
  - Each additional layer can successively reduce signal vs. noise
  - ReLu’s are useful here
- Gradients can explode
  - Learning rates are important here
  - Batch normalization (useful knob) can help
- ReLu layers can die
  - Lower your learning rates

Review: Normalizing Feature Values

- We'd like our features to have reasonable scales
  - Roughly zero-centered, [-1, 1] range often works well
  - Helps gradient descent converge; avoid NaN trap
  - Avoiding outlier values can also help
- Can use a few standard methods:
  - Linear scaling
  - Hard cap (clipping) to max, min
  - Log scaling

Dropout Regularization

- Dropout: Another form of regularization, useful for Neural Networks
- Works by randomly “dropping out” unit activations in a network for a single gradient step
- The more you drop out, the stronger the regularization
  - 0.0 = no dropout regularization
  - 1.0 = drop everything out! learns nothing
  - Intermediate values more useful

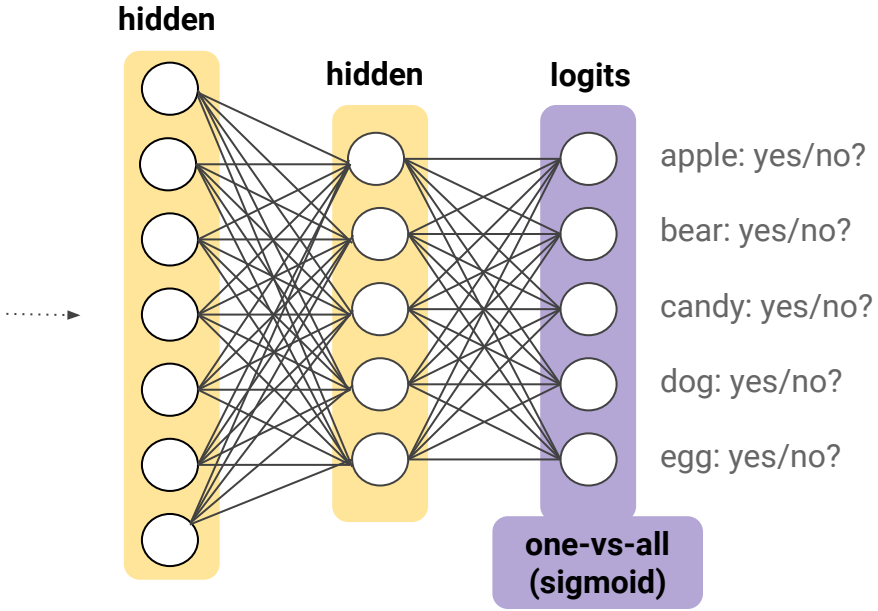
Multi-Class Neural Nets

More than two classes?

- Logistic regression gives useful probabilities for binary-class problems.
  - spam / not-spam
  - click / not-click
- What about multi-class problems?
  - animal, vegetable, mineral
  - red, orange, yellow, green, blue, indigo, violet
  - apple, banana, car, cardiologist, ..., walk sign, zebra, zoo

One-Vs-All Multi-Class

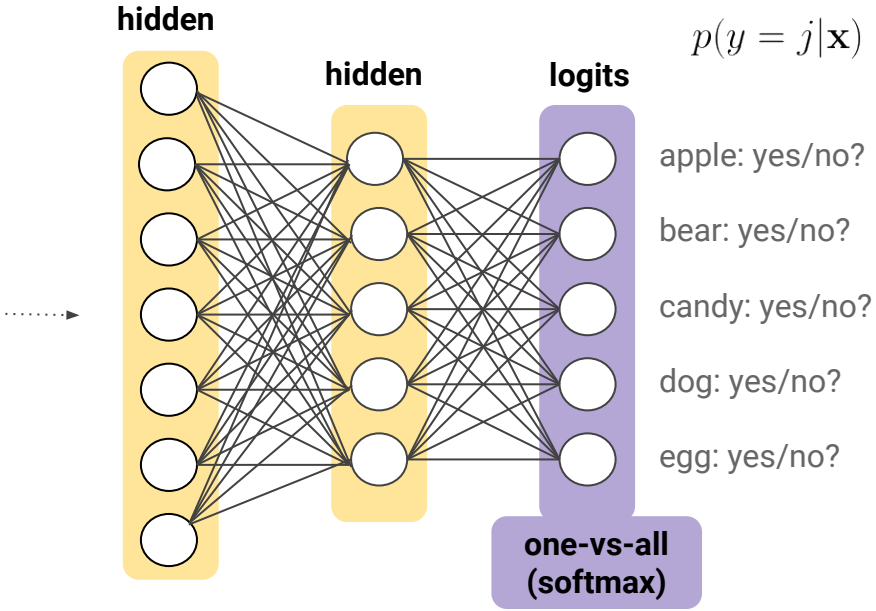
- Create a unique output for each possible class
- Train that on a signal of “my class” vs “all other classes”
- Typically optimize logistic loss



Google

SoftMax Multi-Class

- Add an additional constraint that all nodes to sum to 1.0
  - Allows outputs to be interpreted as probabilities
- Typically optimize cross-entropy loss (measure of similarity of distributions)



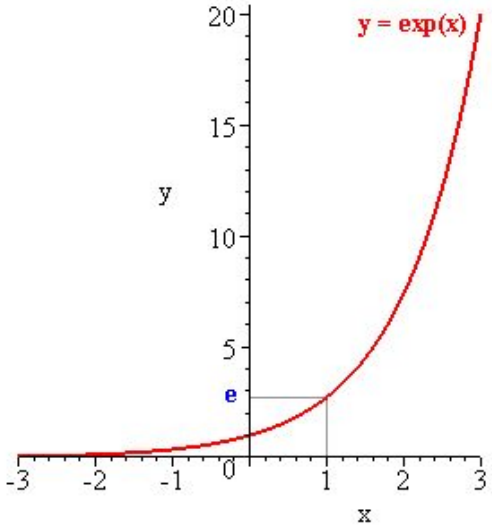
$$p(y=j|\mathbf{x}) = \frac{exp(\mathbf{w}_j^T\mathbf{x} + b_j)}{\sum_{k \in K} exp(\mathbf{w}_k^T\mathbf{x} + b_k)}$$

SoftMax Intuition

The prediction for label *j* is: $y'_j = \mathbf{w}_j^T \mathbf{x} + b_j$

Softmax definition where K is the set of possible labels:

$$p(y = j \mid \mathbf{x}) = \frac{\exp(y'_j)}{\sum_{k \in K} \exp(y'_k)}$$



For example, If we take an input of [1, 2, 3, 4, 1, 2, 3], the softmax is [0.024, 0.064, 0.175, 0.475, 0.024, 0.064, 0.175].

SoftMax Options

- **Full SoftMax**
  - Brute force --Calculates the denominator on all classes. This is only feasible when there are a relatively small number of classes.
- **Candidate Sampling**
  - Calculates the denominator for all the positive classes, but only for a random sample of the negatives. This can scale to millions of classes since the number of positive classes for any single example is generally small.

Single-Label vs. Multi-Label

Multi-Class, Single-Label Classification

- An example may be a member of only one class.
- Constraint that classes are mutually exclusive is important if you want to be able to treat the predictions as probabilities

Multi-Class, Multi-Label Classification:

- An example may be a member of more than one class.
- No additional constraints on class membership to exploit.
- One logistic regression loss for each possible class.

What to use, when?

One-vs-all is less computationally expensive than SoftMax, but at the expense of not having calibration across the labels.

Use One-vs.-All Multi-Class when:

- You want to view the output for each class as a prediction of the probability that the example belongs to the given class
- You do not need to rank the classes (and thus can reduce the computational cost by not calibrating across the classes).

Use SoftMax Multi-Class when:

- You want to rank all classes as to which are the best fit

Embeddings

Motivation From Collaborative Filtering

- **Input:** 1,000,000 users and which of 500,000 movies the user has chosen to watch
- **Task:** Recommend movies to users

To solve this problem some method is needed to determine which movies are similar to each other.

Organizing Movies by Similarity (1d)

Shrek

Incredibles

The Triplets of Belleville

Harry Potter

Star Wars

Bleu

The Dark Knight Rises

Memento

Google

Organizing Movies by Similarity (2d)

Shrek

Incredibles

Harry Potter

Star Wars

The Dark Knight Rises

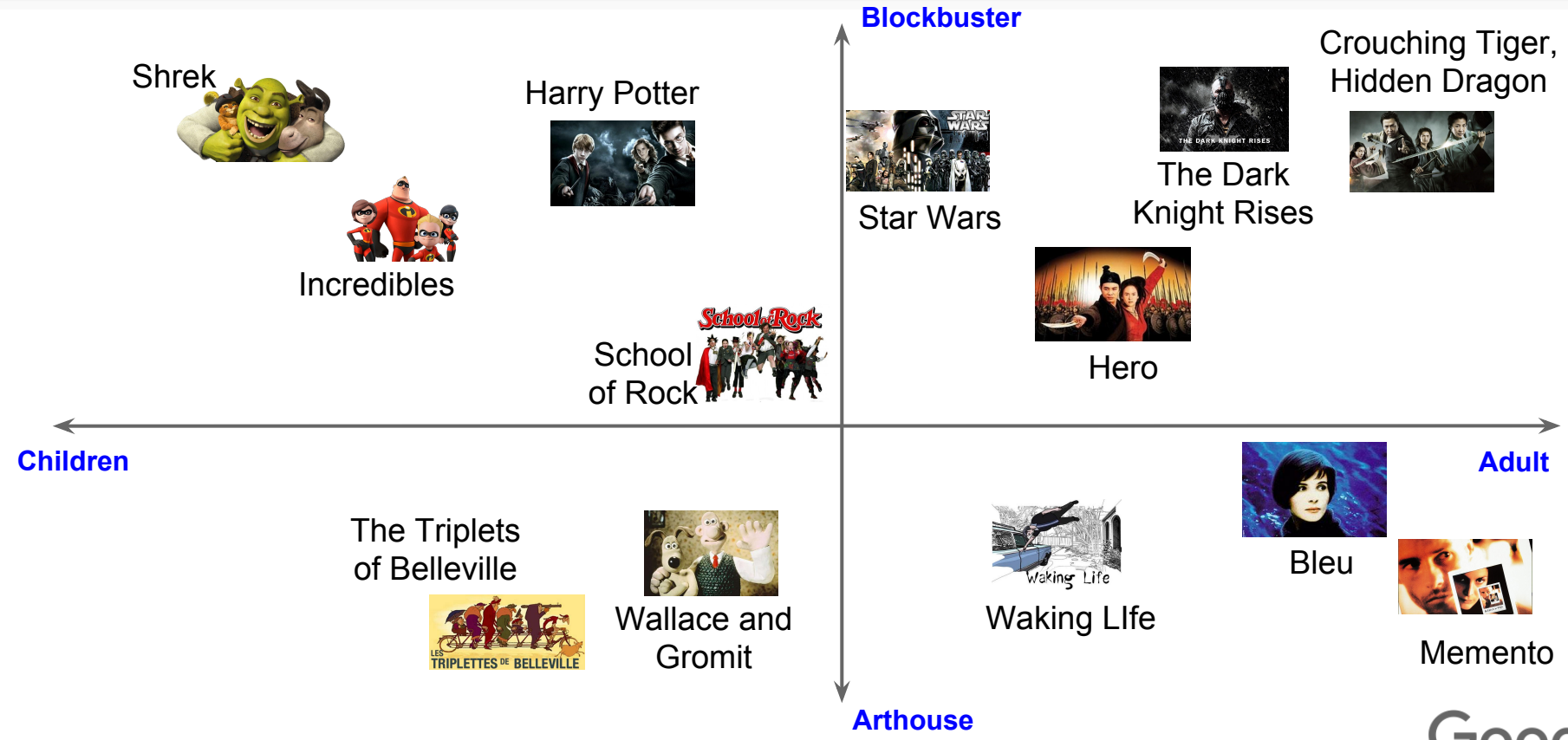
The Triplets of Belleville

Bleu

Memento

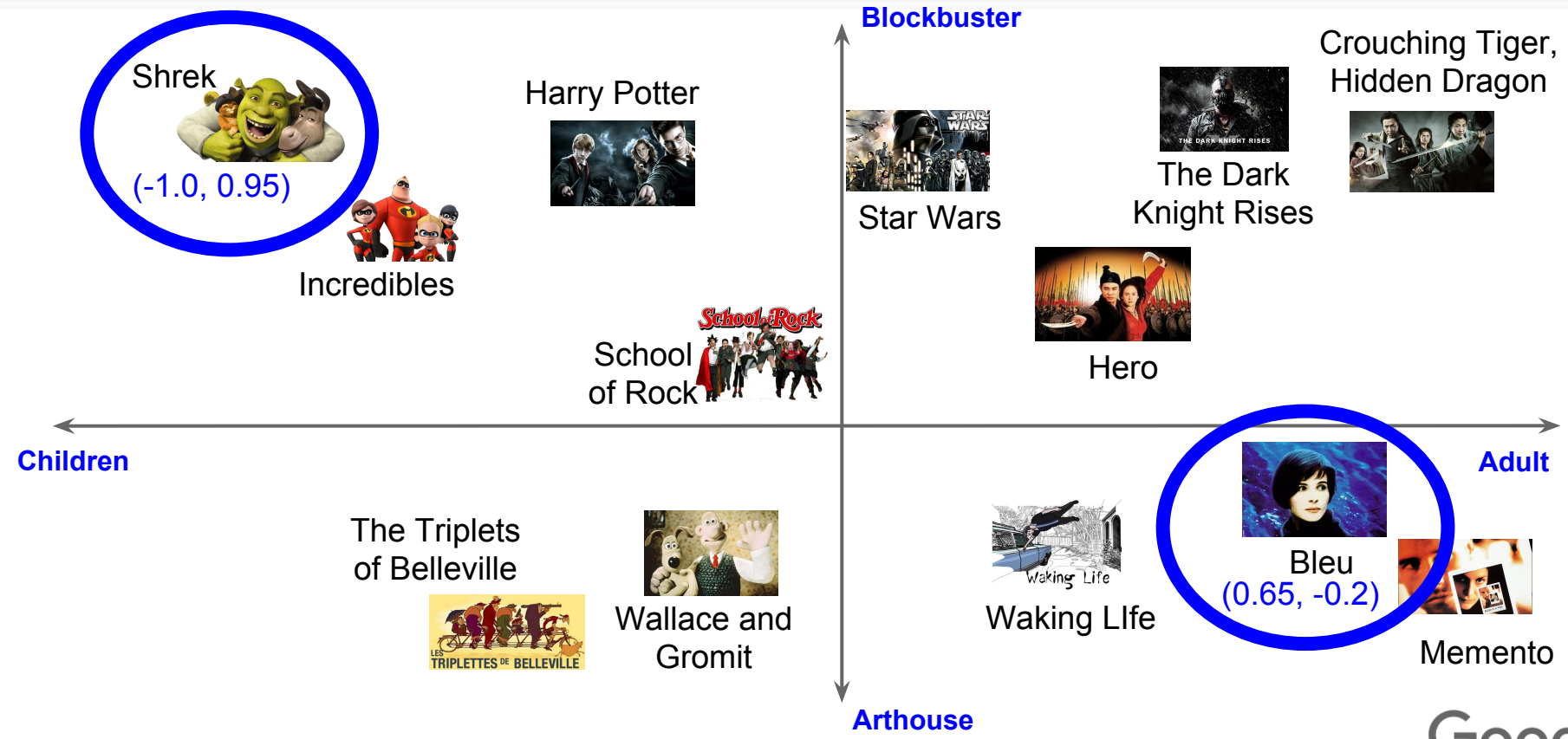
Google

Two-Dimensional Embedding



Google

Two-Dimensional Embedding



Google

d-Dimensional Embeddings

- Assumes user interest in movies can be roughly explained by *d* aspects
- Each movie becomes a *d*-dimensional point where the value in dimension *d* represents how much the movie fits that aspect
- Embeddings can be learned from data

Learning Embeddings in a Deep Network

- No separate training process needed -- the embedding layer is just a hidden layer with one unit per dimension
- Supervised information (e.g. users watched the same two movies) tailors the learned embeddings for the desired task
- Intuitively the hidden units discover how to organize the items in the d-dimensional space in a way to best optimize the final objective

Input Representation

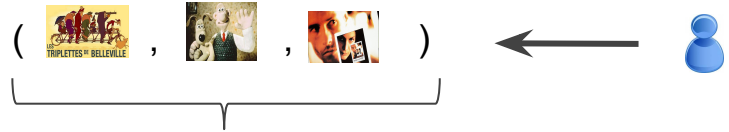
- Each example (a row in this matrix) is a sparse vector of features (movies) that have been watched by the user
- Dense representation of this example as:

(0, 1, 0, 1, 0, 0, 0, 1)

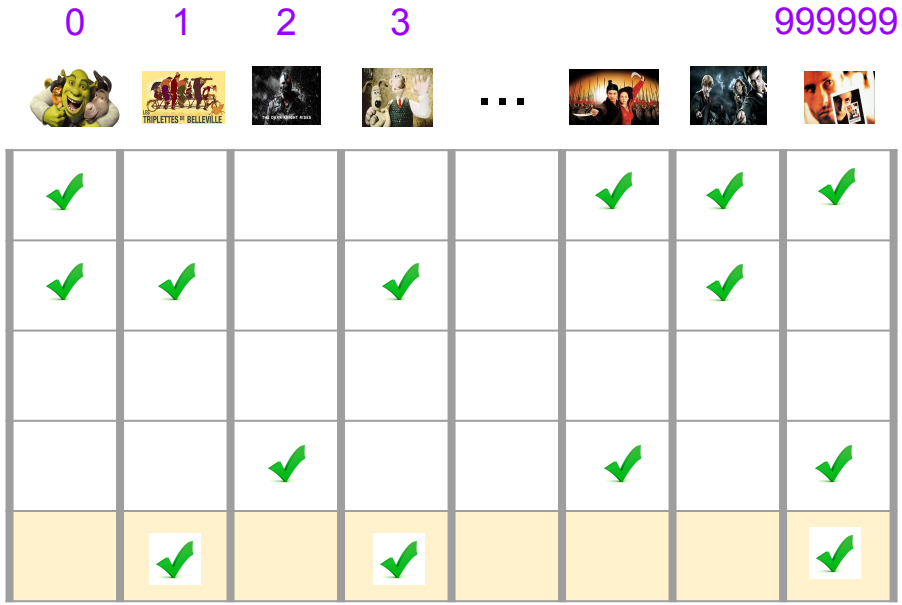
Is not efficient in terms of space and time

Input Representation

1. Build a dictionary mapping each feature to an integer from 0, …, # movies - 1
2. Efficiently represent the sparse vector as just the movies the user watched:

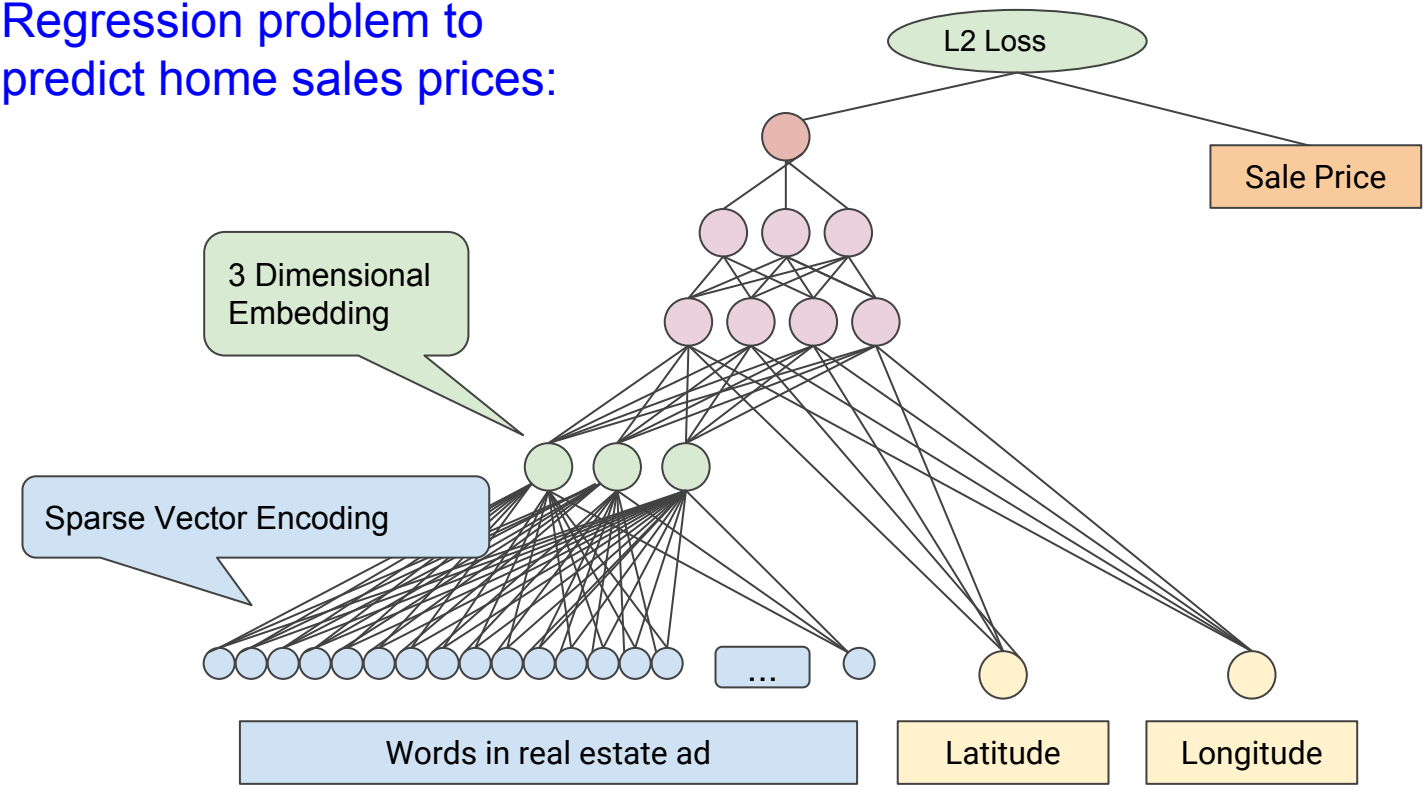


Represented as: (1, 3, 999999)



An Embedding Layer in a Deep Network

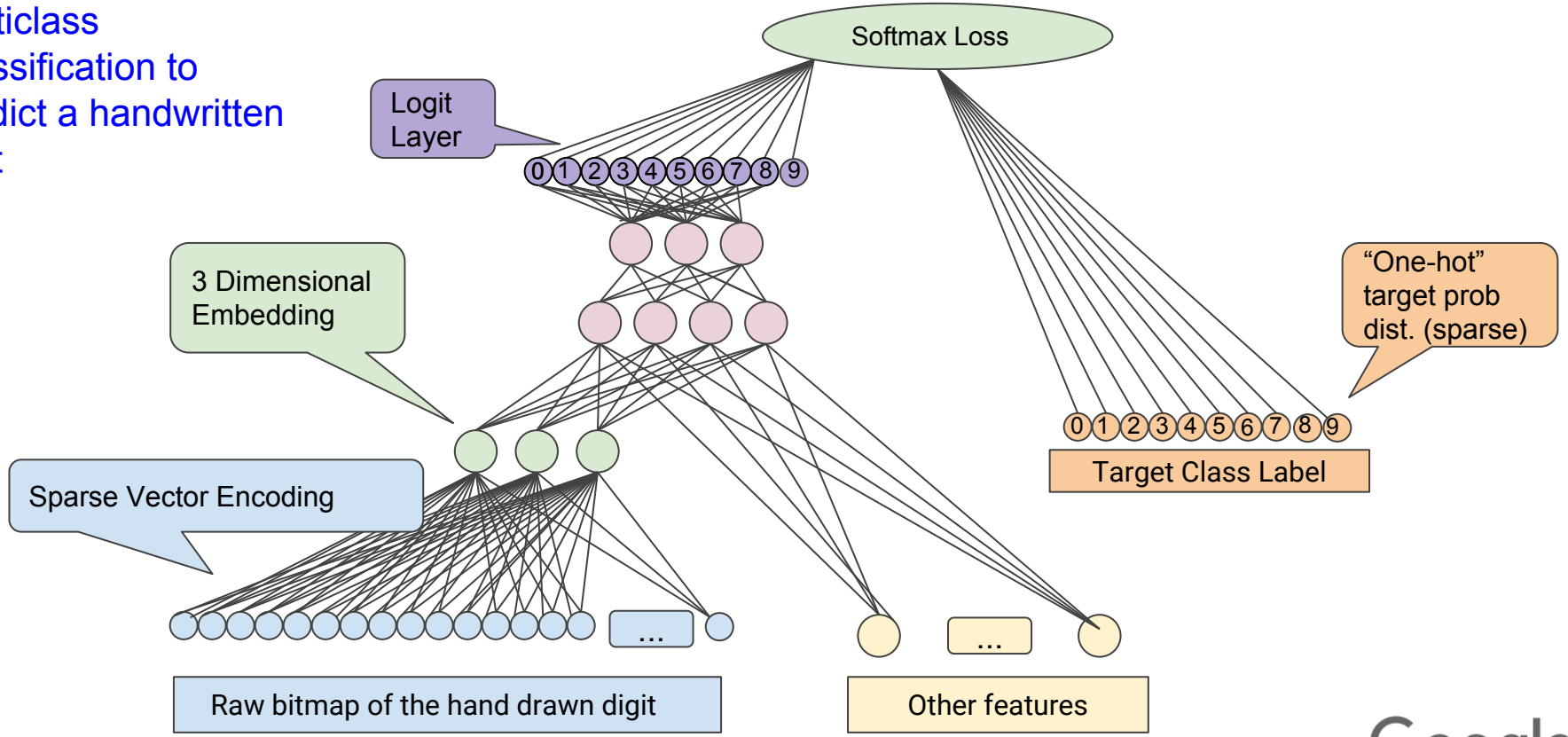
Regression problem to predict home sales prices:



Google

An Embedding Layer in a Deep Network

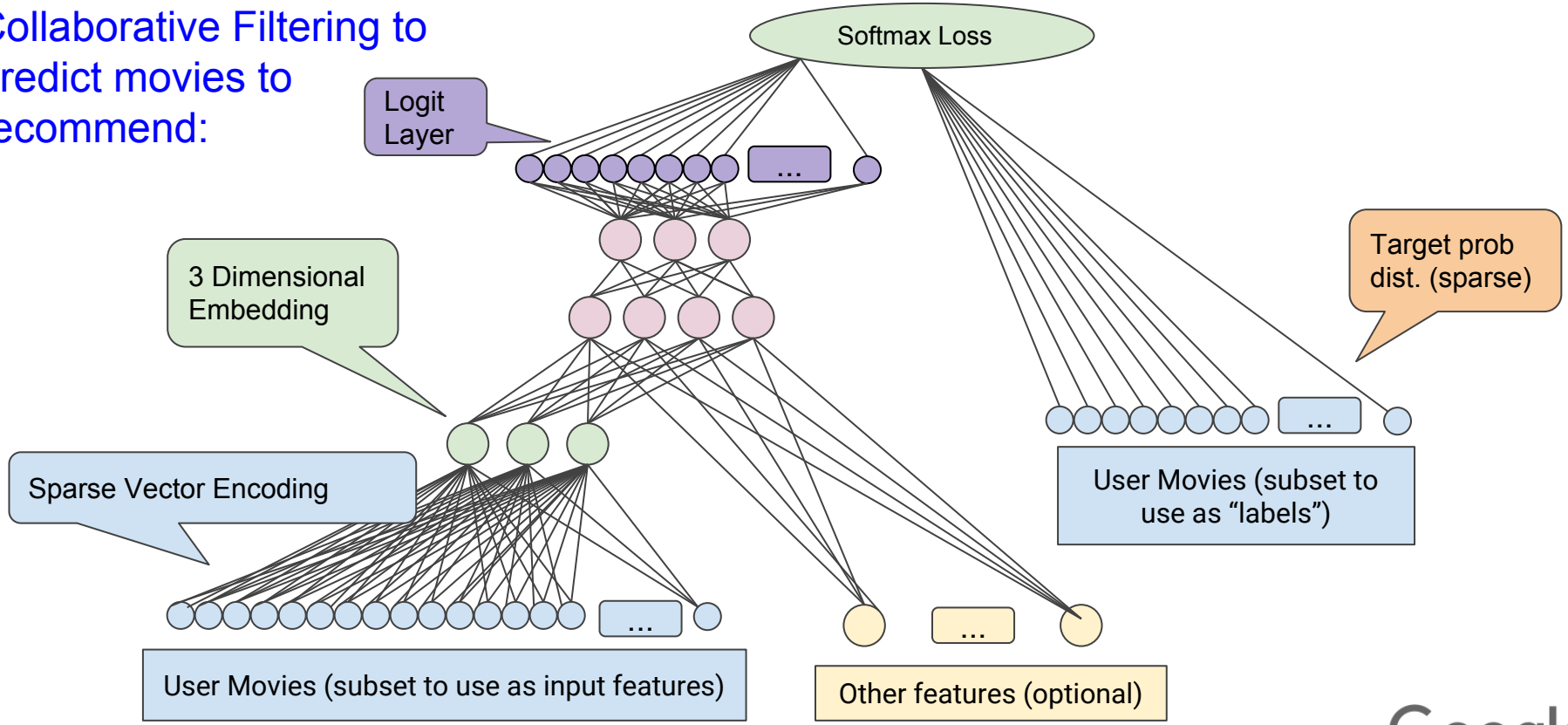
Multiclass Classification to predict a handwritten digit



Google

An Embedding Layer in a Deep Network

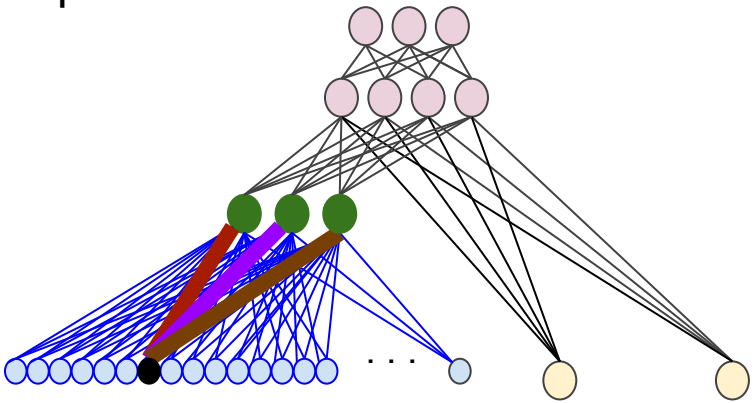
Collaborative Filtering to predict movies to recommend:



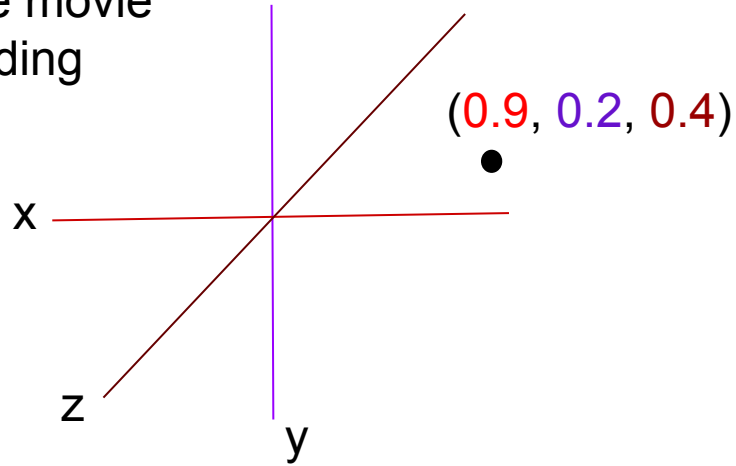
Google

Correspondence to Geometric View

Deep Network



Geometric view of a single movie embedding



- Each of hidden units corresponds to a dimension (latent feature)
- Edge weights between a movie and hidden layer are coordinate values

Google

Selecting How Many Embedding Dims

- Higher-dimensional embeddings can more accurately represent the relationships between input values
- But more dimensions increases the chance of overfitting and leads to slower training
- Empirical rule-of-thumb: $dimensions \approx \sqrt[4]{possible\ values}$
  - A good starting point but should be tuned using the validation data.

Embeddings as a Tool

- Embeddings map items (e.g. movies, text,...) to low-dimensional real vectors in a way that similar items are close to each other
- Embeddings can also be applied to dense data (e.g. audio) to create a meaningful similarity metric
- Jointly embedding diverse data types (e.g. text, images, audio, …) define a similarity between them

Unsupervised Learning: Clustering

Google

What is Clustering?

Given

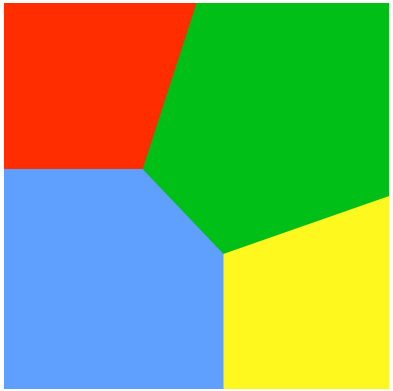
- a set of items
- a similarity metric on these items

Identify groups of most similar items

Some Applications of Clustering

- Grouping similar sets of items
- Data compression
- Generate features for ML systems
- Transfer Information learned from one setting to another (Transfer Learning)

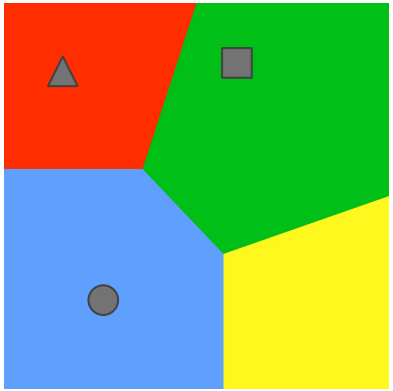
How “Hard” is Clustering?



Hard

Soft

How “Hard” is Clustering?



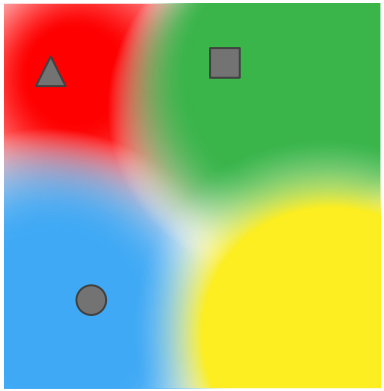
Clusters

Items				
●	0	0	1	0
▲	1	0	0	0
	.	.	.	.
■	0	1	0	0

Hard

Clusters

Items				
●	0.1	0	0.7	0.2
▲	0.8	0.1	0.1	0
	.	.	.	.
■	0.2	0.6	0.1	0.1



Soft

Is Clustering Supervised?

- **No**, if similarity is given (rarely the case)

- Otherwise ask "**Is similarity learned ?**"
  - *Unsupervised*: similarity is designed ad hoc from features
  - *Supervised*: similarity is derived from embeddings via a supervised Deep Neural Network

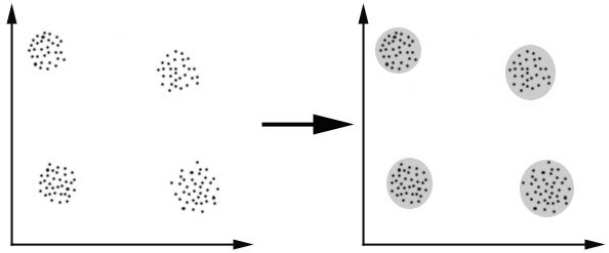
Understand Your Similarity Measure!

- Ad hoc similarity:
  - Handle heterogeneous features with care
    - Numerical data of different scales, different distributions (e.g. Home Price, Lot size, #rooms)

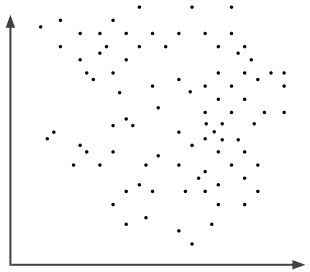
      💡 A good generic solution: use **Quantiles**
    - Categorical data of different cardinality (e.g. Zip code, Home type)
    - Missing data: ignore if rare, otherwise infer via ML

Clustering in Euclidean Space

- Designed with this in mind



- Applied to this



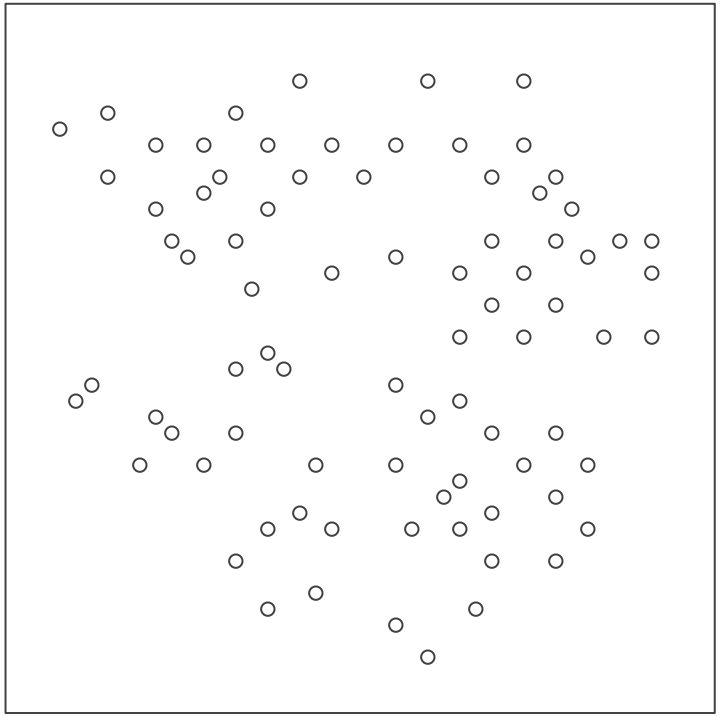
Quality Analysis of Clustering

- Clustering is unsupervised (given similarity):
  no ground truth to compare to

- Quality Analysis is done by
  - Eyeballing
  - When clustering is a piece of a larger system, measure it in the global system

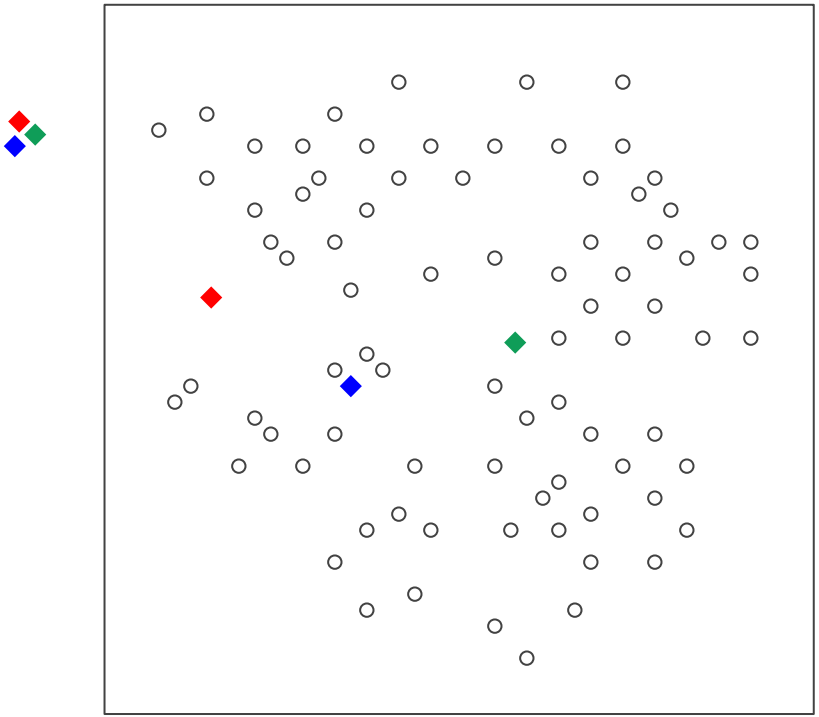
k-Means

- Simple
- Efficient
- Good basic clustering algorithm



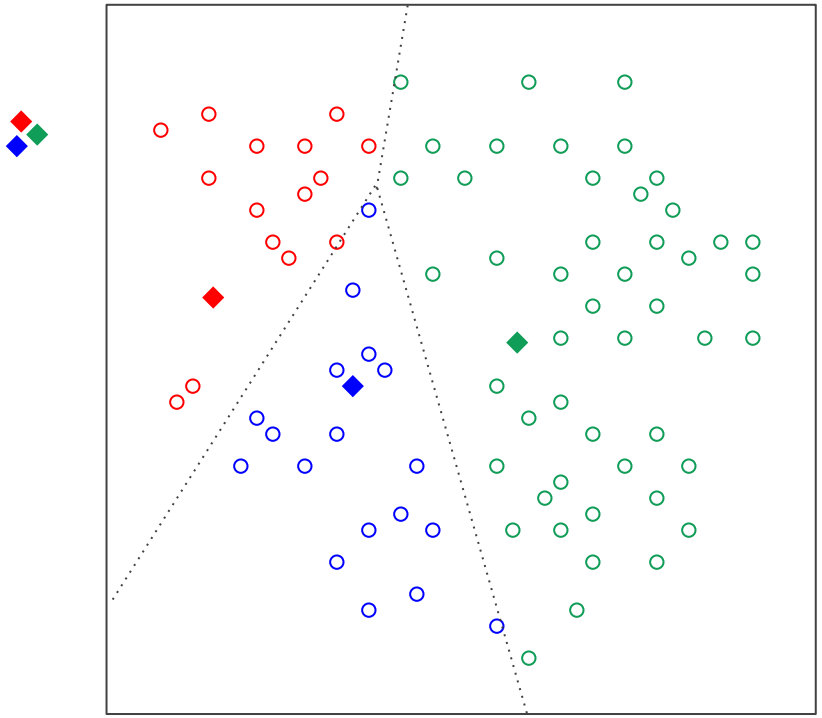
k-Means

- Initialization
  - Select k cluster centers randomly



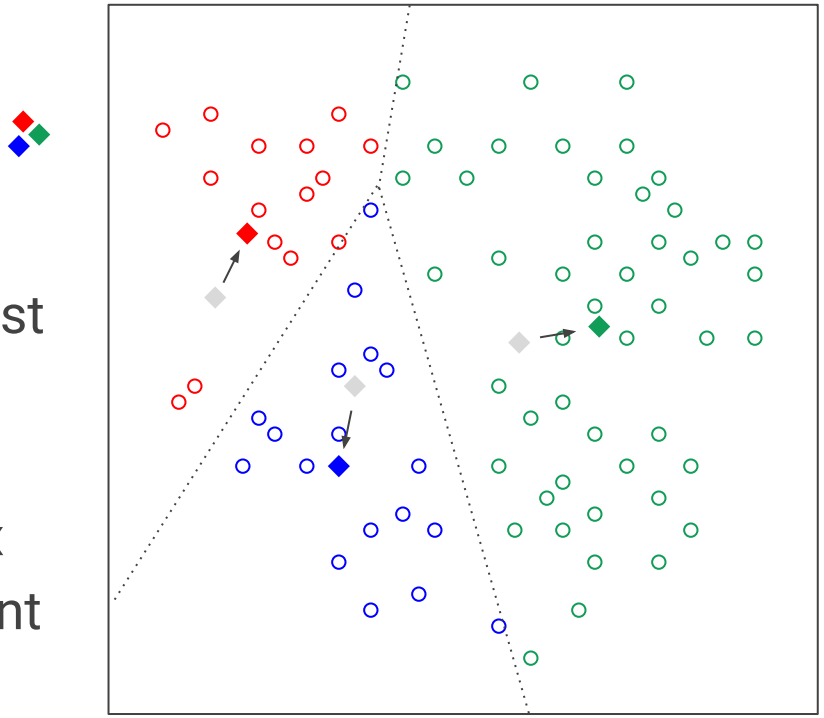
k-Means in action

- Initialization
  - Select k cluster centers randomly
- E-Step (Expectation)
  - Assign each point to the closest cluster



k-Means in action

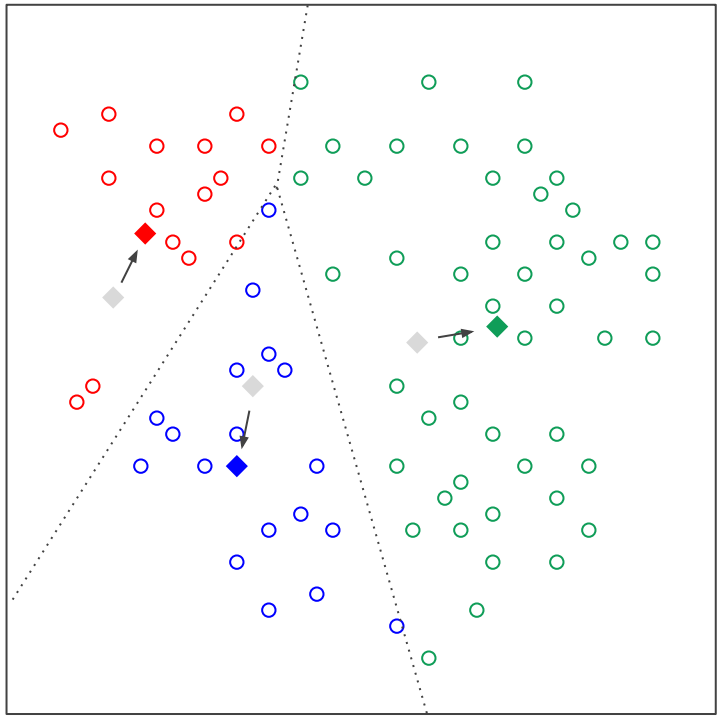
- Initialization
  - Select k cluster centers randomly
- E-Step (Expectation)
  - Assign each point to the closest cluster
- M-Step (Maximization)
  - Recompute centers as mean x and y coordinate among current assignment



k-Means in action

- Initialization
  - Select k cluster centers randomly
- E-Step (Expectation)
  - Assign each point to the closest cluster
- M-Step (Maximization)
  - Recompute centers as mean x and y coordinate among current assignment

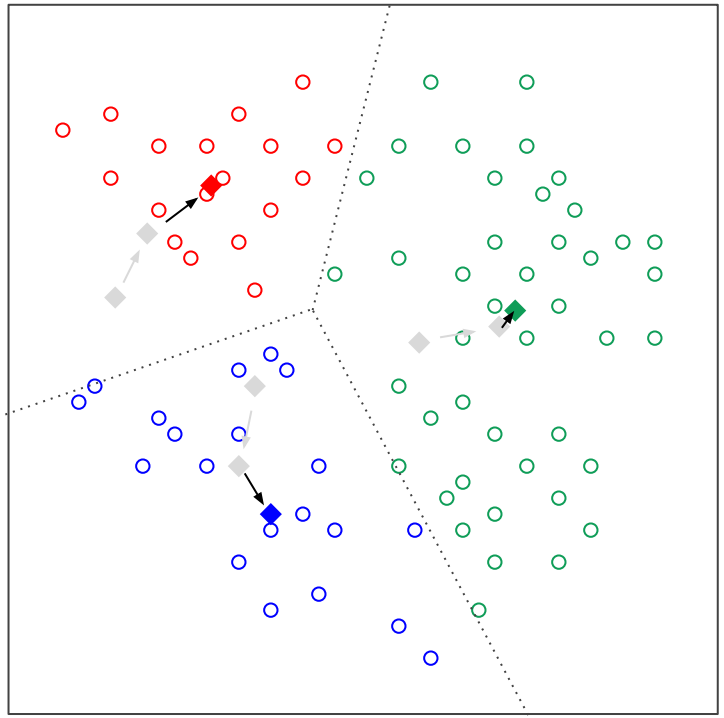
Repeat



k-Means in action

- Initialization
  - Select k cluster centers randomly
- E-Step (Expectation)
  - Assign each point to the closest cluster
- M-Step (Maximization)
  - Recompute centers

Repeat

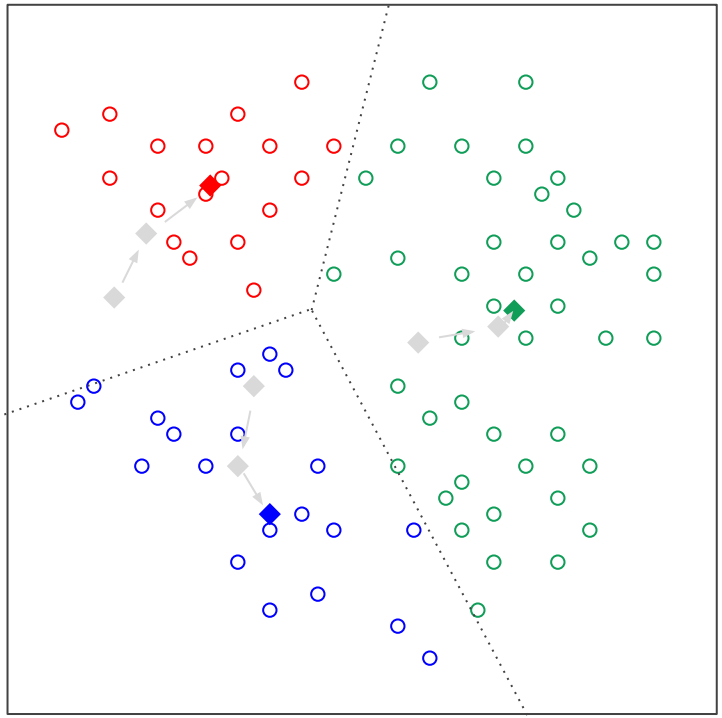


k-Means in action

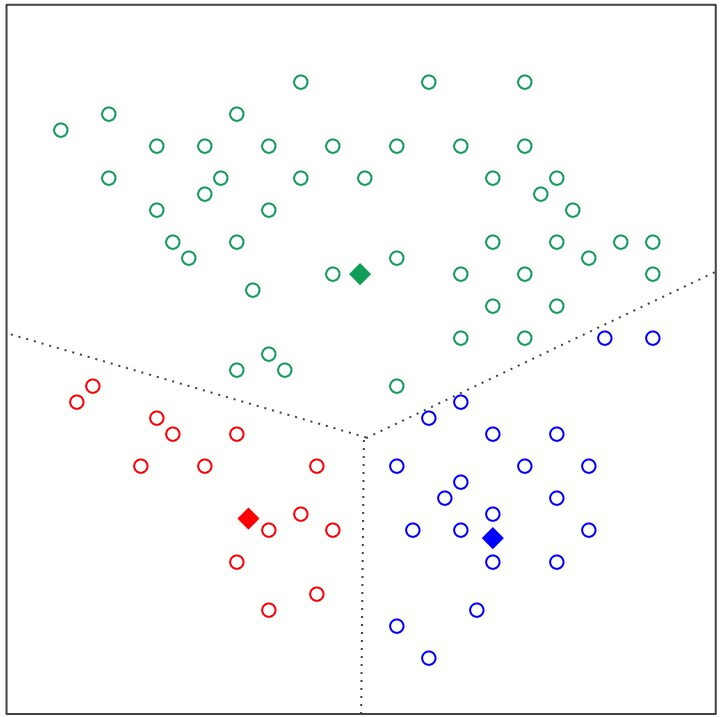
- Repeat until convergence
  - Always converges  when no item is jumping clusters in E-Step

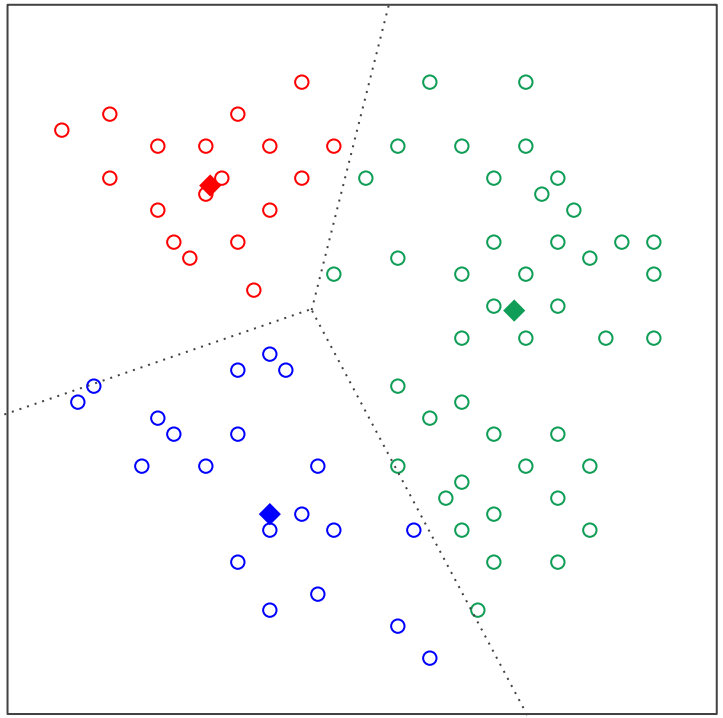
  Minimizes within cluster distance

- Depends on initialization



k-Means for Different Initializations

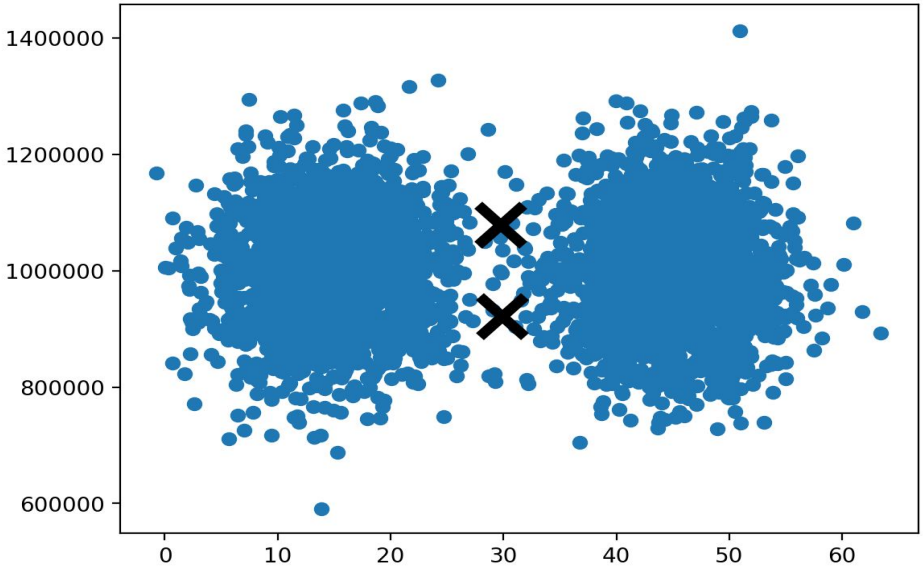


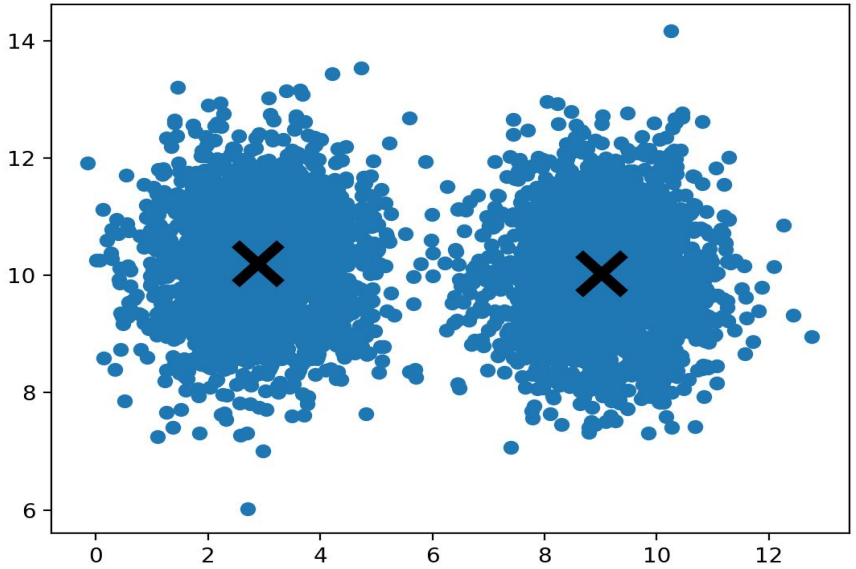


Google

Understand Your Data/Similarity Measure!

Why does k-means produce such drastically different results for visually similar data?





Google

k-Means Summary

- Cons
  - Need to determine k
  - Result depends on the initialization
  - “Curse of dimensionality” -- cost grows exponentially with the dimensionality of the data (points)
- Pros
  - Simple & efficient
  - Guaranteed convergence
  - Can be warm-started
  - Generalization: soft clusters, non-spherical clusters