

Machine Learning 3 - Generalization, K-means

True goal for machine learning



Question

What's the true objective of machine learning?

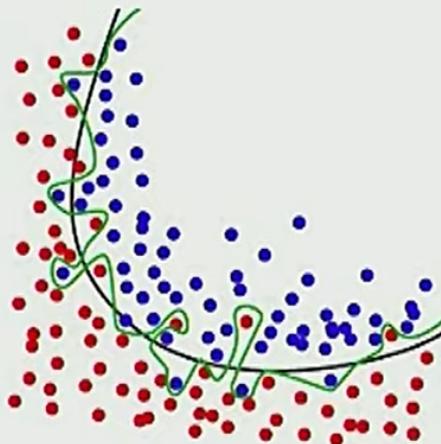
- minimize error on the training set
- minimize training error with regularization
- minimize error on the test set
- minimize error on unseen future examples
- learn about machines

[activate](#) [deactivate](#) [reset](#) [report](#)

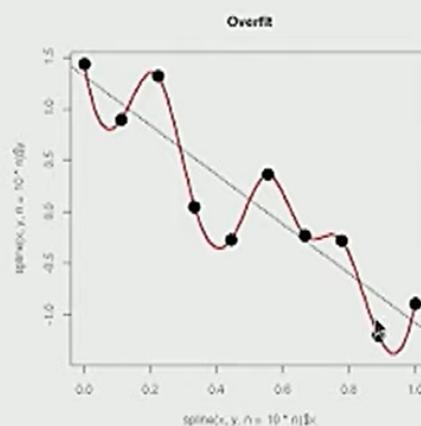
Is to minimize error on unseen future examples

Generalization

Overfitting pictures



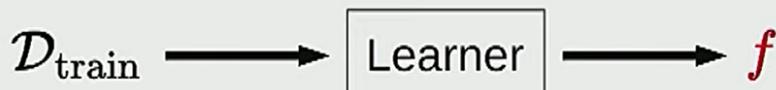
Classification



Regression

Evaluation & Prevent overfitting

Evaluation



How good is the predictor f ?



Key idea: the real learning objective

Our goal is to minimize **error on unseen future examples.**

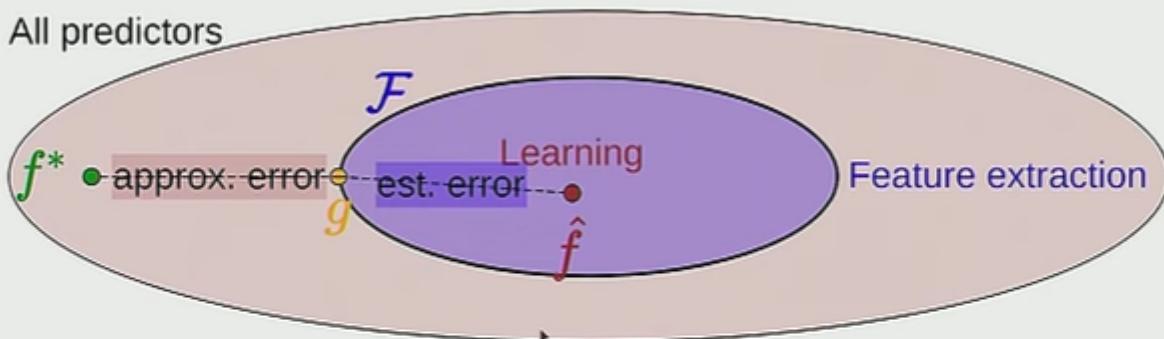
Don't have unseen examples; next best thing:



Definition: test set

Test set $\mathcal{D}_{\text{test}}$ contains examples not used for training.

Approximation and estimation error



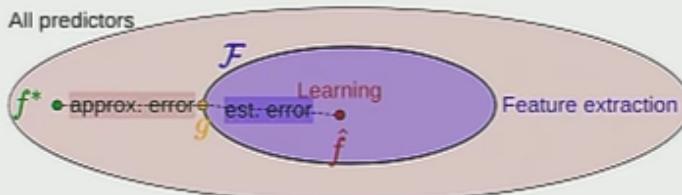
- Approximation error: how good is the hypothesis class?
- Estimation error: how good is the learned predictor **relative to** the potential of the hypothesis class?

$$\underbrace{\text{Err}(\hat{f}) - \text{Err}(g)}_{\text{estimation}} + \underbrace{\text{Err}(g) - \text{Err}(f^*)}_{\text{approximation}}$$

f^* : The true function/answer, the function that always gets the right answer g : The best predictor you can get in your hypothesis class (The functions you get can after feature engineering)

It is quite a trade-off

Effect of hypothesis class size



As the hypothesis class size increases...

Approximation error decreases because:

taking min over larger set

Estimation error increases because:

harder to estimate something more complex

How do we control the hypothesis class size?

How to control the size of Hypothesis class

Strategy 1: dimensionality

$$\mathbf{w} \in \mathbb{R}^d$$

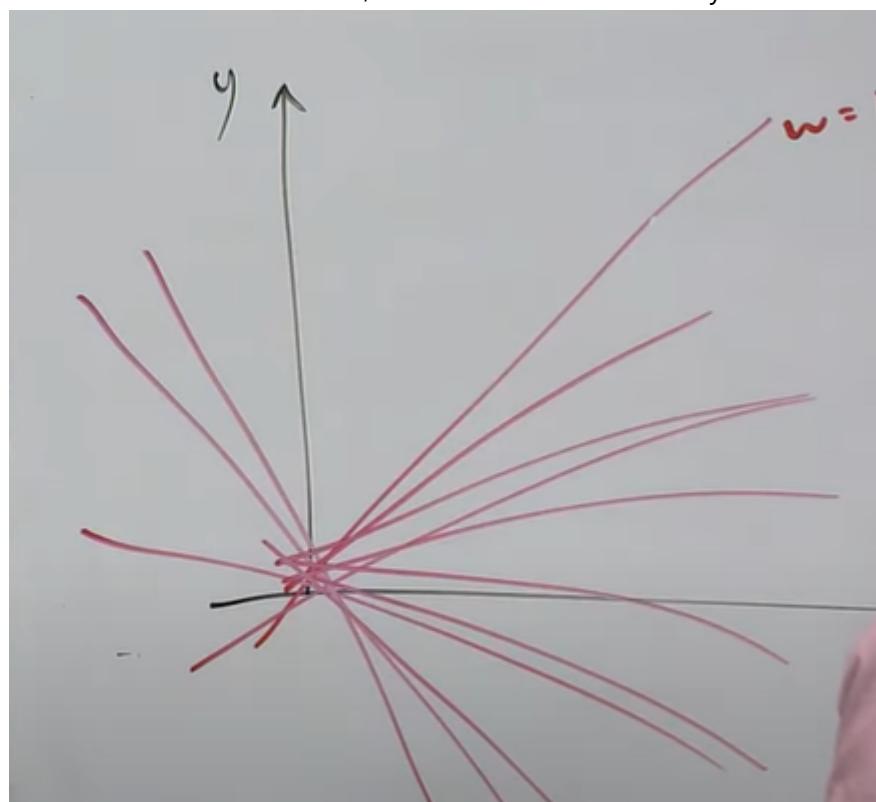
Reduce the dimensionality d :



Norm

Let say we have a linear regression $y = \mathbf{w}x$. The whole function space would be all linear lines that pass the origin.

If somehow we can define, let's say the \mathbf{w} is smaller or closer to 0, then we can eliminate many useless



function, and thus reduce the norm

This means shrink the total number of weight vectors that you are considering, as u putting more constraints, they will be smaller

Regularization

Controlling the norm

Controlling the norm

Regularized objective:

$$\min_{\mathbf{w}} \text{TrainLoss}(\mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$



Algorithm: gradient descent

Initialize $\mathbf{w} = [0, \dots, 0]$

For $t = 1, \dots, T$:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta (\nabla_{\mathbf{w}} [\text{TrainLoss}(\mathbf{w})] + \lambda \mathbf{w})$$

Same as gradient descent, except shrink the weights towards zero by λ .

You can add some **penalty** to your Training loss function

This is saying: Optimizer, you should try to make training loss small, but also try to make the second part(weight vectors) small as well

So this is saying that, fit the data but not at the costs of having huge weight vectors(Which may be overfitting)

It is important that these need to be the same w and you are optimizing the sum

How to do it

Gradient descent

Controlling the Norm - early stopping

Just stop early

Controlling the norm: early stopping



Algorithm: gradient descent

Initialize $\mathbf{w} = [0, \dots, 0]$

For $t = 1, \dots, T$:

$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$

Idea: simply make T smaller

Intuition: if have fewer updates, then $\|\mathbf{w}\|$ can't get too big.

Lesson: try to minimize the training error, but don't try too hard.

Because normally we initialize the weights from a baseline near 0, For neural nets, we might give it a random value near 0

IF u have a pre-given weights, then basically ur saying don't go too far from ur initialization

We start the norm/weight vector from 0, and for each training, generally the norm gonna goes up. So if you stop earlier, you won't give norm much chance to grow too big

Hyperparameters

How to choose them?

Hyperparameters



Definition: hyperparameters

Properties of the learning algorithm (features, regularization parameter λ , number of iterations T , step size η , etc.).

How do we choose hyperparameters?

Choose hyperparameters to minimize $\mathcal{D}_{\text{train}}$ error? **No** - solution would be to include all features, set $\lambda = 0$, $T \rightarrow \infty$.

Choose hyperparameters to minimize $\mathcal{D}_{\text{test}}$ error? **No** - choosing based on $\mathcal{D}_{\text{test}}$ makes it an unreliable estimate of error!

Validation

Problem: can't use test set!

Solution: randomly take out 10-50% of training data and use it instead of the test set to estimate test error.

$\mathcal{D}_{\text{train}} \setminus \mathcal{D}_{\text{val}}$	\mathcal{D}_{val}	$\mathcal{D}_{\text{test}}$
---	----------------------------	-----------------------------



Definition: validation set

A **validation set** is taken out of the training data which acts as a surrogate for the **test set**.

Normal Training Process

Development cycle



Problem: simplified named-entity recognition

Input: a string x (e.g., *Governor [Gavin Newsom] in*)

Output: y , whether x contains a person or not (e.g., +1)



Algorithm: recipe for success

- Split data into train, val, test
- Look at data to get intuition
- Repeat:
 - Implement feature / tune hyperparameters
 - Run learning algorithm
 - Sanity check train and val error rates, weights
 - Look at errors to brainstorm improvements
- Run on test set to get final error rates

You can start with quite specific features and then generalize it

Unsupervised learning

Fully labeled data is very expensive to obtain

Supervision?

Supervised learning:

- Prediction: $\mathcal{D}_{\text{train}}$ contains input-output pairs (x, y)
- Fully-labeled data is very **expensive** to obtain (we can maybe get thousands of labeled examples)

Unsupervised learning:

- Clustering: $\mathcal{D}_{\text{train}}$ only contains inputs x
- Unlabeled data is much **cheaper** to obtain (we can maybe get billions of unlabeled examples)

Clustering

Clustering



Definition: clustering

Input: training set of input points

$$\mathcal{D}_{\text{train}} = \{x_1, \dots, x_n\}$$

Output: assignment of each point to a cluster

$$[z_1, \dots, z_n] \text{ where } z_i \in \{1, \dots, K\}$$

$z_1 \dots z_k$

Tells me which of these K clusters I'm in

K-means objective

Setup:

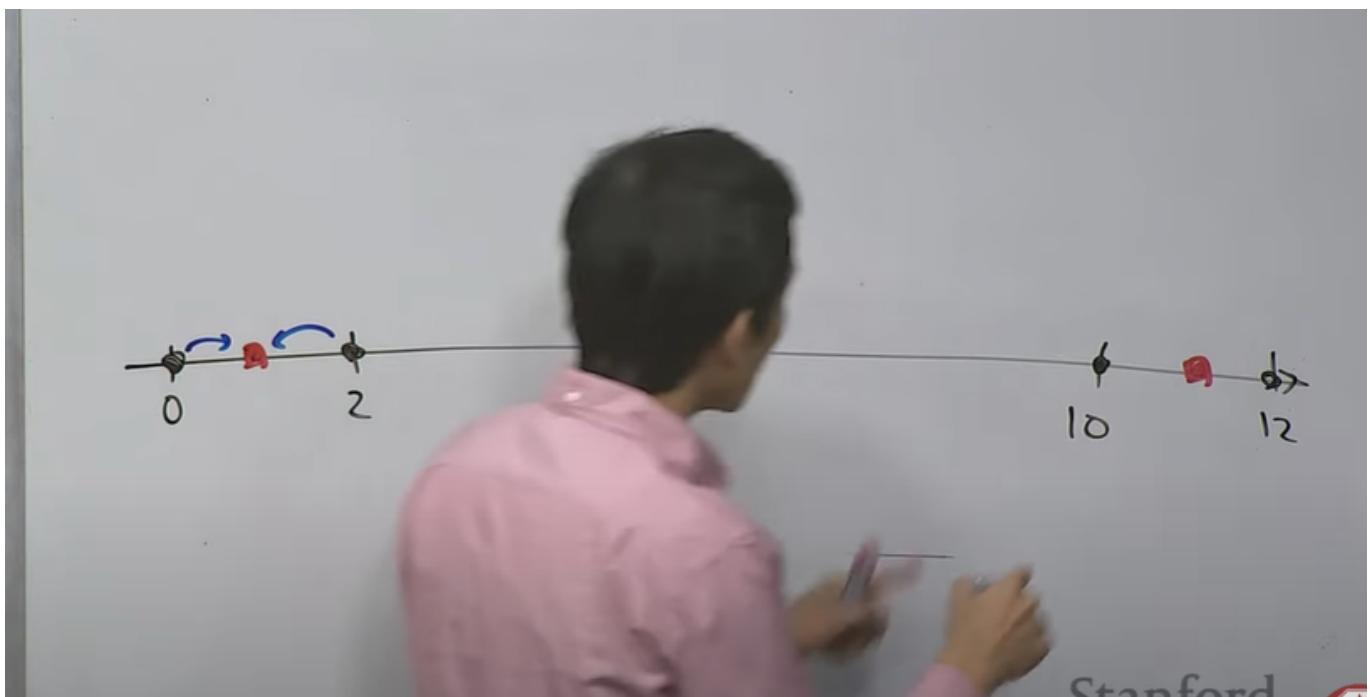
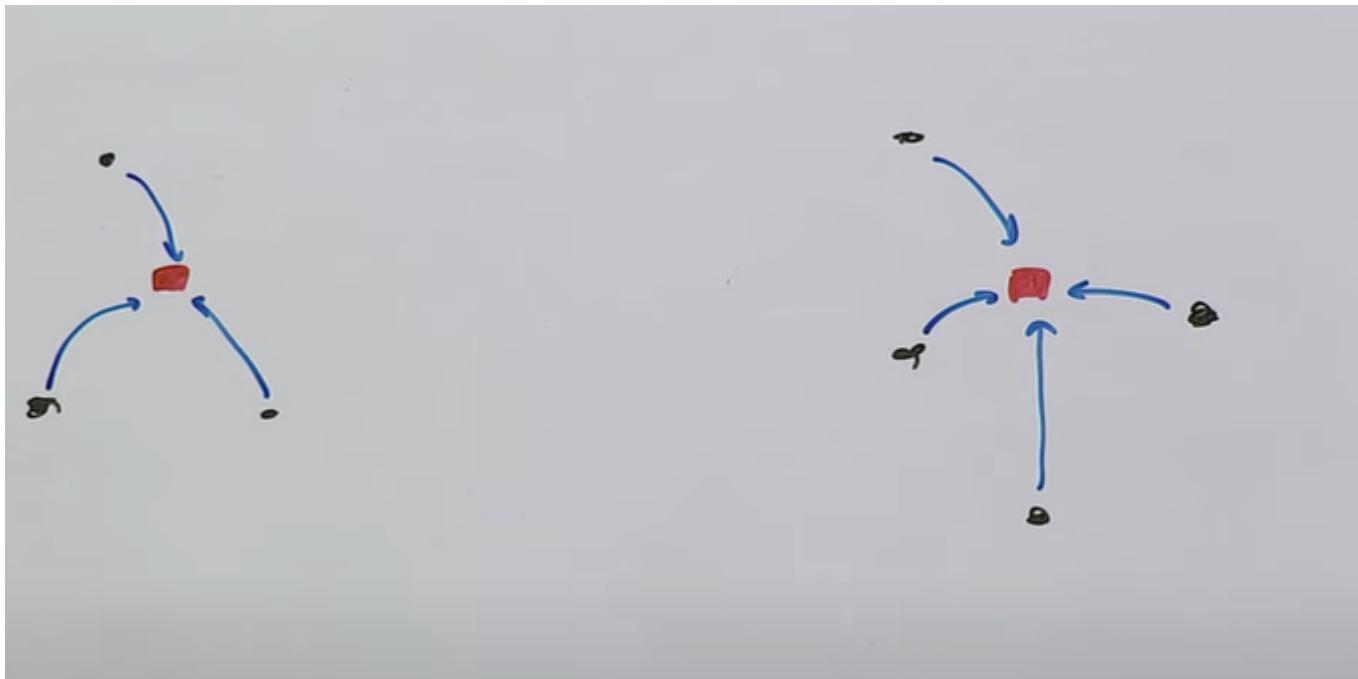
- Each cluster $k = 1, \dots, K$ is represented by a **centroid** $\mu_k \in \mathbb{R}^d$
- Intuition: want each point $\phi(x_i)$ close to its assigned centroid μ_{z_i}

Objective function:

$$\text{Loss}_{\text{kmeans}}(z, \mu) = \sum_{i=1}^n \|\phi(x_i) - \mu_{z_i}\|^2$$

For each cluster, there's gonna be a centroid (质量重心)

The Objective function For each point I measure the distance between that point and the centroid associated And try to make this number as smaller as possible



A simple example, you compare the distance to different centroids, and choose the closest one

K-means algorithm

$$\min_z \min_{\mu} \text{Loss}_{\text{kmeans}}(z, \mu)$$



Key idea: alternating minimization

Tackle **hard** problem by solving **two** easy problems.

Now

it's like the chicken egg problem IF i know the centroid, then I can do assignments IF i know assignments, then I can easily figure out the centroid

But how to get those two?

K-mean Algo

K is a hyperparameter How to u choose K? You can try different K and choose based on loss

For a fixed iteration of T

K-means algorithm

Objective:

$$\min_z \min_{\mu} \text{Loss}_{\text{kmeans}}(z, \mu)$$



Algorithm: K-means

Initialize μ_1, \dots, μ_K randomly.

For $t = 1, \dots, T$:

Step 1: set assignments z given μ

Step 2: set centroids μ given z

Step 1 Given Centriod and figure out the Cluster Assignments

K-means algorithm (Step 1)

Goal: given centroids μ_1, \dots, μ_K , assign each point to the best centroid.

**Algorithm: Step 1 of K-means**

For each point $i = 1, \dots, n$:

Assign i to cluster with closest centroid:

$$z_i \leftarrow \arg \min_{k=1, \dots, K} \|\phi(x_i) - \mu_k\|^2.$$

Just check the distance you compare the distance to different centriods, and assign the point to the closet one

Step 2 Given Cluster assignments and try to find the centroid

K-means algorithm (Step 2)

Goal: given cluster assignments z_1, \dots, z_n , find the best centroids μ_1, \dots, μ_K .

**Algorithm: Step 2 of K-means**

For each cluster $k = 1, \dots, K$:

Set μ_k to average of points assigned to cluster k :

$$\mu_k \leftarrow \frac{1}{|\{i : z_i = k\}|} \sum_{i:z_i=k} \phi(x_i)$$

Mathematically you sum up all the points (feature vectors), and devided by the number of points => You get the centroid now!

Limitation

Local minima

K-means is guaranteed to converge to a local minimum, but is not guaranteed to find the global minimum.



[demo: getting stuck in local optima, seed = 100]

Solutions:

- Run multiple times from different random initializations
- Initialize with a heuristic (K-means++)

This is guaranteed to converge to a local minimum

K-means++ Initialize points far away from each other, generally works well

Summary



Summary

- Feature extraction (think hypothesis classes) [modeling]
- Prediction (linear, neural network, k-means) [modeling]
- Loss functions (compute gradients) [modeling]
- Optimization (stochastic gradient, alternating minimization) [learning]
- Generalization (think development cycle) [modeling]

Machine learning



Key idea: learning

Programs should improve with experience.

So far: reflex-based models

Next time: state-based models