CSC 311: Introduction to Machine Learning

Lecture 3 - Bagging, Linear Models I

Rahul G. Krishnan   Alice Gao

University of Toronto, Fall 2022

Outline

1. Introduction
2. Bias-Variance Decomposition understand generalization.
3. Bagging an ensemble method.
4. Linear Regression a modular approach to ML.
5. Vectorization
6. Optimization
7. Feature Mappings
8. Regularization

Announcements

- HW1 is due next Monday (10% late penalty for each late day, no credit after 3 days).
- We have arranged TA office hours (on website) for the assignment.
- Go to the earliest possible ones you can attend.
- **Manage your time well!** If you wait till the last TA session, you may have a long wait to ask your question.

Today

- Ensembling methods combine multiple models and can perform better than the individual members.
  - We've seen many individual models (KNN, decision trees)
- Bagging: Train models independently on random "resamples" of the training data.
- Linear regression, our first parametric learning algorithm.
  - Illustrates a modular approach to learning algorithms.

Bias/Variance Decomposition

- prediction y at a query $\mathbf{x}$ is a random variable (where the randomness comes from the choice of dataset),
- $y_\star$ is the optimal deterministic prediction, and
- t is a random target sampled from the true conditional $p(t|\mathbf{x})$.

$$\mathbb{E}[(y-t)^2] = \underbrace{(y_\star - \mathbb{E}[y])^2}_{\text{bias}} + \underbrace{\mathrm{Var}(y)}_{\text{variance}} + \underbrace{\mathrm{Var}(t)}_{\text{Bayes error}}$$

Interpretations

$$\mathbb{E}[(y-t)^2] = \underbrace{(y_\star - \mathbb{E}[y])^2}_{\text{bias}} + \underbrace{\mathrm{Var}(y)}_{\text{variance}} + \underbrace{\mathrm{Var}(t)}_{\text{Bayes error}}$$

Bias/variance decomposes the expected loss into three terms:

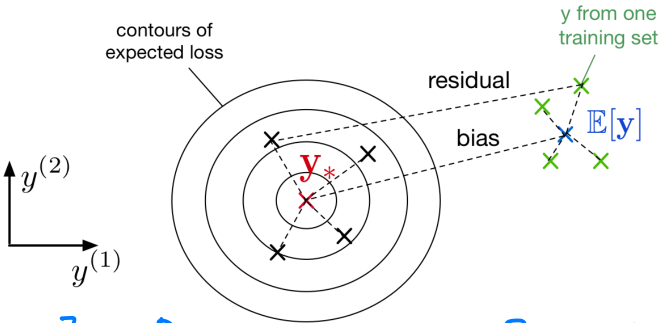
- bias: how wrong the expected prediction is (corresponds to under-fitting)
- variance: the amount of variability in the predictions (corresponds to over-fitting)
- Bayes error: the inherent unpredictability of the targets

Often loosely use "bias" for "under-fitting" and "variance" for "over-fitting".

Overly Simple Model

An overly **simple** model (e.g. KNN with large k) might have

- **high bias**
  (cannot capture the structure in the data)
- **low variance**
  (enough data to get stable estimates)



E[loss] = Bias + Variance + Bayes Error

expected squared loss error = bias + variance + Bayes error.

generalization error : average squared length $\|y-t\|^2$ of the line segment "residual".

bias : average squared length $\|E[y]-y_*\|^2$ of the line segment "bias"

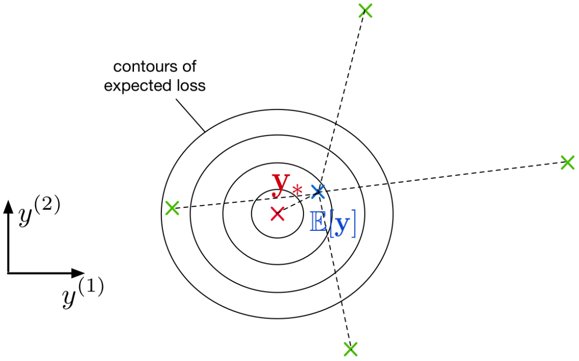
variance : spread in green x's.

Bayes error : spread in black x's.

Overly Complex Model

An overly **complex** model (e.g. KNN with $k = 1$) might have

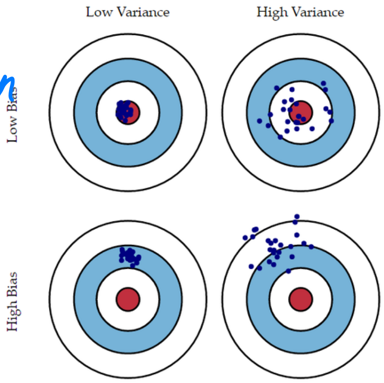
- **low bias**
  (learns all the relevant structure)
- **high variance**
  (fits the quirks of the data you happened to sample)



Bias/Variance Decomposition: Another Visualization

- The following graphic summarizes the previous two slides:

bias : distance between middle point and target



variance : spread of the points.

A: Bayes error

main idea: to average many noisy but approximately unbiased models and reduce the variance.

Bagging Motivation

- Sample m independent training sets from p_{sample}.
- Compute the prediction y_i using each training set.
- Compute the average prediction $y = \frac{1}{m}\sum_{i=1}^{m} y_i$.
- How does this affect the three terms of the expected loss?
  - **Bias:** unchanged,
    since the averaged prediction has the same expectation

    $$\mathbb{E}[y] = \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m} y_i\right] = \mathbb{E}[y_i]$$

  - **Variance:** reduced,
    since we are averaging over independent predictions

    $$\text{Var}[y] = \text{Var}\left[\frac{1}{m}\sum_{i=1}^{m} y_i\right] = \frac{1}{m^2}\sum_{i=1}^{m}\text{Var}[y_i] = \frac{1}{m}\text{Var}[y_i].$$

  - **Bayes error:** unchanged,
    since we have no control over it

$$E\left[\frac{1}{m}\sum_{i=1}^{m} y_i\right] = \frac{1}{m}\sum_{i=1}^{m} E[y_i] = E[y_i]$$

$\uparrow$ linearity of expectation

$\nwarrow$ i^{th} training set is drawn i.i.d. from P_{sample}, so $E[y_i]$ is the same for every i.

Each training set i is identically distributed, so the expectation of an average of the predictions is the same as the expectation of any one prediction y_i.

$$\mathrm{Var}[y] = \mathrm{Var}\left[\frac{1}{m}\sum_{i=1}^{m} y_i\right] \underset{\substack{\uparrow \\ \mathrm{Var}[aX] = a^2\,\mathrm{Var}[X]}}{=} \frac{1}{m^2}\mathrm{Var}\left[\sum_{i=1}^{m} y_i\right] \underset{\substack{\uparrow \\ \text{the predictions } y_i\text{'s} \\ \text{are independent.}}}{=} \frac{1}{m^2}\sum_{i=1}^{m}\mathrm{Var}[y_i]$$

$$= \frac{1}{m}\mathrm{Var}[y_i]$$

↖ each training set is drawn i.i.d. from P_{sample}, so $\mathrm{Var}[y_i]$ is the same for every i.

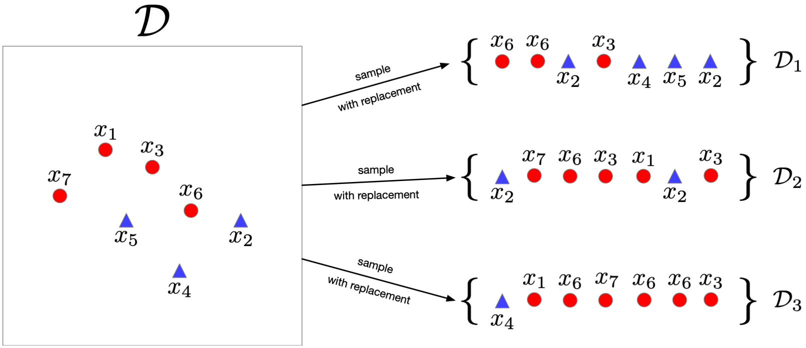
If each prediction y_i has the same variance $\mathrm{Var}[y_i]$, then the average of M such predictions has variance $\frac{1}{m}\mathrm{Var}[y_i]$.

Bagging: The Idea

- In practice, p_{sample} is often expensive to sample from. So training separate models on independently sampled datasets is very wasteful of data!
- Given training set $\mathcal{D}$, use the empirical distribution $p_{\mathcal{D}}$ as a proxy for p_{sample}. This is called bootstrap aggregation or bagging .
  - Take a dataset $\mathcal{D}$ with n examples.
  - Generate m new datasets ("resamples" or "bootstrap samples")
  - Each dataset has n examples sampled from $\mathcal{D}$ with replacement.
  - Average the predictions of models trained on the m datasets.
- One of the most important ideas in statistics!
  - Intuition: As $|\mathcal{D}| \to \infty$, we have $p_{\mathcal{D}} \to p_{\text{sample}}$.

Bagging Example 1/2

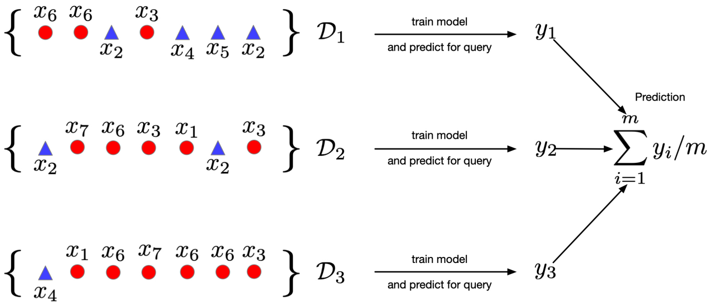
Create $m = 3$ datasets by sampling from D with replacement.
Each dataset contains $n = 7$ examples.



$\mathcal{D}$
x_1 x_3 x_7 x_6 x_5 x_2 x_4
sample with replacement
sample with replacement
sample with replacement
$\{x_6, x_6, x_2, x_3, x_4, x_5, x_2\}$ $\mathcal{D}_1$
$\{x_2, x_7, x_6, x_3, x_1, x_2, x_3\}$ $\mathcal{D}_2$
$\{x_4, x_1, x_6, x_7, x_6, x_6, x_3\}$ $\mathcal{D}_3$

Bagging Example 2/2

Generate prediction y_i using dataset D_i.
Average the predictions.



Aggregating Predictions for Binary Classification

- Classifier i outputs a prediction y_i
- y_i can be real-valued $y_i \in [0,1]$ or a binary value $y_i \in \{0,1\}$
- Average the predictions and apply a threshold.

$$y_{\text{bagged}} = \mathbb{I}\left(\frac{1}{m}\sum_{i=1}^{m} y_i > 0.5\right)$$

- Same as majority vote.

Bagging Properties

- A bagged classifier can be stronger than the average model.
  - E.g. on “Who Wants to be a Millionaire”, “Ask the Audience” is much more effective than “Phone a Friend”.
- But, if m datasets are NOT independent, don’t get the $\frac{1}{m}$ variance reduction.
- Reduce correlation between datasets by introducing *additional* variability
  - Invest in a diversified portfolio, not just one stock.
  - Average over multiple algorithms, or multiple configurations of the same algorithm.

Random Forests

Trees are ideal for bagging since they are low-bias and high-variance models.

- A trick to reduce correlation between bagged decision trees: For each node, choose a random subset of features and consider splits on these features only.
- Probably the best black-box machine learning algorithm.
  - works well with no tuning.
  - widely used in Kaggle competitions.

Bagging Summary

variance

Reduces over-fitting by averaging predictions.

In most competition winners.
A small ensemble often better than a single great model.

Limitations:

- Does not reduce bias in case of squared error.
- Correlation between classifiers means less variance reduction.
  Add more randomness in Random Forests.
- Weighting members equally may not be the best.
  Weighted ensembling often leads to better results if members are very different.

Main Takeaways:

- What is the main idea in bagging?
  - to average multiple noisy but unbiased models to reduce variance. (over-fitting).
  - does not reduce bias.

- Describe the bagging procedure.
  - Sample multiple data-sets w/ replacement.
  - Generate a prediction using each dataset.
  - Aggregate the predictions (averaging or majority voting).

- How can we reduce correlation between trees in a random forest?
  - For each node, choose a subset of the features and consider splits on these features only.

Linear Regression

- Task: predict scalar-valued targets (e.g. stock prices)
- Architecture: linear function of the inputs

A Modular Approach to ML

- choose a model describing relationships between variables
- define a loss function quantifying how well the model fits the data
- choose a regularizer expressing preference over different models
- fit a model that minimizes the loss function and satisfies the regularizer's constraint/penalty, possibly using an optimization algorithm

Supervised Learning Setup

- a collection of training examples labeled w/ correct outputs.

- Input $\mathbf{x} \in \mathcal{X}$ (a vector of features)
- Target $t \in \mathcal{T}$
- Data $\mathcal{D} = \{(\mathbf{x}^{(i)}, t^{(i)})$ for $i = 1, 2, ..., N\}$
- Objective: learn a function $f : \mathcal{X} \to \mathcal{T}$ based on the data such that $t \approx y = f(\mathbf{x})$

Model

model: the set of allowable functions that compute predictions from the inputs.

Model: a *linear* function of the features $\mathbf{x} = (x_1, \dots, x_D) \in \mathbb{R}^D$ to make prediction $y \in \mathbb{R}$ of the target $t \in \mathbb{R}$:

$$y = w_1 x_1 + w_2 x_2 + \cdots + w_D x_D + b$$

$$y = f(\mathbf{x}) = \sum_j w_j x_j + b = \mathbf{w}^\top \mathbf{x} + b$$

- Parameters are weights $\mathbf{w}$ and the bias/intercept b
- Want the prediction to be close to the target: $y \approx t$.

  How do we measure this?

Loss Function

$\mathcal{L}$: is a function of prediction & target.
doesn't care how you produced the prediction.

Loss function $\mathcal{L}(y, t)$ defines how badly the algorithm's prediction y fits the target t for some example $\mathbf{x}$.

small when y and t are close together
large when y and t are far apart.

Squared error loss function: $\mathcal{L}(y, t) = \frac{1}{2}(y - t)^2$

- $y - t$ is the residual, and we want to minimize this magnitude
- $\frac{1}{2}$ makes calculations convenient.

The optimization problem: minimize cost function w.r.t to model parameters.

Cost function: loss function averaged over all training examples also called *empirical* or *average loss*.

a function of the model parameters w, b and t.

$$\mathcal{J}(\mathbf{w}, b) = \frac{1}{2N} \sum_{i=1}^{N} \left(y^{(i)} - t^{(i)} \right)^2 = \frac{1}{2N} \sum_{i=1}^{N} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - t^{(i)} \right)^2$$

choose w, b to minimize J.

Loops v.s. Vectorized Code

two options

option ①

- We can compute prediction for one data point using a for loop:

```
y = b
for j in range(M):
    y += w[j] * x[j]
```

- But, excessive super/sub scripts are hard to work with, and Python loops are slow.

option ②

- Instead, we express algorithms using vectors and matrices.

$$\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix} \qquad \mathbf{w} = (w_1, \ldots, w_D)^\top \qquad \mathbf{x} = (x_1, \ldots, x_D)^\top \qquad \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{pmatrix}$$

$$y = \mathbf{w}^\top \mathbf{x} + b$$

- This is simpler and executes much faster:

```
y = np.dot(w, x) + b
```

Benefits of Vectorization

Why vectorize?

shorter more compact

- The code is simpler and more readable. No more dummy variables/indices!
- Vectorized code is much faster

  Python is high-level language.
  for loops incur interpreter overhead.

  - Cut down on Python interpreter overhead
  - Use highly optimized linear algebra libraries (hardware support)
  - Matrix multiplication very fast on GPU → highly parallelizable.

take time to become comfortable w/ vectorized form.

You will practice switching in and out of vectorized form.

- Some derivations are easier to do element-wise
- Some algorithms are easier to write/understand using for-loops and vectorize later for performance

practice this intentionally.

Predictions for the Dataset

- Put training examples into a design matrix $\mathbf{X}$.
- Put targets into the target vector $\mathbf{t}$.
- We can compute the predictions for the whole dataset.

N examples.
D features.

$$\mathbf{X}\mathbf{w} + b\mathbf{1} = y$$

$$\begin{array}{cc} & D \\ \begin{array}{c} 1 \text{ example} \\ N \end{array} & \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_D^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_D^{(2)} \\ \vdots & \vdots & & \vdots \\ x_1^{(N)} & x_2^{(N)} & \dots & x_D^{(N)} \end{pmatrix} \end{array} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix} + b \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{pmatrix} \quad \begin{pmatrix} t^{(1)} \\ \vdots \\ t^{(N)} \end{pmatrix}$$

D-dim vector

N-dim vector.

one feature
one input dimension.

$$x_1^{(1)} w_1 + x_2^{(1)} w_2 + \dots + x_D^{(1)} w_D + b$$

Computing Squared Error Cost

We can compute the squared error cost across the whole dataset.

$$\mathbf{y} = \mathbf{X}\mathbf{w} + b\mathbf{1}$$

$$\frac{1}{2N}\|Xw+b1-t\|^2 = \quad \mathcal{J} = \frac{1}{2N}\|\mathbf{y} - \mathbf{t}\|^2 \rightarrow \text{Euclidean norm } L^2 \text{ norm.}$$

Sometimes we may use $\mathcal{J} = \frac{1}{2}\|\mathbf{y} - \mathbf{t}\|^2$, without a normalizer. This would correspond to the sum of losses, and not the averaged loss. The minimizer does not depend on N (but optimization might!).

Combining Bias and Weights

We can combine the bias and the weights and add a column of 1's to design matrix.

Our predictions become

$$\mathbf{y} = \mathbf{X}\mathbf{w}.$$

$$\mathbf{X} = \begin{bmatrix} 1 & [\mathbf{x}^{(1)}]^\top \\ 1 & [\mathbf{x}^{(2)}]^\top \\ 1 & \vdots \end{bmatrix} \in \mathbb{R}^{N \times (D+1)} \text{ and } \mathbf{w} = \begin{bmatrix} b \\ w_1 \\ w_2 \\ \vdots \end{bmatrix} \in \mathbb{R}^{D+1}$$

Solving the Minimization Problem

(assume that we combine b into the w vector.)

Goal is to minimize the cost function $\mathcal{J}(\mathbf{w})$.

Recall: the minimum of a smooth function (if it exists) occurs at a critical point, i.e. point where the derivative is zero.

How do we find weights w such that $\frac{\partial J}{\partial w} = 0$

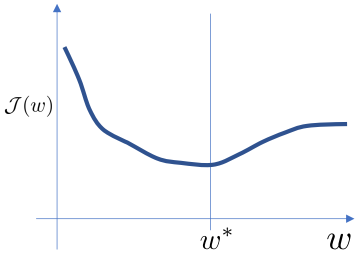
$$\nabla_{\mathbf{w}} \mathcal{J} = \frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial \mathcal{J}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{J}}{\partial w_D} \end{pmatrix}$$

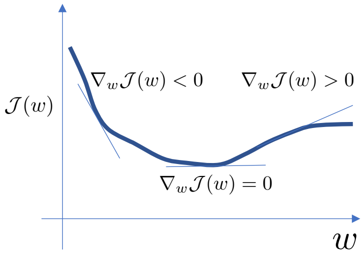
Solutions may be direct or iterative.

- Direct solution: set the gradient to zero and solve in closed form — directly find provably optimal parameters.
- Iterative solution: repeatedly apply an update rule that gradually takes us closer to the solution.

Minimizing 1D Function

- Consider $\mathcal{J}(w)$ where w is 1D.
- Seek $w = w^*$ to minimize $\mathcal{J}(w)$.
- The gradients can tell us where the maxima and minima of functions lie
- **Strategy:** Write down an algebraic expression for $\nabla_w \mathcal{J}(w)$. Set $\nabla_w \mathcal{J}(w) = 0$. Solve for w.




$\mathcal{J}(w)$
$\nabla_w \mathcal{J}(w) < 0$
$\nabla_w \mathcal{J}(w) > 0$
$\nabla_w \mathcal{J}(w) = 0$
w

Direct Solution for Linear Regression

no $\frac{1}{N}$, sum of losses.

- Seek $\mathbf{w}$ to minimize $\mathcal{J}(\mathbf{w}) = \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2$
- Taking the gradient with respect to $\mathbf{w}$ and setting it to $\mathbf{0}$, we get:

$$\nabla_{\mathbf{w}}\mathcal{J}(\mathbf{w}) = \mathbf{X}^\top\mathbf{X}\mathbf{w} - \mathbf{X}^\top\mathbf{t} = \mathbf{0}$$

See course notes for derivation.

$$\boxed{X^T X} w = \boxed{X^T t}$$

$$A \; w = C$$

- Optimal weights:

$$\mathbf{w}^* = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{t}$$

a system of D linear equations w/ D unknowns/variables.

- Few models (like linear regression) permit direct solution.

unusual to have a closed-form solution.
in most cases, the system of equations is non-linear. and doesn't have closed-form solutions. only a handful algorithms in this course have closed form solutions.

Direct Solution for Linear Regression

$$J = \frac{1}{2} \sum_{i=1}^{N} \left(\sum_{j=1}^{D} w_j x_j^{(i)} - t^{(i)} \right)^2$$

$$\Rightarrow \quad \frac{\partial J}{\partial w_j} = \sum_{i=1}^{N} x_j^{(i)} \left(\sum_{j'=1}^{D} w_{j'} x_{j'}^{(i)} - t^{(i)} \right) = 0$$

$$\Rightarrow \quad \sum_{j'=1}^{D} \left(\sum_{i=1}^{N} x_j^{(i)} x_{j'}^{(i)} \right) w_{j'} - \sum_{i=1}^{N} x_j^{(i)} t^{(i)} = 0$$

$$\Rightarrow \quad \sum_{j'=1}^{D} \underbrace{\left(\sum_{i=1}^{N} x_j^{(i)} x_{j'}^{(i)} \right)}_{A_{jj'}} w_{j'} = \underbrace{\sum_{i=1}^{N} x_j^{(i)} t^{(i)}}_{c_j}$$

$$\sum_{j'=1}^{D} A_{jj'} w_{j'} = c_j , \quad \forall j = 1, \dots, D.$$

Direct Solution for Linear Regression. (vectorized form).

$$J = \frac{1}{2}(Xw - t)^T(Xw - t)$$

$$\Rightarrow \quad \frac{\partial J}{\partial w} = X^T(Xw - t) = 0$$

$$\Rightarrow \quad X^TXw - X^Tt = 0$$

$$\Rightarrow \quad \underbrace{X^TX}\, w = \underbrace{X^Tt}$$

$$\Rightarrow \quad w = (X^TX)^{-1}X^Tt$$

Iterative Solution: Gradient Descent

- Many optimization problems don't have a direct solution.
- A more broadly applicable strategy is gradient descent.
- Gradient descent is an iterative algorithm, which means we apply an update repeatedly until some criterion is met.
- We initialize the weights to something reasonable (e.g. all zeros) and repeatedly adjust them in the direction of steepest descent.

that most decreases the cost function.

until the weights converge or stop changing much.
or until we get tired of waiting.

Deriving Update Rule

In what direction should I update w?

change w in the direction opposite the gradient.

Observe: positive

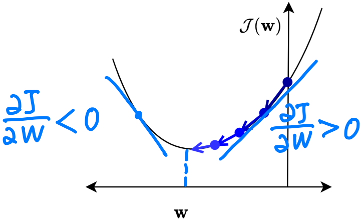
- if $\partial \mathcal{J} / \partial w_j > 0$, then decreasing $\mathcal{J}$ requires decreasing w_j.
- if $\partial \mathcal{J} / \partial w_j < 0$, then decreasing $\mathcal{J}$ requires increasing w_j.

negative

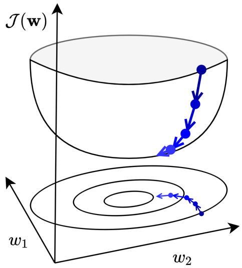
The following update always decreases the cost function for small enough α (unless $\partial \mathcal{J} / \partial w_j = 0$):

$$w_j \leftarrow w_j - \alpha \frac{\partial \mathcal{J}}{\partial w_j}$$

for 1D function,
gradient = slope



$\mathcal{J}(\mathbf{w})$
$\frac{\partial J}{\partial w} < 0$
$\frac{\partial J}{\partial w} > 0$
$\mathbf{w}$



Setting Learning Rate

How much should I change w at each step ?

Gradient descent update rule:

$$w_j \leftarrow w_j - \alpha \frac{\partial \mathcal{J}}{\partial w_j}$$

$\alpha > 0$ is a learning rate (or step size).

- The larger α is, the faster $\mathbf{w}$ changes.
- Values are typically small, e.g. 0.01 or 0.0001.
- We'll see later how to tune the learning rate.
- If minimizing total loss rather than average loss, needs a smaller learning rate ($\alpha' = \alpha/N$).

Gradient Descent Intuition

- Gradient descent gets its name from the gradient, the direction of fastest *increase.* (steepest ascent)

$$\nabla_{\mathbf{w}}\mathcal{J} = \frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial \mathcal{J}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{J}}{\partial w_D} \end{pmatrix}$$

- Update rule in vector form:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}}$$

  Update rule for linear regression:

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)})\, \mathbf{x}^{(i)}$$

- Gradient descent updates $\mathbf{w}$ in the direction of fastest *decrease.*
- Once it converges, we get a critical point, i.e. $\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \mathbf{0}$.

Gradient Descent Update for Linear Regression.

$$w \leftarrow w - \alpha \frac{\partial J}{\partial w} \quad \text{or} \quad w_j \leftarrow w_j - \alpha \frac{\partial J}{\partial w_j}$$

$$\begin{cases} \dfrac{\partial J}{\partial w_j} = \dfrac{1}{N} \displaystyle\sum_{i=1}^{N} x_j^{(i)} \left(\sum_{j'=1}^{D} w_{j'} x_{j'}^{(i)} - t^{(i)} \right) \\ w_j \leftarrow w_j - \dfrac{\alpha}{N} \displaystyle\sum_{i=1}^{N} x_j^{(i)} \left(\sum_{j'=1}^{D} w_{j'} x_{j'}^{(i)} - t^{(i)} \right) \end{cases}$$

$$\begin{cases} \dfrac{\partial J}{\partial w} = \dfrac{1}{N} X^T (Xw - t) \\ w \leftarrow w - \dfrac{\alpha}{N} X^T (Xw - t) \end{cases}$$

(vectorized form)

Why Use Gradient Descent?

direct solution : exact optimum.

gradient descent : approach the optimum gradually.

closed form solution for a handful of models., GD as long as we can compute gradient.

- Applicable to a much broader set of models.
- Easier to implement than direct solutions.
- More efficient than direct solution for regression in high-dimensional space. solving a linear system more expensive than a gradient update.
  - The linear regression direction solution $(\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{t}$ requires matrix inversion, which is $\mathcal{O}(D^3)$. GD can be much faster.
  - Gradient descent update costs $\mathcal{O}(ND)$ or less with stochastic gradient descent.
  - Huge difference if D is large.

- Many software packages can compute gradient automatically. & efficiently. no need to do it by hand.

Even if we have direct solution, GD is more practical.

Feature Mapping

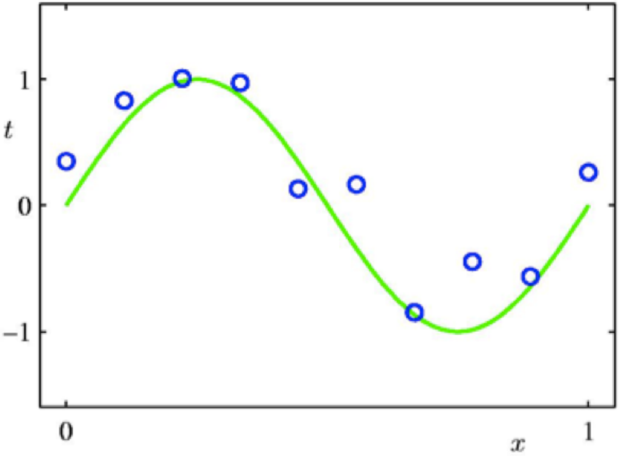
Linear regression sounds pretty limited.

Can we use linear regression to model a non-linear relationship?

- Map the input features to another space $\boldsymbol{\psi}(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}^d$.
- Treat the mapped feature (in $\mathbb{R}^d$) as the input of a linear regression procedure.

Modeling a Non-Linear Relationship

$y = w_3 x^3 + w_2 x^2 + w_1 x + w_0.$



use linear regression as (x, x^2, x^3) as inputs.

Polynomial Feature Mapping

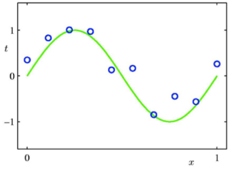
Fit the data using a degree-M polynomial function of the form:

not linear in x.

$$y = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{i=0}^{M} w_i x^i$$

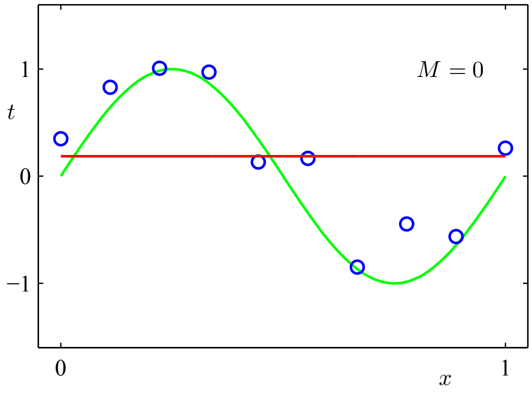
but linear in $(1, x, x^2, x^3, \ldots, x^M)$

- The feature mapping is $\boldsymbol{\psi}(x) = [1, x, x^2, \ldots, x^M]^\top$.
- $y = \boldsymbol{\psi}(x)^\top \mathbf{w}$ is linear in $w_0, w_1, \ldots$. instead of $x^T w$.
- Use linear regression to find $\mathbf{w}$.



Polynomial Feature Mapping with $M = 0$

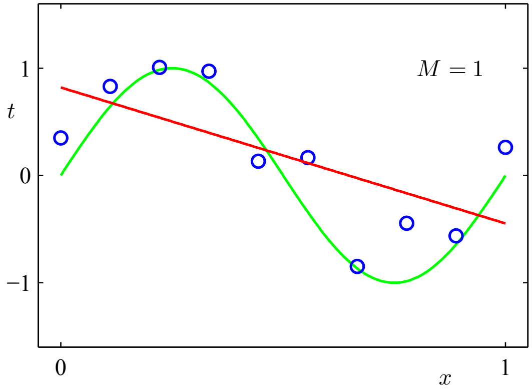
$$y = w_0$$



[Pattern Recognition and Machine Learning, Christopher Bishop.]

Polynomial Feature Mapping with $M = 1$

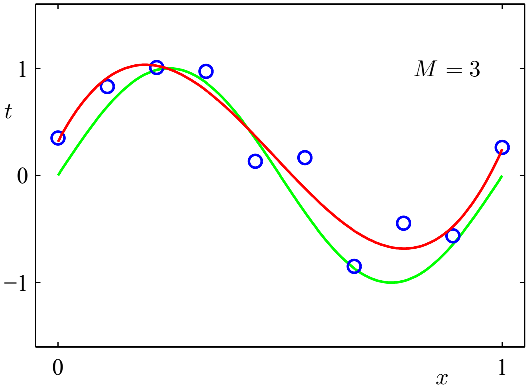
$$y = w_0 + w_1 x$$



[Pattern Recognition and Machine Learning, Christopher Bishop.]

Polynomial Feature Mapping with $M = 3$

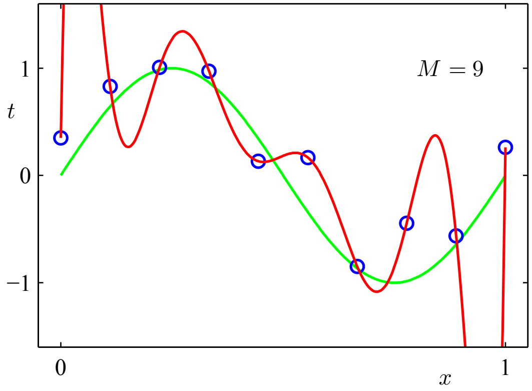
$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$



[Pattern Recognition and Machine Learning, Christopher Bishop.]

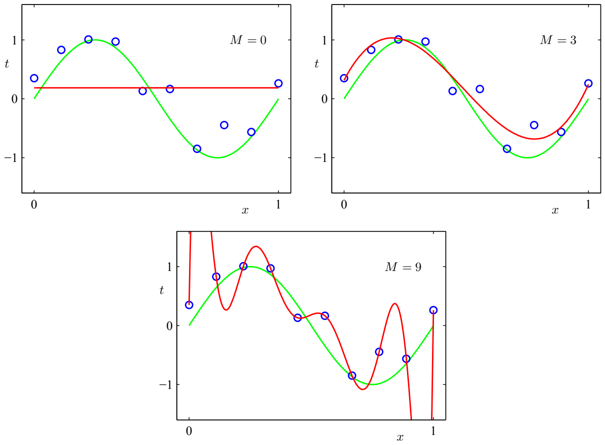
Polynomial Feature Mapping with $M = 9$

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots + w_9 x^9$$



[Pattern Recognition and Machine Learning, Christopher Bishop.]

Model Complexity and Generalization

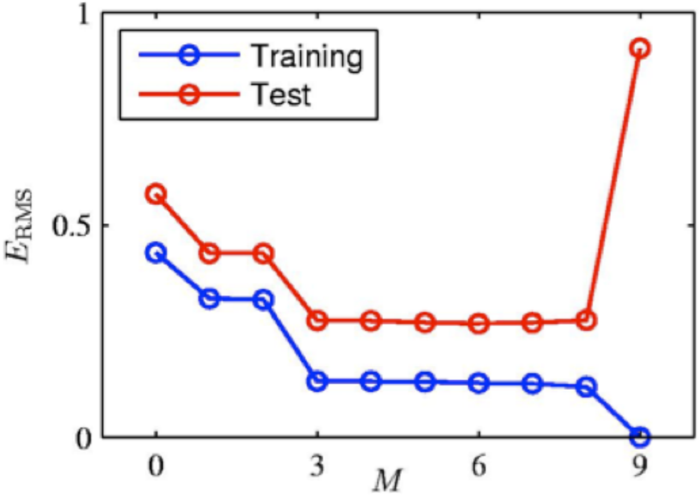


Under-fitting (M=0):
Model is too simple, doesn't fit data well.

Good model (M=3):
Small test error, generalizes well.

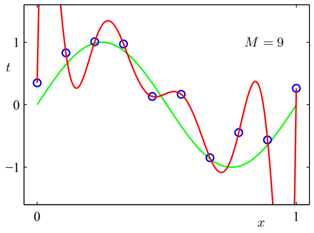
Over-fitting (M=9):
Model is too complex, fits data perfectly.

Model Complexity and Generalization



Model Complexity and Generalization

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43



- As M increases, the magnitude of coefficients gets larger.
- For $M = 9$, the coefficients have become finely tuned to the data.
- Between data points, the function exhibits large oscillations.

Feature mapping is useful, but not a silver bullet / magical weapon.

- must choose features in advance. not easy to choose good features. feature engineering takes time and creativity.
- in high dimensions, feature representation can get very large.

We will use neural networks to learn non-linear predictions directly from inputs. This eliminates the need for hand-engineering of features.

Regularization

- The degree M of the polynomial controls the model's complexity.
- The value of M is a hyperparameter for polynomial expansion, just like k in KNN. We can tune it using a validation set.
- Restricting the number of parameters / basis functions (M) is a crude approach to controlling the model complexity.
- Another approach: keep the model large, but regularize it
  - Regularizer: a function that quantifies how much we prefer one hypothesis vs. another

L^2 (or ℓ_2) Regularization

- Encourage the weights to be small by choosing the L^2 penalty as our regularizer.

$$\mathcal{R}(\mathbf{w}) = \tfrac{1}{2}\|\mathbf{w}\|_2^2 = \frac{1}{2}\sum_j w_j^2.$$

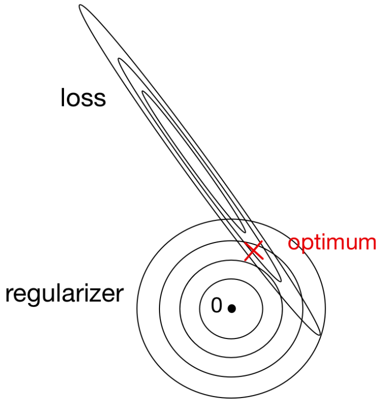
- The regularized cost function makes a tradeoff between the fit to the data and the norm of the weights.

$$\mathcal{J}_{\text{reg}}(\mathbf{w}) = \mathcal{J}(\mathbf{w}) + \lambda\mathcal{R}(\mathbf{w}) = \mathcal{J}(\mathbf{w}) + \frac{\lambda}{2}\sum_j w_j^2$$

- If you fit training data poorly, $\mathcal{J}$ is large.
  If the weights are large in magnitude, $\mathcal{R}$ is large.
- Large λ penalizes weight values more.
- λ is a hyperparameter we can tune with a validation set.

L^2 (or ℓ_2) Regularization

- The geometric picture:



L^2 Regularized Least Squares: Ridge regression

For the least squares problem, we have $\mathcal{J}(\mathbf{w}) = \frac{1}{2N}\|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2$.

- When $\lambda > 0$ (with regularization), regularized cost gives

$$\begin{aligned}\mathbf{w}_\lambda^{\text{Ridge}} = \underset{\mathbf{w}}{\operatorname{argmin}}\, \mathcal{J}_{\text{reg}}(\mathbf{w}) =& \underset{\mathbf{w}}{\operatorname{argmin}}\, \frac{1}{2N}\|\mathbf{X}\mathbf{w} - \mathbf{t}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2 \\ =& (\mathbf{X}^\top\mathbf{X} + \lambda N\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{t}\end{aligned}$$

- The case $\lambda = 0$ (no regularization) reduces to least squares solution!
- Can also formulate the problem as

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{t}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

with solution

$$\mathbf{w}_\lambda^{\text{Ridge}} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{t}.$$

Direct Solution for Ridge Regression.

$$J_{reg}(w) = \frac{1}{2}\sum_{i=1}^{N}\left(\sum_{j=1}^{D} w_j x_j^{(i)} - t^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{D} w_j^2$$

$$\frac{\partial J_{reg}}{\partial w_j} = \sum_{i=1}^{N}\left(\sum_{j'=1}^{D} w_{j'} x_{j'}^{(i)} - t^{(i)}\right) x_j^{(i)} + \lambda w_j = 0$$

$$\sum_{i=1}^{N}\left(\sum_{j'=1}^{D} w_{j'} x_{j'}^{(i)} x_j^{(i)}\right) + \lambda w_j = \sum_{i=1}^{N} t^{(i)} x_j^{(i)}$$

Direct Solution for Ridge Regression (vectorized form)

$$J_{reg}(w) = \frac{1}{2}\|Xw - t\|^2 + \frac{\lambda}{2}\|w\|^2$$

$$\frac{\partial J_{reg}}{\partial w} = X^T(Xw - t) + \lambda w = 0$$

$$\Longrightarrow X^TXw - X^Tt + \lambda w = 0.$$

$$\Longrightarrow X^TXw - X^Tt + \lambda I w = 0$$, I is an identity matrix. $(Iw = w)$

$$\Longrightarrow (X^TX + \lambda I)w = X^Tt.$$

$$\Longrightarrow w = (X^TX + \lambda I)^{-1}X^Tt.$$

Gradient Descent under the L^2 Regularization

- Gradient descent update to minimize $\mathcal{J}$:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial}{\partial \mathbf{w}} \mathcal{J}$$

- The gradient descent update to minimize the L^2 regularized cost $\mathcal{J} + \lambda \mathcal{R}$ results in weight decay:

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \alpha \frac{\partial}{\partial \mathbf{w}} (\mathcal{J} + \lambda \mathcal{R}) \\ &= \mathbf{w} - \alpha \left(\frac{\partial \mathcal{J}}{\partial \mathbf{w}} + \lambda \frac{\partial \mathcal{R}}{\partial \mathbf{w}} \right) \\ &= \mathbf{w} - \alpha \left(\frac{\partial \mathcal{J}}{\partial \mathbf{w}} + \lambda \mathbf{w} \right) \\ &= (1 - \alpha\lambda)\mathbf{w} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}} \end{aligned}$$

Conclusions

Linear regression exemplifies recurring themes of this course:

- choose a model and a loss function
- formulate an optimization problem
- solve the minimization problem using one of two strategies
  - direct solution (set derivatives to zero)
  - gradient descent
- vectorize the algorithm, i.e. represent in terms of linear algebra
- make a linear model more powerful using features
- improve the generalization by adding a regularizer