Machine Learning

CSC 548, Artificial Intelligence II

Overview

- Supervised learning
- Generalization
- Unsupervised learning

Application: Spam Classification

- Input: $x =$ email message
- Output: $y \in \{\text{spam}, \text{not-spam}\}$
- Objective: obtain a predictor f where $f(x) = y$
    - in statistics, y is known as a response, and when x is a real vector, it is known as the covariate.

Types of Prediction Tasks

- Binary classification:

$$f(x) = y, y \in \{+1, -1\}$$

- Regression:

$$f(x) = y, y \in \mathbb{R}$$

- Multiclass classification: y is a category
- Ranking: y is a permutation
- Structured prediction: y is an object built from parts
- In the context of classification, f is called a classifier and y is called a label (or category, class, tag)

Supervised Learning

- The starting point of machine learning is data
- In supervised learning, the data provides both inputs and outputs
- Notation:
  - (x, y) specifies that y is the ground-truth output for x
  - $\mathcal{D}_{\text{train}} = [(x_1, y_1), \ldots, (x_n, y_n)]$ is the training data which forms a partial specification of the desired behavior of a predictor
- Learning is about taking $\mathcal{D}_{\text{train}}$ and producing a predictor f that approximately works for examples not seen in the training data

Feature Extraction

- Feature extractor: given input x, output a set of (feature name, feature value) pairs
- Feature extraction requires intuition about the task and also what machine learning algorithms are capable of, so it is a bit of an art
- Example: predict whether a string is an email address
  - possible features:
    - length
    - fraction of alphabetic characters
    - ends with `.com`
    - contains `@`

Feature Vector Notation

- Definition: For an input x its feature vector is:

  $$\phi(x) = [\phi_1(x), \ldots, \phi_d(x)]$$

  .

  where each component $\phi_j(x)$ for $j = 1, \ldots, d$, represents a feature.

- Think of $\phi(x) \in \mathbb{R}^d$ as a point in a high-dimensional space.

Weight Vector

- Weight vector: for each feature j, have a real number w_j representing the contribution of feature to prediction.
- In the context of binary classification with binary features ($\phi_j(x) \in \{0, 1\}$), the weights $w_j \in \mathbb{R}$ have an intuitive interpretation: if w_j is positive, then the presence of feature j ($\phi_j(x) = 1$) favors a positive classification (and the converse if w_j is negative.

Linear Predictors

- Given a feature vector $\phi(x) \in \mathbb{R}^d$ and a weight vector $\mathbf{w} \in \mathbb{R}^d$, the prediction score is

$$\mathbf{w} \cdot \phi(x) = \sum_{j=1}^{d} w_j \phi_j(x)$$

  That is, the inner product or weighted sum of features

Linear Predictors

- For binary classification we have

$$f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x)) = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \phi(x) > 0 \\ -1 & \text{if } \mathbf{w} \cdot \phi(x) < 0 \\ ? & \text{if } \mathbf{w} \cdot \phi(x) = 0 \end{cases}$$

- In general, binary classifier $f_{\mathbf{w}}$ dfines a hyperplane decision boundary with normal vector **w**.
  - In $\mathbb{R}^2$: hyperplane is a line.
  - In $\mathbb{R}^3$: hyperplane is a plane.

Learning Framework

- Given a linear predictor $f_{\mathbf{w}}$ based on a feature extractor ϕ, how do we learn **w** from the training data.
- Loss minimization is a framework that casts learning as an optimization problem.
- Note we can separate the problem into a model (optimization problem) and algorithm (optimization algorithm)

Loss Functions

- Definition: a loss function $\text{Loss}(x, y, \mathbf{w})$ quantifies how unhappy you would be if you used $\mathbf{w}$ to make a prediction on x when the correct output is y.
- The loss function is the object that we want to minimize.

Score and Margin

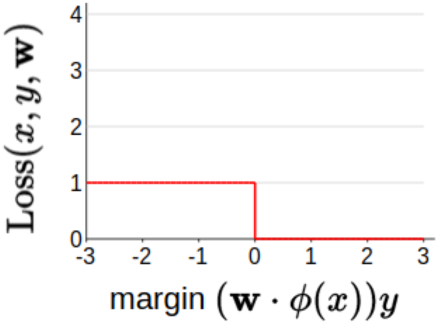
- Correct label: y
- Predicted label: $y' = f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x))$
- Definition: The score on an example (x, y) is $\mathbf{w} \cdot \phi(x)$, how confident we are in predicting $+1$
- Definition: The margin on an example (x, y) is $\mathbf{w} \cdot \phi(x))y$, how correct we are

Binary Classification

- Recall the binary classifier: $f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x))$
- Definition: zero-one loss

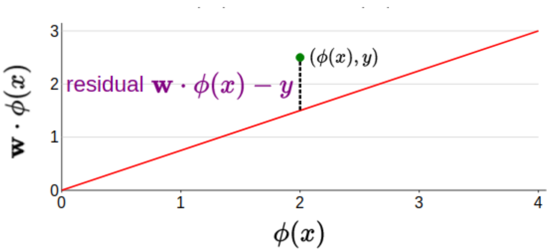
$$\begin{aligned} \text{Loss}_{0-1}(x, y, \mathbf{w} &= \mathbf{1}[f_{\mathbf{w}}(x) \neq y] \\ &= \mathbf{1}[\underbrace{\mathbf{w} \cdot (x))y}_{\text{margin}} \leq 0] \end{aligned}$$

Binary Classification



- $\text{Loss}_{0-1}(x, y, \mathbf{w}) = \mathbf{1}[(\mathbf{w} \cdot \phi(x))y \leq 0]$

Linear Regression



- Definition: The residual is $(\mathbf{w} \cdot \phi(x)) - y$, the amount by which the prediction $f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$ overshoots the target y.

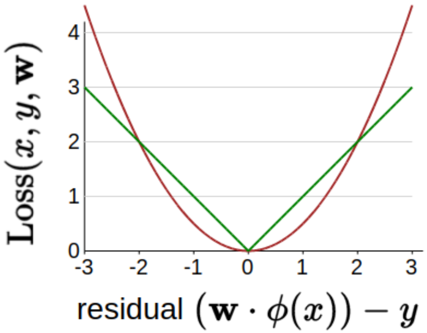
Linear Regression

- Definition: the squared loss is

$$\text{Loss}_{\text{squared}}(x, y, \mathbf{w}) = \underbrace{(f_{\mathbf{w}}(x) - y)^2}_{\text{residual}}$$

- Example:
  - $\mathbf{w} = [2, -1], \phi(x) = [2, 0], y = -1$
  - $\text{Loss}_{\text{squared}}(x, y, \mathbf{w}) = 25$

Regression Loss Functions



- $\text{Loss}_{\text{squared}}(x, y, \mathbf{w}) = (f_{\mathbf{w}}(x) - y)^2$
- $\text{Loss}_{\text{absdev}}(x, y, \mathbf{w}) = |f_{\mathbf{w}}(x) - y|$

Loss Minimization Framework

- Key Idea: minimize training loss

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$

$$\min_{\mathbf{w}\in\mathbb{R}^d} \text{TrainLoss}(\mathbf{w})$$

- We need to set **w** to make global tradeoffs – not every example can be happy.

How to Optimize?

- Definition: The gradient $\nabla_{\mathbf{w}}\text{TrainLoss}(\mathbf{w})$ is the direction that increases the loss the most.
- Algorithm: gradient descent
  - Initialize $\mathbf{w} = [0, \dots, 0]$
  - For $t = 1, \dots, T$:
    - $\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \underbrace{\nabla_{\mathbf{w}}\text{TrainLoss}(\mathbf{w})}_{\text{gradient}}$
- Gradient descent is an iterative optimization
- The step size η and number of iterations T are hyperparameters.

Least Squares Regression

- Objective function:

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} (\mathbf{w} \cdot \phi(x) - y)^2$$

- Gradient (use chain rule):

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} 2(\underbrace{\mathbf{w} \cdot \phi(x)}_{\text{prediction}} - \underbrace{y}_{\text{target}})\phi(x)$$

Gradient Descent is Slow

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$

- Gradient descent:

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

- Problem: each iteration requires going over all training examples – expensive when there is lots of data.

Stochastic Gradient Descent

$$\mathsf{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\mathsf{train}}|} \sum_{(x,y)\in\mathcal{D}_{\mathsf{train}}} \mathsf{Loss}(x, y, \mathbf{w})$$

- Gradient descent (GD):
  - $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathsf{TrainLoss}(\mathbf{w})$
- Stochastic gradient descent (SGD):
  - For each $(x, y) \in \mathcal{D}_{\mathsf{train}}$:
    - $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathsf{Loss}(x, y, \mathbf{w})$
- Key idea: stochastic updates; it is not about quality, but quantity

Step Size

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$$

- Question: what should η be?
    - Near zero: conservative, more stable
    - Becomes more aggressive, faster as η increases
- Strategies:
    - Constant: $\eta = 0.1$
    - Decreasing: $\eta = \frac{1}{\sqrt{\# \text{ updates made so far}}}$

Summary so Far

- Linear predictors:

$$f_{\mathbf{w}}(x) \text{based on score} \mathbf{w} \cdot \phi(x)$$

- Loss minimization: learning as optimization

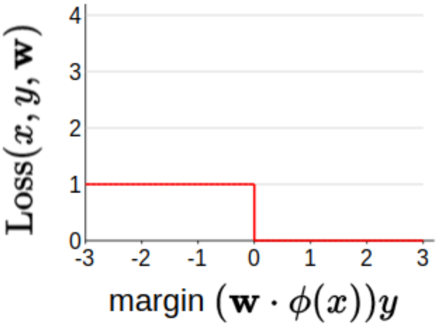
$$\min_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$$

- Stochastic gradient descent: optimization algorithm

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$$

Zero-one Loss

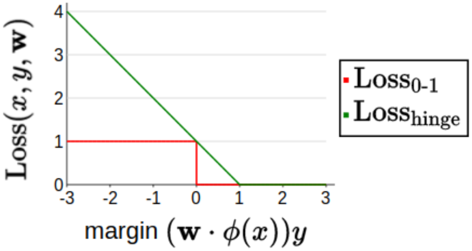
$$\text{Loss}_{0-1}(x, y, \mathbf{w}) = \mathbf{1}[(\mathbf{w} \cdot \phi(x))y \leq 0]$$



- Problems:
  - Gradient of Loss_{0-1} is 0 everywhere, SGD not applicable
  - Loss_{0-1} is insensitive to how badly the model messed up

Hinge Loss

$$\text{Loss}_{hinge}(x, y, \mathbf{w}) = \max\{[1 - (\mathbf{w} \cdot \phi(x))y, 0\}$$



- Intuition: hinge loss upper bounds 0-1 loss, have non-trivial gradient
- Try to increse margin if it is less than 1

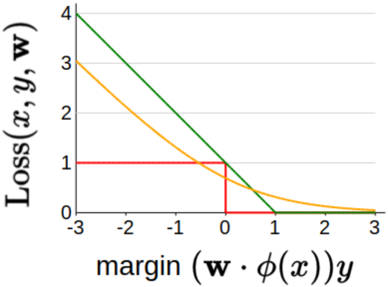
Hinge Loss

- Gradient of hinge loss

$$\nabla_{\mathbf{w}} \text{Loss}_{hinge}(x, y, \mathbf{w}) = \begin{cases} -\phi(x)y & \text{if } \mathbf{w} \cdot \phi(x)y < 1 \\ 0 & \text{if } \mathbf{w} \cdot \phi(x)y > 1 \end{cases}$$

Logistic Regression

$$\text{Loss}_{logistic}(x, y, \mathbf{w}) = \log\{1 + e^{-(\mathbf{w}\cdot\phi(x))y}$$



- Intuition: try to increase margin even when it already exceeds 1

Summary so Far

	Classification	Regression
Predictor $f_{\mathbf{w}}$	sign(score)	score
Relate to correct y	margin (score)	residual (score $- y$)
Loss functions	zero-one	squared
	hinge	absolute deviation
	logistic	
Algorithm	SGD	SGD

Components

- Score (drives prediction)

$$\mathbf{w} \cdot \phi(x)$$

- Learning chooses **w** via optimization
- Feature extraction specifies $\phi(x)$ based on domain knowledge

Feature Templates

- A feature template is a group of features all computed in a similar way
- A feature template allows us to define a set of related features
- A feature template can be written as a description with a blank
- Examples:
    - Length greater than ____
    - Pixel intensity of position ____, ____

Feature Representation

- Array representation
  - Good for dense features
- Dictionary representation
  - Good for sparse features

Hypothesis Class

- Predictor:

  $f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$ or $\text{sign}(\mathbf{w} \cdot \phi(x))$

- Definition: A hypothesis class is the set of possible predictors with a fixed $\phi(x)$ and varying $\mathbf{w}$:

$$\mathcal{F} = \{f_{\mathbf{w}} : \mathbf{w} \in \mathbb{R}^d\}$$

Example: Beyond Linear Functions

- Regression: $x \in \mathbb{R}, y \in \mathbb{R}$
- Linear functions: $\phi(x) = x$

$$\mathcal{F}_1 = \{x \mapsto w_1 x + w_2 x^2 : w_1 \in \mathbb{R}, w_2 = 0\}$$

- Quadratic functions: $\phi(x) = [x, x^2]$

$$\mathcal{F}_1 = \{x \mapsto w_1 x + w_2 x^2 : w_1 \in \mathbb{R}, w_2 \in \mathbb{R}\}$$

Linear in What?

- Prediction driven by score $\mathbf{w} \cdot \phi(x)$
  - Linear in $\mathbf{w}$: yes
  - Linear in $\phi(x)$: yes
  - Linear in x: no (x is not necessarily even a vector)
- Key idea: non-linearity
  - Predictors $f_{\mathbf{w}}$ can be expressive non-linear functions and decision boundaries of x
  - Score $\mathbf{w} \cdot \phi(x)$ is a linear function of $\mathbf{w}$, which permits efficient learning

Summary so Far

- Feature templates: organize related (spares) features
- Hypothesis class: defined by features (what is possible)
- Linear classifiers: can produce non-linear decision boundaries

Motivating Example

- Predicting car collision
- Input: position of two oncoming cars $x = [x_1, x_2]$
- Output: whether safe ($y = +1$) or collide ($y = -1$)
- True function: safe if cars sufficiently far
  $y = \text{sign}(|x_1 - x_2| - 1)
- Examples:

x	y
$[1, 3]$	$+1$
$[3, 1]$	$+1$
$[1, 0.5]$	-1

Decomposing the Problem

- Test if car 1 is far right of car 2

  $h_1 = \mathbf{1}[x_1 - x_2 \geq 1]$

- Test if car 2 is far right of car 1

  $h_2 = \mathbf{1}[x_2 - x_1 \geq 1]$

- Safe if at least one is true

  $y = \mathrm{sign}(h_1 + h_2)$

x	h_1	h_2	y
$[1, 3]$	0	1	$+1$
$[3, 1]$	1	0	$+1$
$[1, 0.5]$	0	0	-1

Learning Strategy

- Define: $\phi(x) = [1, x_1, x_2]$
- Intermediate hidden subproblems:

  $h_1 = \mathbf{1}[v_1 \cdot \phi(x) \geq 0]$

  $h_2 = \mathbf{1}[v_2 \cdot \phi(x) \geq 0]$
- Final prediction

  $f_{\mathbf{V},\mathbf{w}}(x) = \text{sign}(w_1 h_1 + w_2 h_2)$
- Key idea: joint learning – learn both hidden subproblems $V = (v_1, v_2)$ and combination weights $w = [w_1, w_2]$

Gradients

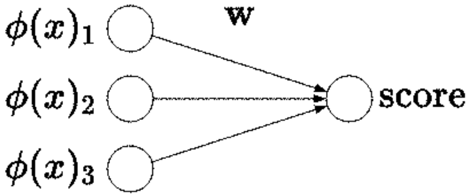
- Problem: gradient of h_1 with respect to v_1 is 0
- Definition: the logistic function maps $(-\infty, \infty)$ to $[0, 1]$:

  $\sigma(z) = (1 + e^{-z})^{-1}$
- Derivative

  $\sigma'(z) = \sigma(z)(1 - \sigma(z))$
- Solution: $h_1 = \sigma(v_1 \cdot \phi(x))$

Linear Functions

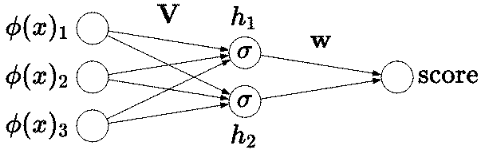
- Linear functions:



- Output: score $= \mathbf{w} \cdot \phi(x)$

Neural Networks

- Neural network (one hidden layer):



- Intermediate hidden units:

  $h_j = \sigma(v_j \cdot \phi(x))$

  $\sigma(z) = (1 + e^{-z})^{-1}$

- Output: score $= \mathbf{w} \cdot \mathbf{h}$

Training Neural Networks

- Optimization problem:

  $\min_{\mathbf{V},\mathbf{w}} \text{TrainLoss}(\mathbf{V}, \mathbf{w})$

- Goal compute gradient

  $\nabla_{\mathbf{V},\mathbf{w}} \text{TrainLoss}(\mathbf{V}, \mathbf{w})$

- Mathematically: grind through the chain rule
- Algorithm: backpropagation

Nearest Neighbors

- Algorithm: nearest neighbors
- Training: just store $\mathcal{D}_{\text{train}}$
- Predictor $f(x')$:
  - Find $(x, y) \in \mathcal{D}_{\text{train}}$ where $\|\phi(x) - \phi(x')\|$ is smallest
  - return y
- Idea: similar examples tend to have similar outputs

Summary of Learners

- Linear predictors: combine raw features
    - prediction is fast, easy to learn, weak use of features
- Neural networks: combine learned features
    - prediction is fast, hard to learn, powerful use of features
- Nearest neighbors: predict according to similar examples
    - prediction is slow, easy to learn, powerful use of features

Evaluation

- How good is a predictor f?
- Goal: minimize error on unseen future examples
- But, we do not have unseen examples
- So, make a test set $\mathcal{D}_{\text{test}}$ that 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?

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

where f^* is the target predictor and $g \in \mathcal{F}$ is the best predictor in the hypothesis class in the sense of minimizing test error

Effect of Hypothesis Class Size

- As the hypothesis class size increases
    - approximation error decreases
        - because taking the min over a larger set
    - estimation error increases
        - because harder to estimate something more complex
- Idea: minimize training error, but keep the hypothesis class small

Hyperparameters

- Definition: hyperparameters are properties of the learning algorithm, such as features, number of iterations step size, etc.
- How do we choose hyperparameters?
  - Choose hyperparameters to minimize $\mathcal{D}_{\text{train}}$ error? No, the solution would be to include all features and set the iterations to infinity
  - 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: cannot use the test set
- Solution: randomly take out, say, 10-50% of the training data and use it instead of the test set to estimate test error
- Definition: a validation set is taken out of the training data which acts as a surrogate for the test set.

Development Cycle

- Split data into training, validation, and test sets
- Look at data to get intuition
- Repeat:
  - implement feature / adjust hyperparameters
  - run learning algorithm
  - sanity check training and validation error rates
  - look at errors to brainstorm improvements
- Run on test set to get final error rates

Supervision

- Supervised learning
  - Prediction: $\mathcal{D}_{\text{train}}$ contains input-output pairs (x, y)
  - Fully-labeled data is very expensive to obtain
- Unsupervised learning
  - Clustering: $\mathcal{D}_{\text{train}}$ only contains inputs x
  - Unlabeled data is much cheaper to obtain
  - Key idea: data has lots of rich latent structures and we want to discover this structure automatically

Clustering

- Input: training set of input points

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

- Output: assignment of each point to a cluster

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

- Intuition: want similar points to be in the same cluster, and dissimilar points to be in different clusters

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:

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

  need to choose centroids μ and assignments z jointly

K-means Algorithm

Step 1 Goal: given centroids $\mu_i, \ldots, \mu_K$, assign each point to the best centroid.

- For each point $i = 1, \ldots, n$:
  - Assign i to cluster with closest centroid:
    - $z_i \leftarrow \min_{k=1,\ldots,K} \|\phi(x_i) - \mu_k\|^2$

Step 2 Goal: given cluster assignments $z_1, \ldots, z_n$, find the best centroids $\mu_1, \ldots, \mu_K$.

- For each cluster $k = 1, \ldots, K$:
  - Set μ_k to average of points assigned to cluster k:

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

- For each point $i = 1, \ldots, n$:
  - Assign i to cluster with closest centroid:
    - $z_i \leftarrow \min_{k=1,\ldots,K} \|\phi(x_i) - \mu_k\|^2$

K-means Algorithm

- Objective $\min_z \min_\mu \text{Loss}_{\text{kmeans}}(z, \mu)$
- Initialize $\mu_1, \ldots, \mu_K$ randomly
- For $t = 1, \ldots, T$:
  - Step 1: set assignments z given μ
  - Step 2: set centroids μ given z

Local Minima

- K-means is guaranteed to converge to a local minimum, but is not guaranteed to find the global minimum
- Solutions:
  - Run multiple times from different random initializations
  - Initialize with a heuristic (K-means++)

Summary

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