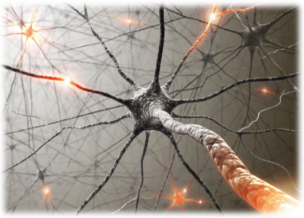
# Lecture 2.1: Machine learning I





# Course plan

Search problems

Markov decision processes

Adversarial games

Constraint satisfaction problems

Bayesian networks

**Reflex** **States** **Variables** **Logic**

"Low-level intelligence" "High-level intelligence"

**Machine learning**



# Course plan

Search problems

Markov decision processes

Adversarial games

Constraint satisfaction problems

Bayesian networks

**Reflex** **States** **Variables** **Logic**

"Low-level intelligence" "High-level intelligence"

**Machine learning**



# Roadmap

**Linear predictors**

Loss minimization



- We now embark on our journey into machine learning with the simplest yet most practical tool: **linear predictors**, which cover both classification and regression and are examples of reflex models.
- After getting some geometric intuition for linear predictors, we will turn to learning the weights of a linear predictor by formulating an optimization problem based on the **loss minimization** framework.

# Application: spam classification

Input: $x =$ email message

```
From:  robinjia@stanford.edu
Date:  June 27, 2019
Subject:  CS221 announcement

Hello students,
 I've attached the answers to homework 1...
```

```
From:  a9k62n@hotmail.com
Date:  June 27, 2019
Subject:  URGENT

Dear Sir or maDam:
 my friend left sum of 10m dollars...
```

Output: $y \in \{\text{spam}, \text{not-spam}\}$

Objective: obtain a **predictor** $f$

$$x \rightarrow \boxed{f} \rightarrow y$$

- First, some terminology. A **predictor** is a function $f$ that maps an **input** $x$ to an **output** $y$. 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 (e.g., email $\Rightarrow$ spam/not spam):

$$x \longrightarrow \boxed{f} \longrightarrow y \in \{+1, -1\}$$

Regression (e.g., location, year $\Rightarrow$ housing price):

$$x \longrightarrow \boxed{f} \longrightarrow y \in \mathbb{R}$$



- In the context of classification tasks, $f$ is called a **classifier** and $y$ is called a **label** (sometimes class, category, or tag). The key distinction between binary classification and regression is that the former has **discrete** outputs (e.g., "yes" or "no"), whereas the latter has **continuous** outputs.
- Note that the dichotomy of prediction tasks are not meant to be formal definitions, but rather to provide intuitions.
- For instance, binary classification could technically be seen as a regression problem if the labels are $-1$ and $+1$. And structured prediction generally refers to tasks where the possible set of outputs $y$ is huge (generally, exponential in the size of the input), but where each individual $y$ has some structure. For example, in machine translation, the output is a sequence of words.

# Types of prediction tasks

Multiclass classification: $y$ is a category

 $\longrightarrow \boxed{f} \longrightarrow$ cat

Ranking: $y$ is a permutation

$\boxed{1}\boxed{2}\boxed{3}\boxed{4} \longrightarrow \boxed{f} \longrightarrow$ 2 3 4 1

Structured prediction: $y$ is an object which is built from parts

*la casa blu* $\longrightarrow \boxed{f} \longrightarrow$ *the blue house*



# Question

Give an example of a prediction task (e.g., image $\Rightarrow$ face/not face) that is exciting to you.



# Data

Example: specifies that $y$ is the ground-truth output for $x$

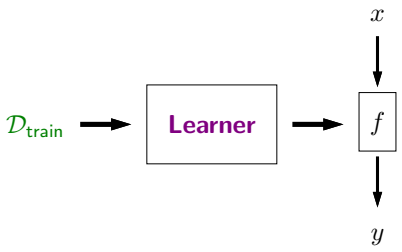
$$(x, y)$$

Training data: list of examples

$\mathcal{D}_{\text{train}} = [$
("...10m dollars...",+1),
("...CS221...", -1),
$]$

- The starting point of machine learning is the data.
- For now, we will focus on **supervised learning**, in which our data provides both inputs and outputs, in contrast to unsupervised learning, which only provides inputs.
- A (supervised) **example** (also called a data point or instance) is simply an input-output pair $(x, y)$, which specifies that $y$ is the ground-truth output for $x$.
- The **training data** $\mathcal{D}_{\text{train}}$ is a multiset of examples (repeats are allowed, but this is not important), which forms a partial specification of the desired behavior of a predictor.

# Framework





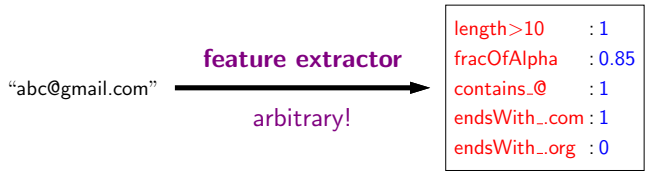

- **Learning** is about taking the training data $\mathcal{D}_{\text{train}}$ and producing a predictor $f$, which is a function that takes inputs $x$ and tries to map them to outputs $y = f(x)$. One thing to keep in mind is that we want the predictor to approximately work even for examples that we have not seen in $\mathcal{D}_{\text{train}}$. This problem of generalization, which we will discuss two lectures from now, forces us to design $f$ in a principled, mathematical way.
- We will first focus on examining what $f$ is, independent of how the learning works. Then we will come back to learning $f$ based on data.

# Feature extraction

Example task: predict $y$, whether a string $x$ is an email address

Question: what properties of $x$ **might be** relevant for predicting $y$?

Feature extractor: Given input $x$, output a set of (feature name, feature value) pairs.

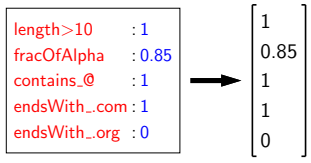





- We will consider predictors $f$ based on **feature extractors**. Feature extraction is a bit of an art that requires intuition about both the task and also what machine learning algorithms are capable of.
- The general principle is that features should represent properties of $x$ which **might be** relevant for predicting $y$. It is okay to add features which turn out to be irrelevant, since the learning algorithm can sort it out (though it might require more data to do so).

# Feature vector notation

Mathematically, feature vector doesn't need feature names:




**Definition: feature vector**

For an input $x$, its feature vector is:

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

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

- Each input $x$ represented by a **feature vector** $\phi(x)$, which is computed by the feature extractor $\phi$. When designing features, it is useful to think of the feature vector as being a map from strings (feature names) to doubles (feature values). But formally, the feature vector $\phi(x) \in \mathbb{R}^d$ is a real vector $\phi(x) = [\phi_1(x), \ldots, \phi_d(x)]$, where each component $\phi_j(x)$, for $j = 1, \ldots, d$, represents a feature.
- This vector-based representation allows us to think about feature vectors as a point in a (high-dimensional) vector space, which will later be useful for getting some geometric intuition.

# Weight vector

Weight vector: for each feature $j$, have real number $w_j$ representing contribution of feature to prediction

| | |
|---|---|
| length>10 | :-1.2 |
| fracOfAlpha | :0.6 |
| contains_@ | :3 |
| endsWith_.com | :2.2 |
| endsWith_.org | :1.4 |
| ... | |



- So far, we have defined a feature extractor $\phi$ that maps each input $x$ to the feature vector $\phi(x)$. A **weight vector** $\mathbf{w} = [w_1, \ldots, w_d]$ (also called a parameter vector or weights) specifies the contributions of each feature vector to the 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. Conversely, if $w_j$ is negative, then the presence of feature $j$ favors a negative classification.
- Note that while the feature vector depends on the input $x$, the weight vector does not. This is because we want a single predictor (specified by the weight vector) that works on any input.

# Linear predictors

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

| | |
|---|---|
| length>10 | :-1.2 |
| fracOfAlpha | :0.6 |
| contains_@ | :3 |
| endsWith_.com | :2.2 |
| endsWith_.org | :1.4 |

Feature vector $\phi(x) \in \mathbb{R}^d$

| | |
|---|---|
| length>10 | :1 |
| fracOfAlpha | :0.85 |
| contains_@ | :1 |
| endsWith_.com | :1 |
| endsWith_.org | :0 |

**Score**: weighted combination of features

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

Example: $-1.2(1) + 0.6(0.85) + 3(1) + 2.2(1) + 1.4(0) = 4.51$



- Given a feature vector $\phi(x)$ and a weight vector $\mathbf{w}$, we define the prediction **score** to be their inner product. The score intuitively represents the degree to which the classification is positive or negative.
- The predictor is linear because the score is a linear function of $\mathbf{w}$ (more on linearity in the next lecture).
- Again, in the context of binary classification with binary features, the score aggregates the contribution of each feature, weighted appropriately. We can think of each feature present as voting on the classification.

# Linear predictors

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

Feature vector $\phi(x) \in \mathbb{R}^d$

For binary classification:

**Definition: (binary) linear classifier**

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

- We now have gathered enough intuition that we can formally define the predictor $f$. For each weight vector $\mathbf{w}$, we write $f_{\mathbf{w}}$ to denote the predictor that depends on $\mathbf{w}$ and takes the sign of the score.
- For the next few slides, we will focus on the case of binary classification. Recall that in this setting, we call the predictor a (binary) classifier.
- The case of $\mathbf{w} \cdot \phi(x) = 0$ is a boundary case that isn't so important. We can just predict $+1$ arbitrarily as a matter of convention.

# Geometric intuition

A binary classifier $f_{\mathbf{w}}$ defines a hyperplane with normal vector $\mathbf{w}$.

($\mathbb{R}^2 \implies$ hyperplane is a line; $\mathbb{R}^3 \implies$ hyperplane is a plane)

Example:

$\mathbf{w} = [2, -1]$

$\phi(x) \in \{[2, 0], [0, 2], [2, 4]\}$

[whiteboard]



- So far, we have talked about linear predictors as weighted combinations of features. We can get a bit more insight by studying the **geometry** of the problem.
- Let's visualize the predictor $f_{\mathbf{w}}$ by looking at which points it classifies positive. Specifically, we can draw a ray from the origin to $\mathbf{w}$ (in two dimensions).
- Points which form an acute angle with $\mathbf{w}$ are classified as positive (dot product is positive), and points that form an obtuse angle with $\mathbf{w}$ are classified as negative. Points which are orthogonal $\{z \in \mathbb{R}^d : \mathbf{w} \cdot z = 0\}$ constitute the **decision boundary**.
- By changing $\mathbf{w}$, we change the predictor $f_{\mathbf{w}}$ and thus the decision boundary as well.

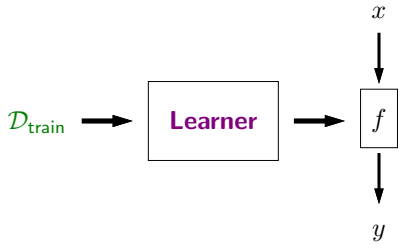
# Roadmap

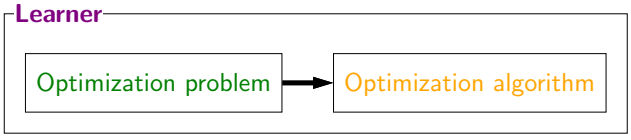
Linear predictors

**Loss minimization**



# Framework






# Optimization

Discrete optimization: a discrete object

$$\min_{p \in \text{Paths}} \text{Distance}(p)$$

**Algorithmic** tool: dynamic programming

Continuous optimization: a vector of real numbers

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

**Algorithmic** tool: gradient descent (next class)

- So far we have talked about linear predictors $f_{\mathbf{w}}$ which are based on a feature extractor $\phi$ and a weight vector $\mathbf{w}$. Now we turn to the problem of estimating (also known as fitting or learning) $\mathbf{w}$ from training data.
- The **loss minimization** framework is to cast learning as an optimization problem.
- What do I mean by an optimization problem? There are two main types of we'll consider: discrete optimization problems (mostly for inference) and continuous optimization problems (mostly for learning). We already saw discrete optimization in the first lecture.
- As we'll see in this class, it's often helpful to separate your problem into a model (optimization problem) and an algorithm (optimization algorithm). Today we'll set up an optimization problem for learning; next week, we will learn about gradient descent, one of the most common continuous optimziation algorithms.

# Loss functions

**Definition: loss function**

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 $y$. It is the object we want to minimize.



# Score and margin

Correct label: $y$

Predicted label: $y' = f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x))$

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

**Definition: score**

The score on an example $(x, y)$ is $\mathbf{w} \cdot \phi(x)$, how **confident** we are in predicting $+1$.

**Definition: margin**

The margin on an example $(x, y)$ is $(\mathbf{w} \cdot \phi(x))y$, how **correct** we are.



- Before we talk about what loss functions look like and how to learn $\mathbf{w}$, we introduce another important concept, the notion of a **margin**. Suppose the correct label is $y \in \{-1, +1\}$. The margin of an input $x$ is $\mathbf{w} \cdot \phi(x)y$, which measures how correct the prediction that $\mathbf{w}$ makes is. The larger the margin, the better, and non-positive margins correspond to classification errors.
- Note that if we look at the actual prediction $f_{\mathbf{w}}(x)$, we can only ascertain whether the prediction was right or not. By looking at the score and the margin, we can get a more nuanced view onto the behavior of the classifier.
- Geometrically, if $\|\mathbf{w}\| = 1$, then the margin of an input $x$ is exactly the distance from its feature vector $\phi(x)$ to the **decision boundary**.

# Question

When does a binary classifier err on an example?

- margin less than 0
- margin greater than 0
- score less than 0
- score greater than 0



# Binary classification

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

Recall the binary classifier:

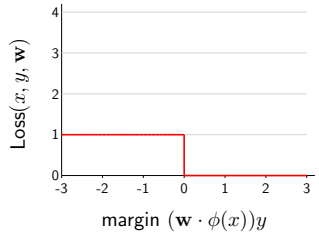
$$f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x))$$

**Definition: zero-one loss**

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

- Now let us define our first loss, function, the **zero-one loss**. This corresponds exactly to our familiar notion of whether our predictor made a mistake or not. We can also write the loss in terms of the margin.

## Binary classification



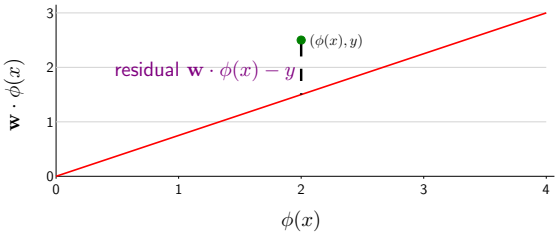

$$\text{Loss}_{0\text{-}1}(x, y, \mathbf{w}) = \mathbf{1}[(\mathbf{w} \cdot \phi(x))y \leq 0]$$



- We can plot the loss as a function of the margin. From the graph, it is clear that the loss is 1 when the margin is negative and 0 when it is positive.

## Linear regression

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




**Definition: residual**

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



- Now let's turn for a moment to regression, where the output $y$ is a real number rather than $\{-1, +1\}$. Here, the **zero-one loss** doesn't make sense, because it's unlikely that we're going to predict $y$ exactly.
- Let's instead define the **residual** to measure how close the prediction $f_{\mathbf{w}}(x)$ is to the correct $y$. The residual will play the analogous role of the margin for classification and will let us craft an appropriate loss function.

## Linear regression

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

**Definition: squared loss**

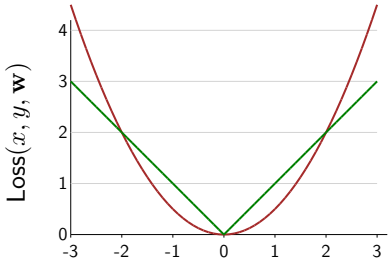
$$\text{Loss}_{\text{squared}}(x, y, \mathbf{w}) = (\underbrace{f_{\mathbf{w}}(x) - y}_{\text{residual}})^2$$

Example:

$$\mathbf{w} = [2, -1], \phi(x) = [2, 0], y = -1$$

$$\text{Loss}_{\text{squared}}(x, y, \mathbf{w}) = 25$$

# Regression loss functions




$$\text{Loss}_{\text{squared}}(x, y, \mathbf{w}) = (\mathbf{w} \cdot \phi(x) - y)^2$$

$$\text{Loss}_{\text{absdev}}(x, y, \mathbf{w}) = |\mathbf{w} \cdot \phi(x) - y|$$



- A popular and convenient loss function to use in linear regression is the **squared loss**, which penalizes the residual of the prediction quadratically. If the predictor is off by a residual of $10$, then the loss will be $100$.
- An alternative to the squared loss is the **absolute deviation loss**, which simply takes the absolute value of the residual.

# Loss minimization framework

So far: one example, $\text{Loss}(x, y, \mathbf{w})$ is easy to minimize.

**Key idea: minimize training loss**

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

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

Key: need to set $\mathbf{w}$ to make global tradeoffs — not every example can be happy.



- Note that on one example, both the squared and absolute deviation loss functions have the same minimum, so we cannot really appreciate the differences here. However, we are learning $\mathbf{w}$ based on a whole training set $\mathcal{D}_{\text{train}}$, not just one example. We typically minimize the **training loss** (also known as the training error or empirical risk), which is the average loss over all the training examples.
- Importantly, such an optimization problem requires making tradeoffs across all the examples (in general, we won't be able to set $\mathbf{w}$ to a single value that makes every example have low loss).

# Which regression loss to use?

Example: $\mathcal{D}_{\text{train}} = \{(1, 0), (1, 2), (1, 1000)\}$, $\phi(x) = x$

For least squares ($L_2$) regression:

$$\text{Loss}_{\text{squared}}(x, y, \mathbf{w}) = (\mathbf{w} \cdot \phi(x) - y)^2$$

- $\mathbf{w}$ that minimizes training loss is **mean** $y$
- Mean: tries to accommodate every example, popular

For least absolute deviation ($L_1$) regression:

$$\text{Loss}_{\text{absdev}}(x, y, \mathbf{w}) = |\mathbf{w} \cdot \phi(x) - y|$$

- $\mathbf{w}$ that minimizes training loss is **median** $y$
- Median: more robust to outliers



- Now the question of which loss we should use becomes more interesting.
- For example, consider the case where all the inputs are $\phi(x) = 1$. Essentially the problem becomes one of predicting a single value $y^*$ which is the least offensive towards all the examples.
- If our loss function is the squared loss, then the optimal value is the mean $y^* = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} y$. If our loss function is the absolute deviation loss, then the optimal value is the median.
- The median is more robust to outliers: you can move the furthest point arbitrarily farther out without affecting the median. This makes sense given that the squared loss penalizes large residuals a lot more.
- In summary, this is an example of where the choice of the loss function has a qualitative impact on the weights learned, and we can study these differences in terms of the objective function without thinking about optimization algorithms.

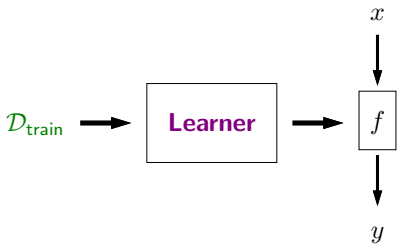
# Summary

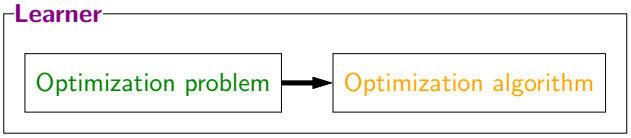
$$\underbrace{\mathbf{w} \cdot \phi(x)}_{\text{score}}$$

| | Classification | Linear regression |
|---|---|---|
| Predictor $f_{\mathbf{w}}$ | sign(score) | score |
| Relate to correct $y$ | margin (score $y$) | residual (score $- y$) |
| Loss functions | zero-one | squared<br>absolute deviation |



# Framework






# Next class

Loss minimization:

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

Use an optimization algorithm (stochastic gradient descent) to find $w$.

Linear predictors:

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

Which feature vector $\phi(x)$ to use?