Artificial Intelligence

CS 165A

Oct 8, 2020

Hello!

Instructor: Prof. Yu-Xiang Wang

Today

→ Supervised learning

→ Continuous optimization

Anonymous student feedback

- "I really like the interactivity in the chat. I would like more, personally."
- "more explanation of notation during lecture"
- "At the end of Lecture 2 when defining generalization error it said "Gen(H) := sup(...)" I'm not sure what sup means."

$$\text{Gen}(H) = \sup_{h \in H} \left| \text{train}(h) - E[\text{err}(h)] \right| \quad (\text{max})$$

Link to submit feedback: https://forms.gle/Eenu1aw3SzBxGcTaA

Recap: Last lecture

- Machine learning overview
- Supervised learning: Spam filtering as an example
  - Features, feature extraction
  - Models, hypothesis class
  - Choosing an appropriate hypothesis class
- Performance measure

Recap: Building a classifier agent

Modeling

- Feature engineering
- Specify a family of classifiers

Inference

Deployment to email client

Learning

Learning the best performing classifier

Recap: Mathematically defining the supervised learning problem

- Feature space: $\mathcal{X} = \mathbb{R}^d$
- Label space: $\mathcal{Y} = \{0, 1\} = \{\text{non-spam}, \text{spam}\}$
- A classifier (hypothesis): $h : \mathcal{X} \rightarrow \mathcal{Y}$
- A hypothesis class: $\mathcal{H}$
- Data: $(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$
- Learning task: Find $h \in \mathcal{H}$ that "works well".

Recap: The "free parameters" of the two hypothesis classes we learned

- Decision trees

- Linear classifiers

Recap: The “free parameters” of the two hypothesis classes we learned

- Decision trees
  - “Which feature to use when branching?”
  - “The threshold parameter”
  - “Which label to assign at the leaf node”
  - …
- Linear classifiers

Recap: The "free parameters" of the two hypothesis classes we learned

- Decision trees
  - "Which feature to use when branching?"
  - "The threshold parameter"
  - "Which label to assign at the leaf node"
  - …
- Linear classifiers
  - "Coefficient vector of the score function"
  - a (d+1) dimensional vector.

Recap: What do we mean by “working well”?

- Recall: the PEAS specification of a task environment
  - **P**erformance measure, **E**nvironment, **A**ctuators, **S**ensors.

- What’s the “Performance measure” for a classifier agent?

Recap: What do we mean by "working well"?

- Recall: the PEAS specification of a task environment
  - **P**erformance measure, **E**nvironment, **A**ctuators, **S**ensors.

- What's the "Performance measure" for a classifier agent?

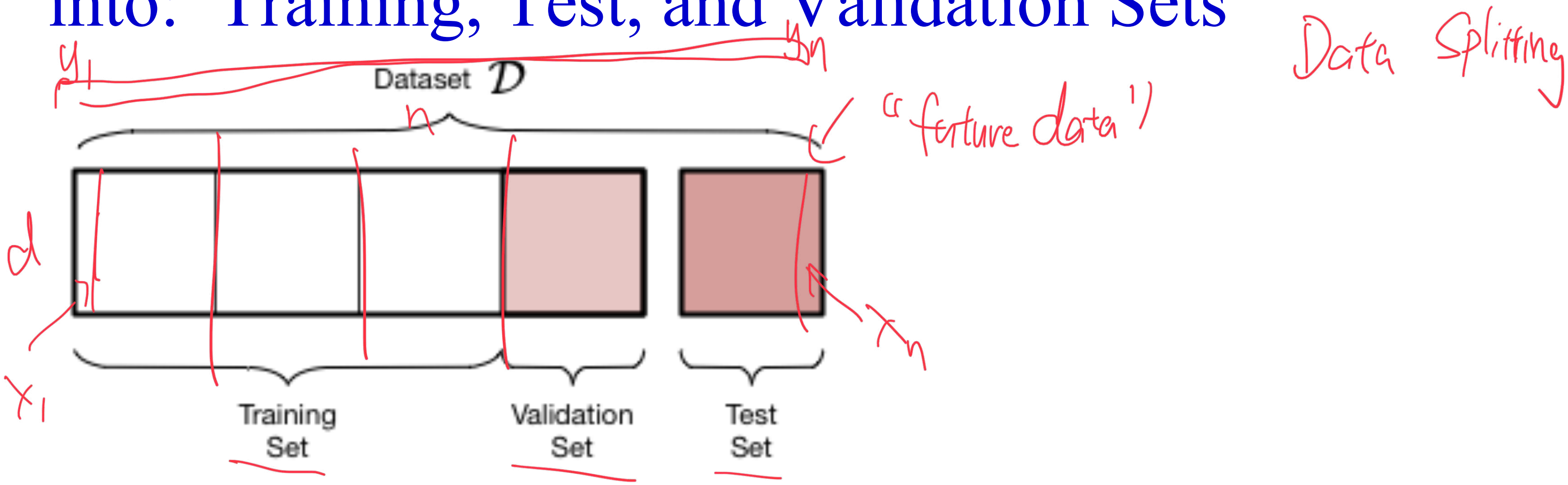
  - Really the **average error rate** on **new** data points.
  - But all we have is a training dataset.
  - Training error: (empirical) error rate on the training data.

  - When does the learned classifier ***generalize***? SLT
  - How to know it if it does not?

Plan for today

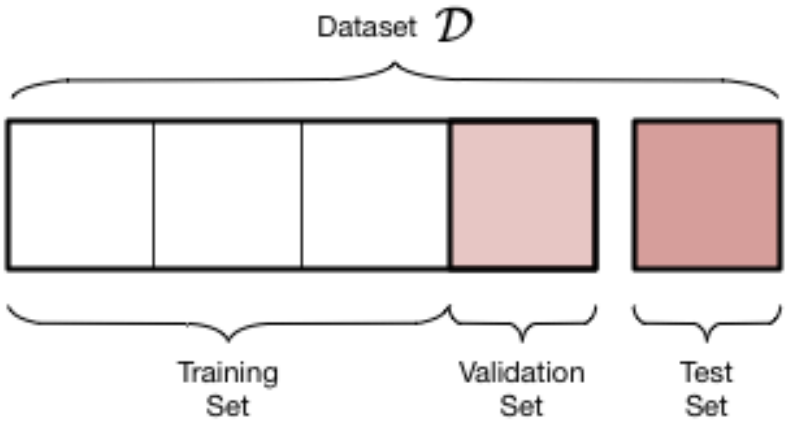
- Preventing overfitting in practice
- More caveats about ML agents
- Continuous optimization
  - How to learn a linear classifier?

Empirically measuring the test error by splitting the data into: Training, Test, and Validation Sets



Read more in Section 18.4 in the AIMA textbook.

Empirically measuring the test error by splitting the data into: Training, Test, and Validation Sets

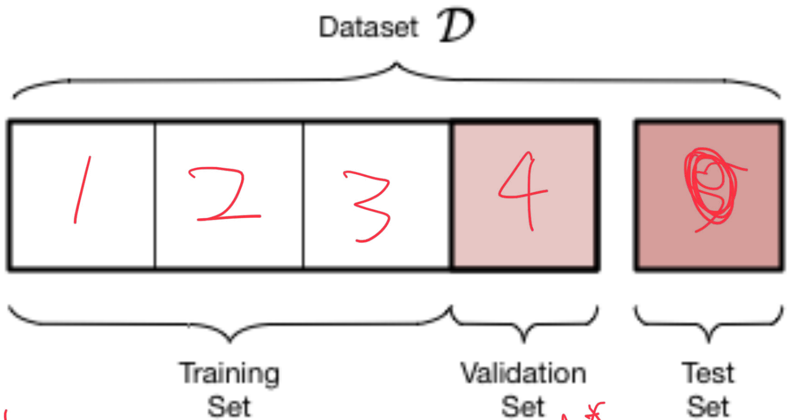


Validation set is used for model-selection:
- choosing decision tree vs. linear classifier
- Select features, tune hyperparameters

Test set is used only once to report the final results.

Read more in Section 18.4 in the AIMA textbook.

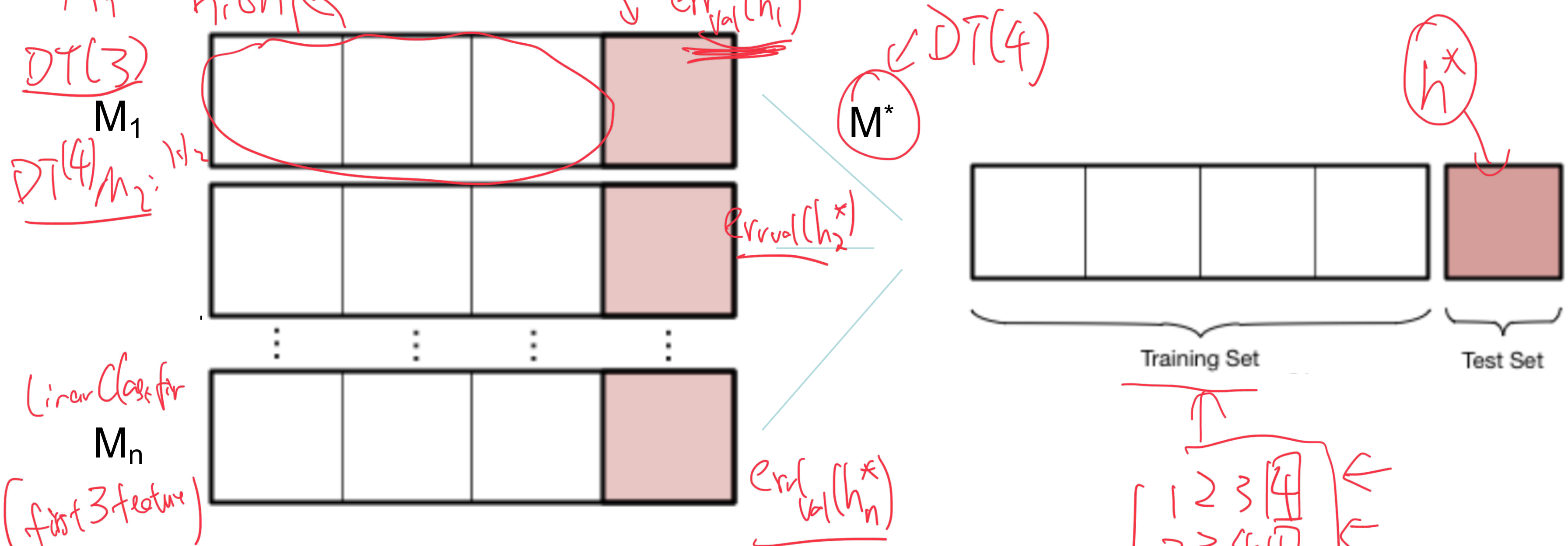
Empirically measuring the test error by splitting the data into: Training, Test, and Validation Sets



Validation set is used for model-selection:

- choosing decision tree vs. linear classifier
- Select features, tune hyperparameters

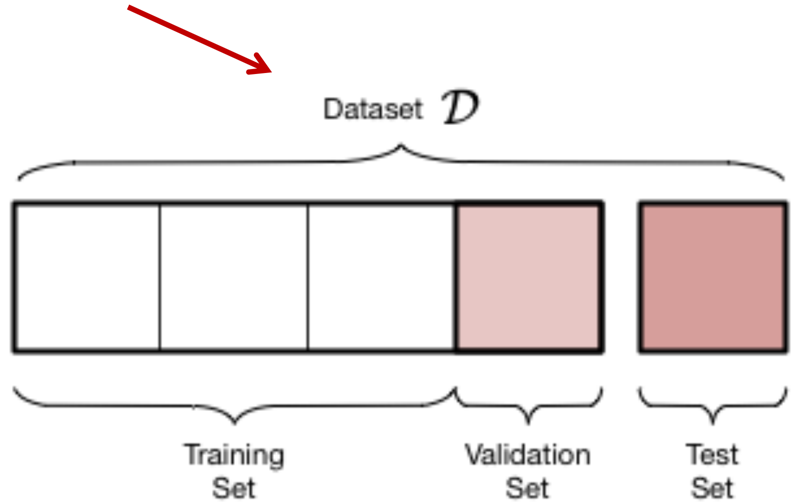
Test set is used only once to report the final results.

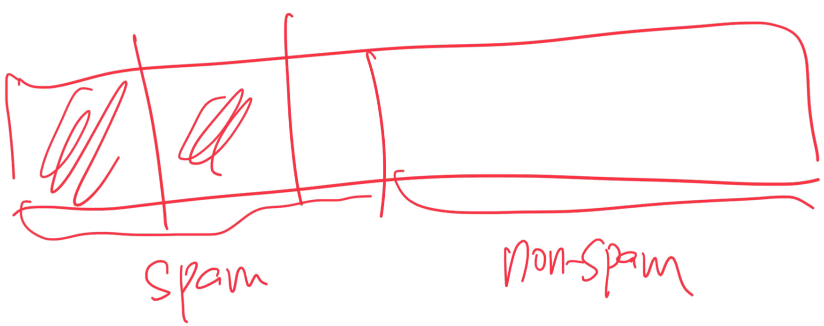


Read more in Section 18.4 in the AIMA textbook.

A practical note: Always shuffling the data before splitting them into Training-Validation-test set

data shall be randomly shuffled before splitting





Case study: Biotech startup (3 min discussion)

- Problem (true story, according to Alex Smola)
  - Biotech startup wants to detect prostate cancer
  - Easy to get blood samples from sick patients
  - Hard to get blood samples from healthy ones.

- Solution?
  - Get blood samples from male university students
  - Use them as healthy reference.
  - Classifier gets 100% accuracy.

1. training set not Robust representative

- What is wrong?

2. age

The problem of distribution shift

Training data

The problem of distribution shift

Training data

Test data received during:
Prediction / inference /Deployment

The problem of distribution shift

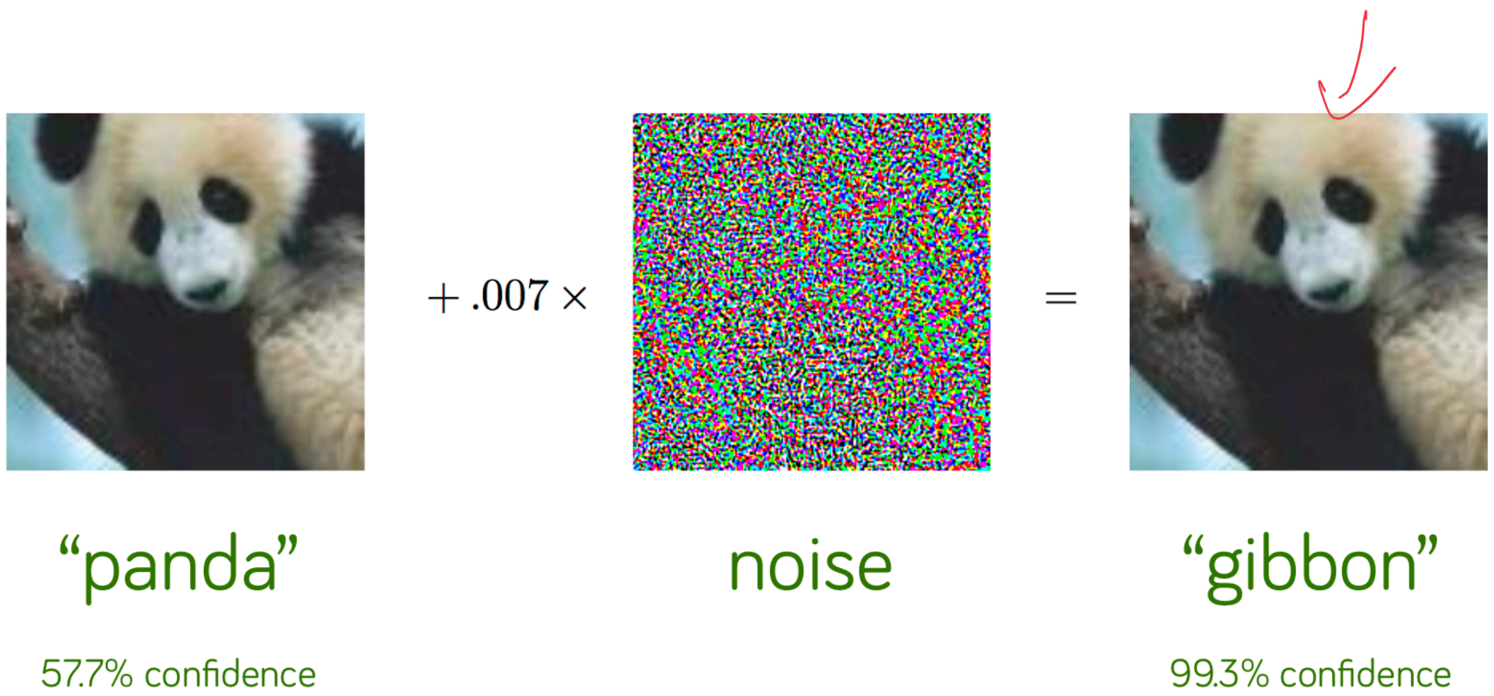
Training data

Test data received during:

Prediction / inference /Deployment

** Machine learning is only "**guaranteed to work**" when the training data are **drawn i.i.d.** from the **same distribution** as the new data that we will receive in the "inference" phase.

“Adversarial Examples” are consequences of distribution-shift



(Goodfellow et al., 2015)

Quick checkpoint

- Feature extraction
- Specifying a "hypothesis class"
  - indexed by "free parameters"
- Learning == search for the best hypothesis
- Ideally, we want to minimize "test error"
  - but all we have access to is the training data.
  - minimize "training error" (Statistical learning theory says that this is OK)
  - We have a practical way --- data-splitting --- to evaluate a classifier

Remainder of this lecture

Modeling

- Feature engineering
- Specify a family of classifiers

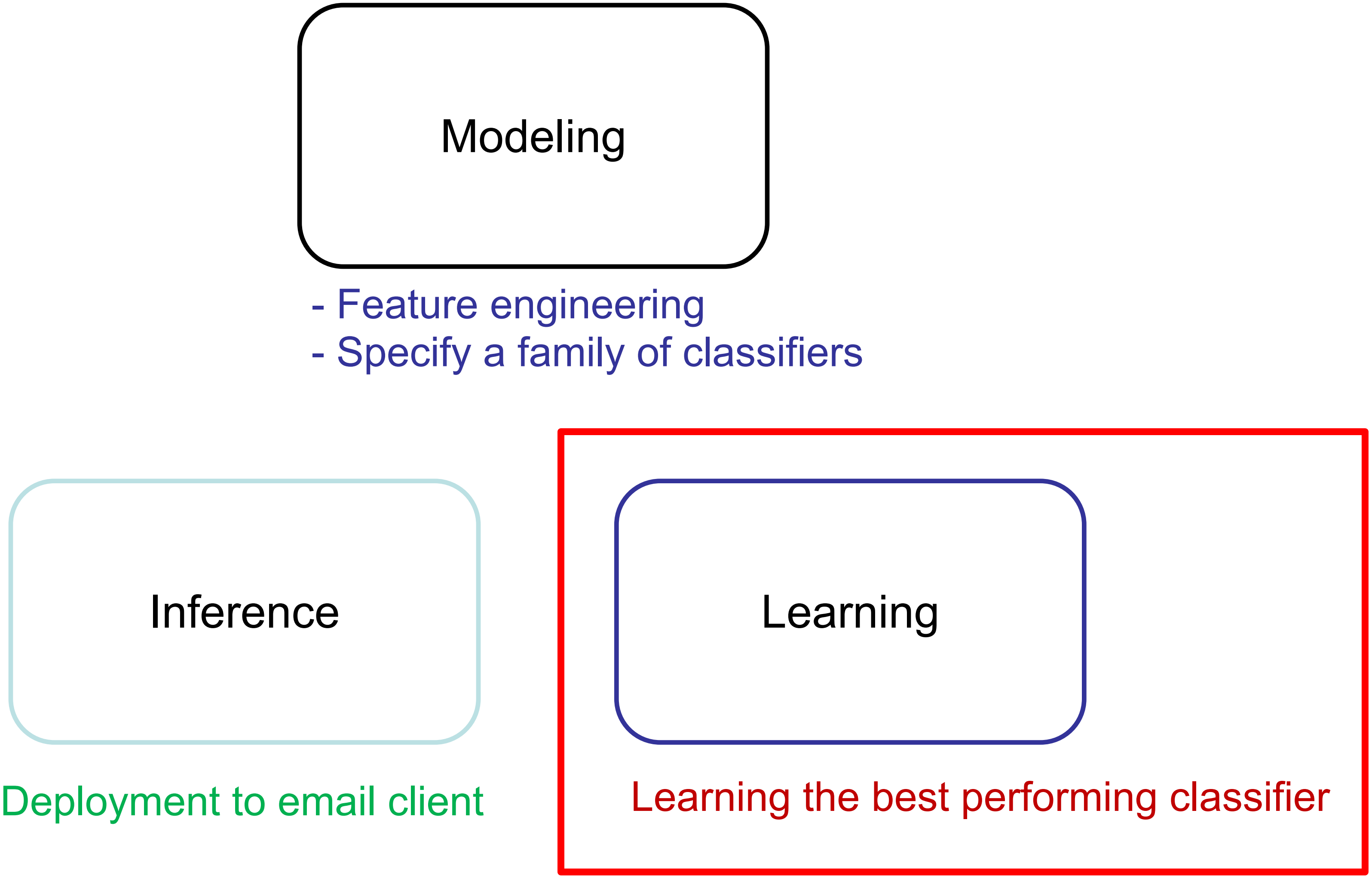
Inference

Deployment to email client

Learning

Learning the best performing classifier

Remainder of this lecture



Example: Linear classifiers

- Score(x) = $w_0 + w_1$ * 1(hyperlinks) + w_2 * 1(contact list) + w_3 * misspelling + w_4 * length

- A linear classifier: h(x) = 1 if Score(x) > 0 and 0 otherwise.

  Spam non-spam

- Reparameterization
  - If we redefine $\mathcal{Y} = \{-1, 1\}$

    {0, 1}
  - A compact representation:

$$h(x) = \text{sign}(w^T[1; x])$$

Example: Linear classifiers

- Score(x) = $w_0 + w_1$ * 1(hyperlinks) + w_2 * 1(contact list) + w_3 * misspelling + w_4 * length

- A linear classifier: h(x) = 1 if Score(x) > 0 and 0 otherwise.

- Reparameterization
  - If we redefine $\mathcal{Y} = \{-1, 1\}$
  - A compact representation:

$$h(x) = \text{sign}(w^T [1; x])$$

(from here onwards, we will redefine the feature vector x to be [1;x] w.l.o.g., for the interest of simplifying the notation)

How do we learn a linear classifier?

- Linear classifier:

$$h_w(x) = \text{sign}(w^T x)$$

- Training data:

$$(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$$

- Solving the following optimization problem:

$$\min_{w \in \mathbb{R}^d} \text{Error}(w) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(h_w(x_i) \neq y_i)$$

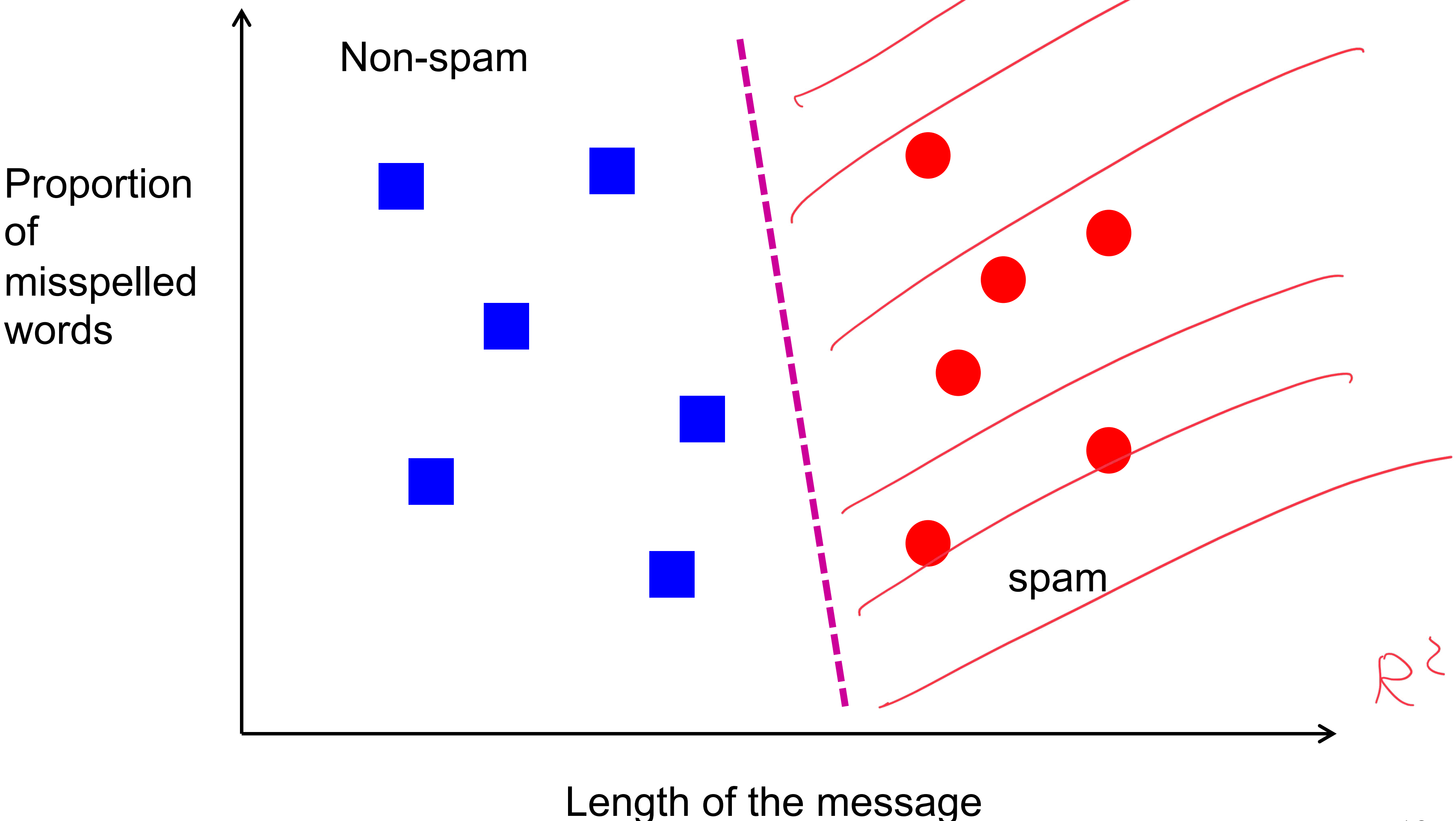
- Learning: Find the linear classifier that makes **the smallest number of mistakes** on the training data.

Geometric view: Linear classifier are "half-spaces"!

the set of input x when the classifier w will out-put "1"

$\{ x \mid w_0 + w_1 * x1 + w_2 * x2 + w_3 * x3 + w_4 * x4 > 0\}$

The set of all "emails" that will be classified as "Spams".



In the case when the training data is linearly separable, there is a polynomial time algorithm.

- Why?

(Also, you'll see the perceptron algorithm in CS165B.)

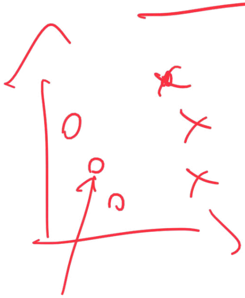
In the case when the training data is linearly separable, there is a polynomial time algorithm.

- Why?
  - Easiest way to see it is that it is a **linear program**.

(Also, you'll see the perceptron algorithm in CS165B.)

In the case when the training data is linearly separable, there is a polynomial time algorithm.

- Why?
  - Easiest way to see it is that it is a **linear program**.

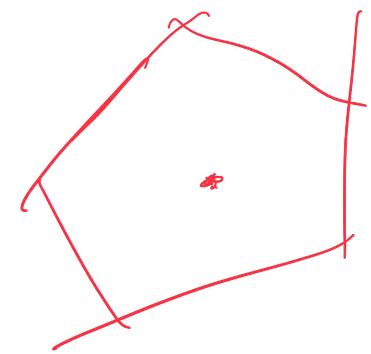


find $w \in \mathbb{R}^d$

subject to:

$$w^T x_i > 0 \ \forall i \in \{1, 2, ..., n\} \text{ s.t.} y_i = 1$$

$$w^T x_i \leq 0 \ \forall i \in \{1, 2, ..., n\} \text{ s.t.} y_i = -1$$



(Also, you'll see the perceptron algorithm in CS165B.)

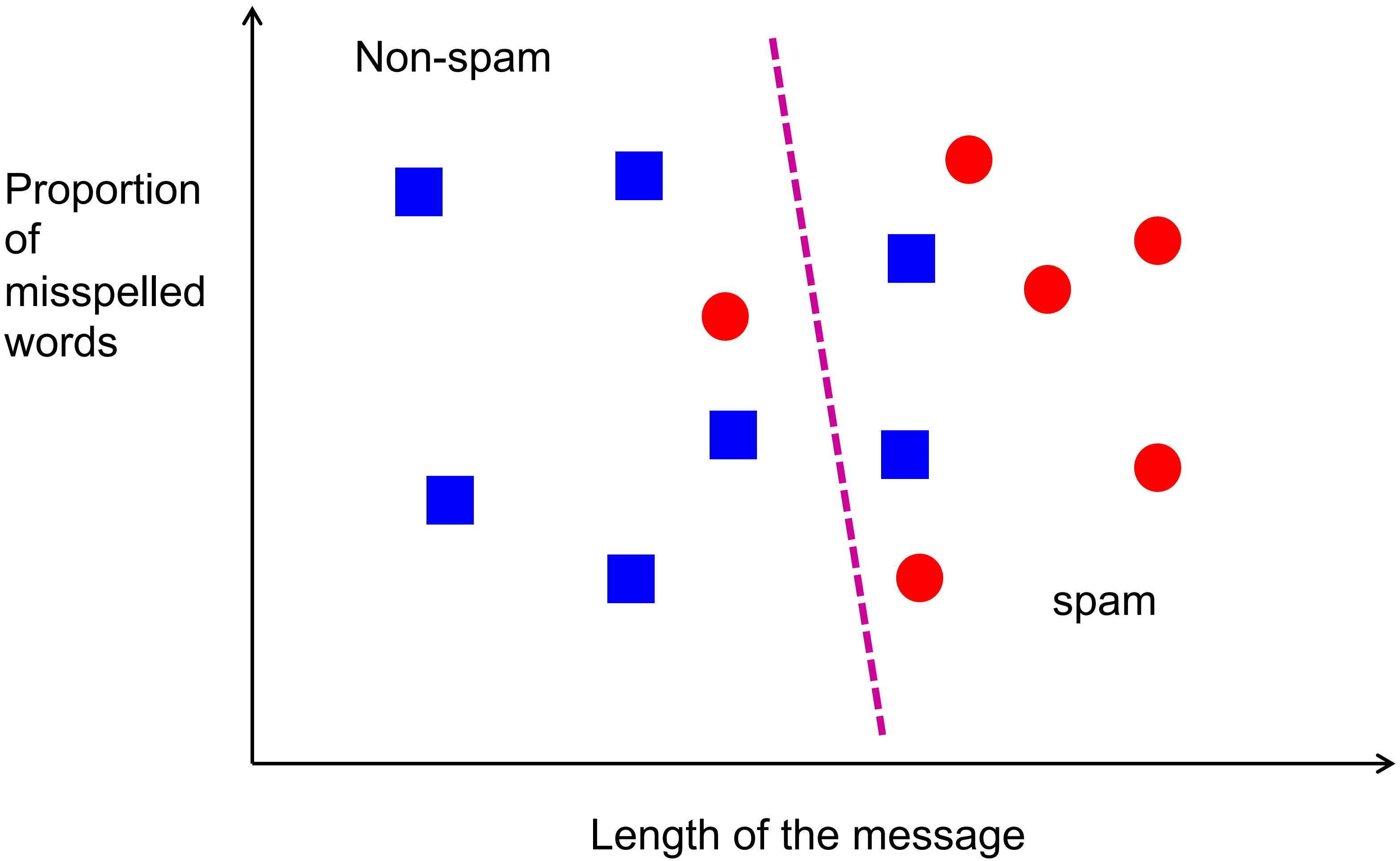
In the case when the training data is linearly separable, there is a polynomial time algorithm.

- Why?
  - Easiest way to see it is that it is a **linear program**.
  - Polynomial time algorithm exists for all LPs. (taught in CS 130a/b)

$$
\begin{aligned}
&\text{find } w \in \mathbb{R}^d \\
&\text{subject to:} \\
&\quad w^T x_i > 0 \; \forall i \in \{1, 2, ..., n\} \text{ s.t.} y_i = 1 \\
&\quad w^T x_i \leq 0 \; \forall i \in \{1, 2, ..., n\} \text{ s.t.} y_i = -1
\end{aligned}
$$

(Also, you'll see the perceptron algorithm in CS165B.)

Best linear separator in general (linearly non-separable cases) is NP-hard.



What do we do? General idea of bounded rationality.

— Error rate

- Design rational agent = maximize some utility function
- Sometimes the utility function is too difficult to maximize
- Maybe we can maximize an approximation to the utility function is computationally easier
- So we can focus on modelling (e.g., better features, better hypothesis class)

Just "relax": relaxing a hard problem into an easier one

$$\min_{w \in \mathbb{R}^d} \text{Error}(w) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(\text{sign}(w^T x_i) \neq y_i)$$

$$\Downarrow$$

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \ell(w^T x_i, y_i).$$

Loss functions and surrogate losses

Loss functions and surrogate losses

- 0-1 loss: $\mathbf{1}(h_w(x) \neq y) = \begin{cases} 1 & \text{if } h_w(x) \neq y \\ 0 & \text{otherwise} \end{cases}$

Loss functions and surrogate losses

- 0-1 loss: $\mathbf{1}(h_w(x) \neq y) = \mathbf{1}(\mathrm{sign}(S_w(x)) \neq y)$

Loss functions and surrogate losses

- 0-1 loss: $\mathbf{1}(h_w(x) \neq y) = \mathbf{1}(\text{sign}(S_w(x)) \neq y)$
- Square loss: $(y - S_w(x))^2$

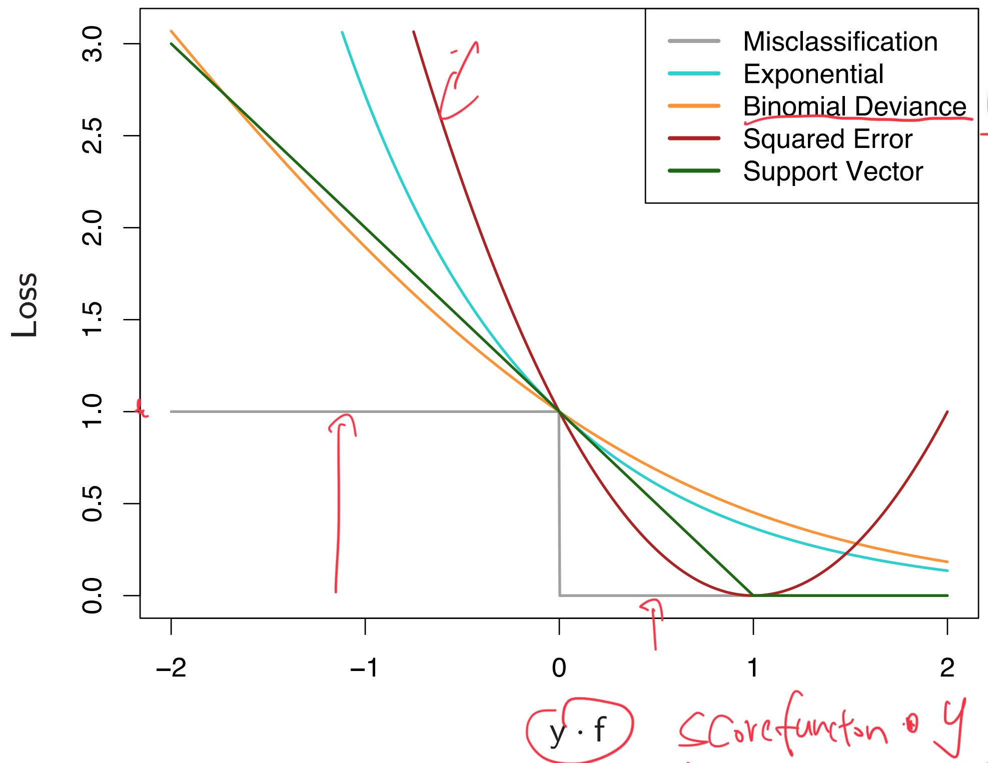
Loss functions and surrogate losses

- 0-1 loss: $\mathbf{1}(h_w(x) \neq y) = \mathbf{1}(\text{sign}(S_w(x)) \neq y)$
- Square loss: $(y - S_w(x))^2$
- Logistic loss: $\log_2(1 + \exp(-y \cdot S_w(x)))$

Loss functions and surrogate losses

- 0-1 loss: $\mathbf{1}(h_w(x) \neq y) = \mathbf{1}(\text{sign}(S_w(x)) \neq y)$
- Square loss: $(y - S_w(x))^2$
- Logistic loss: $\log_2(1 + \exp(-y \cdot S_w(x)))$
- Hinge loss: $\max(0, 1 - y \cdot S_w(x))$

Visualizing the relaxed "surrogate loss" functions



** "Binomial deviance" is the "logistic loss" from the previous slide.

FIGURE 10.4. *Loss functions for two-class classification. The response is* $y = \pm 1$*; the prediction is* f*, with class prediction* $\text{sign}(f)$*. The losses are misclassification:* $I(\text{sign}(f) \neq y)$*; exponential:* $\exp(-yf)$*; binomial deviance:* $\log(1 + \exp(-2yf))$*; squared error:* $(y - f)^2$*; and support vector:* $(1 - yf)_+$ *(see Section 12.3). Each function has been scaled so that it passes through the point* $(0, 1)$.

(Section 10.4 of the ESL book)

Intuition of the logistic loss (3 min discussion)

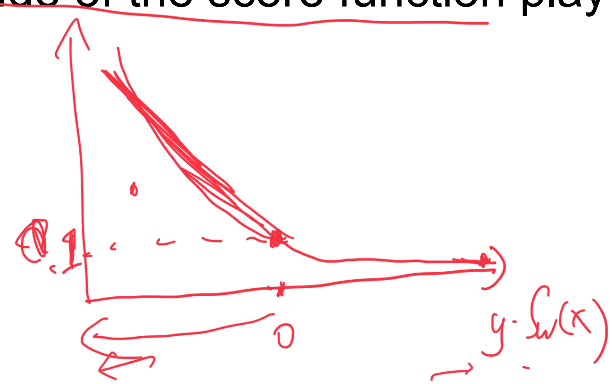
$$\log_2(1 + \exp(-y \cdot S_w(x)))$$

Try plotting the function value against $y \cdot S_w(x)$

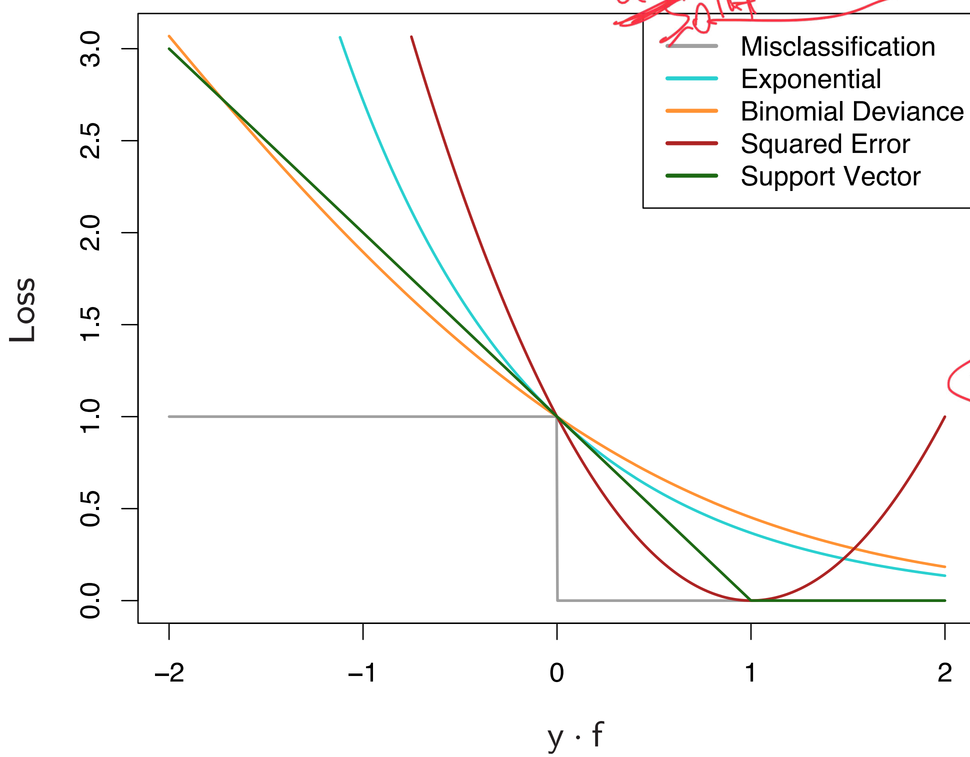
1. What happens when the classifier make a mistake?

2. What happens when the classifier are correct?

3. What role does the magnitude of the score function play?

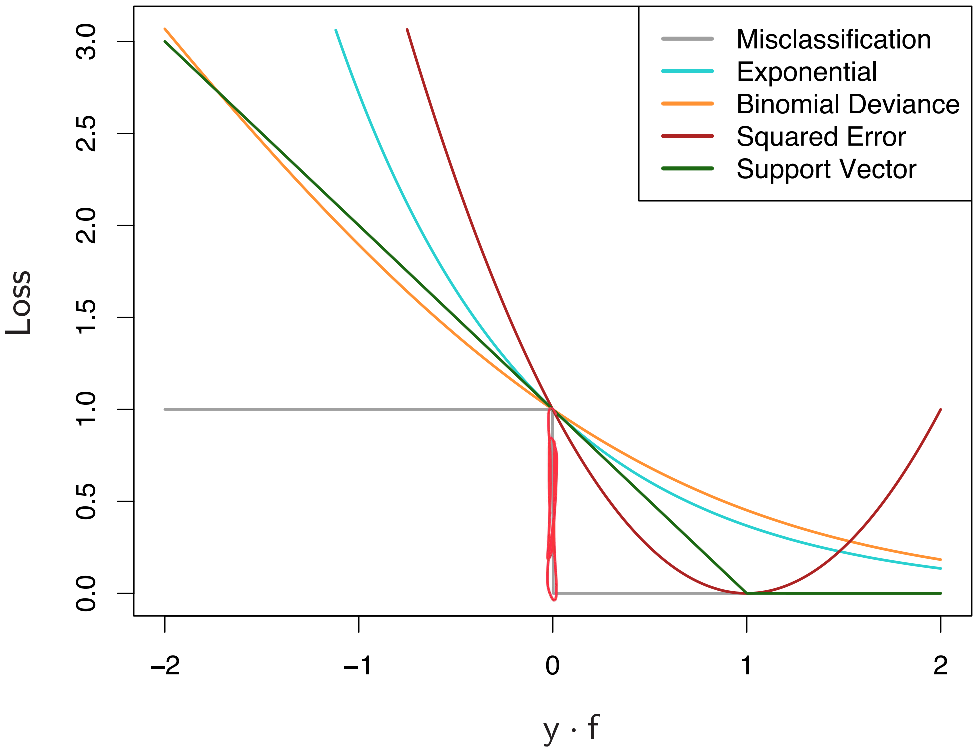


Why are "surrogate losses" easier to minimize?



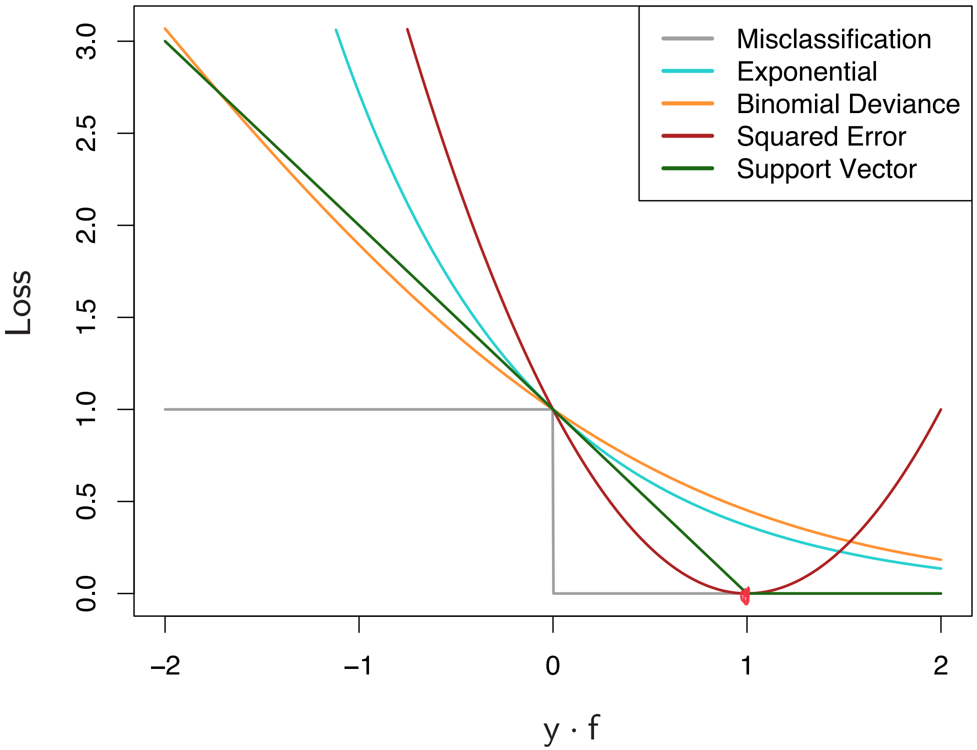
Why are "surrogate losses" easier to minimize?

- They are continuous.



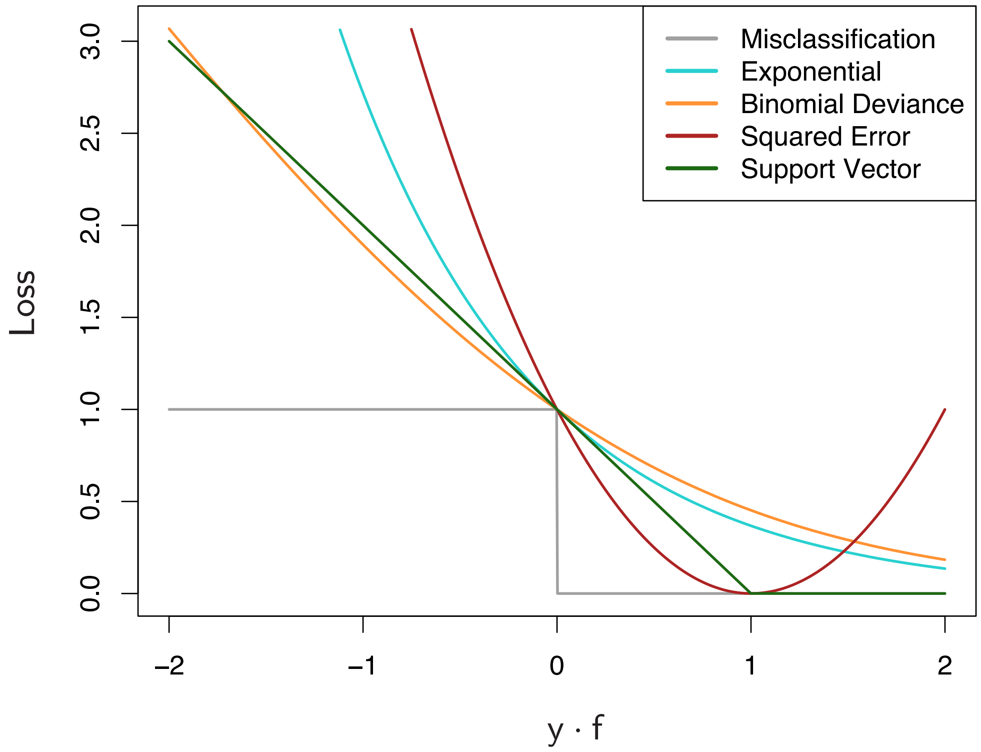
Why are “surrogate losses” easier to minimize?

- They are continuous.
- Differentiable (except hinge loss).

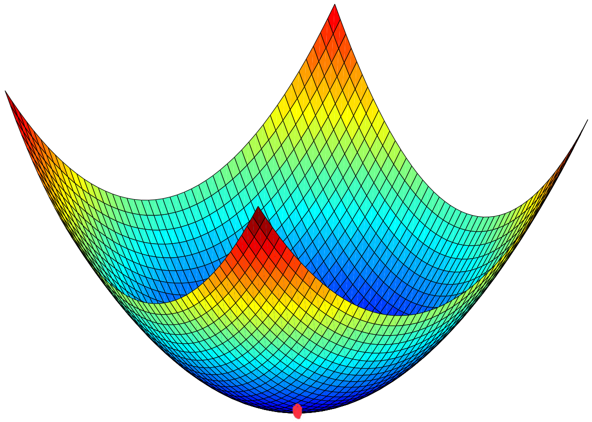


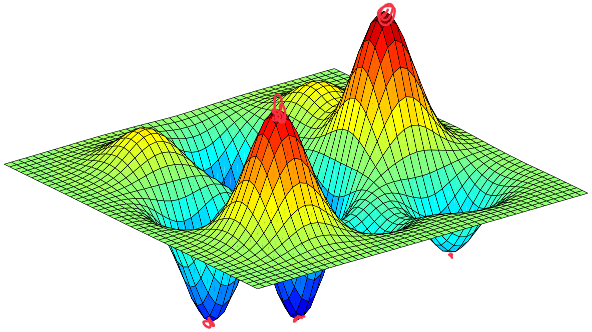
Why are "surrogate losses" easier to minimize?

- They are continuous.
- Differentiable (except hinge loss).
- Convex.



Convex vs Nonconvex optimization

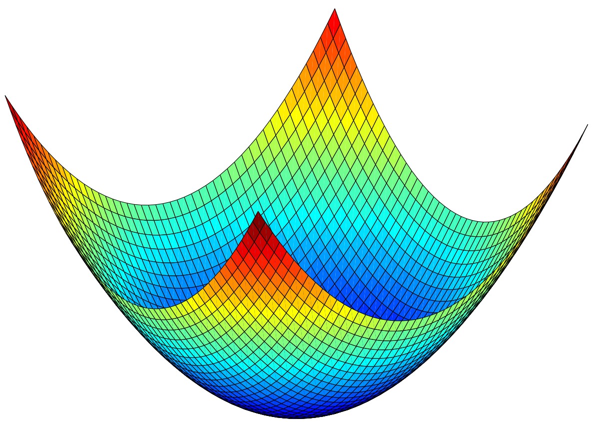


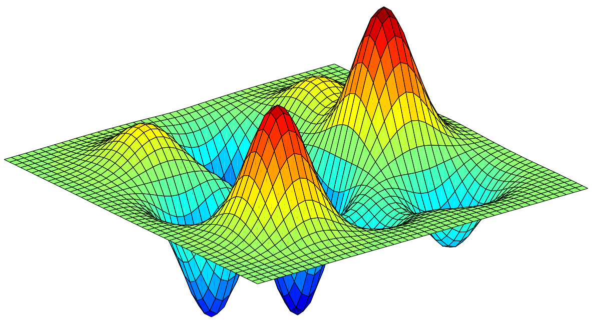


- Unique optimum: global/local.

- Multiple local optima
- In high dimensions possibly exponential local optima

Convex vs Nonconvex optimization





- Unique optimum: global/local.

- Multiple local optima
- In high dimensions possibly exponential local optima

* Be careful: The surrogate loss being convex does not imply all ML problems using surrogate losses are convex. Linear classifiers are, but non-linear classifiers are usually not. Take "convex optimization" to know more.

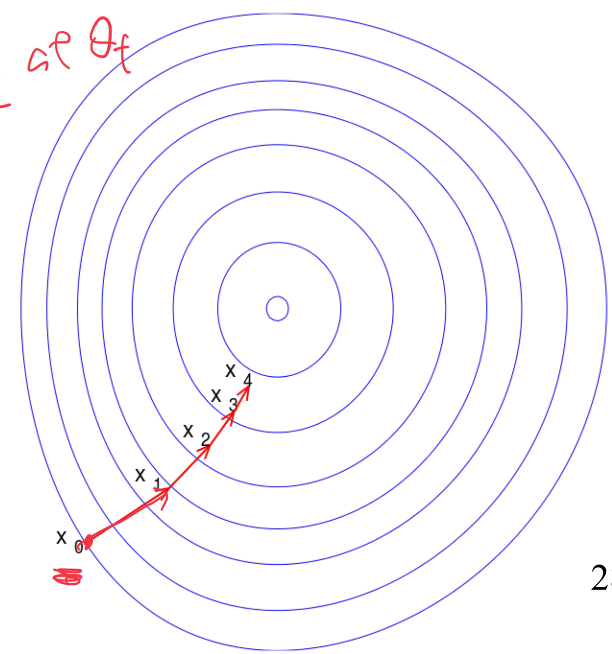
How do we optimize a continuously differentiable function in general?

- The problem: $\min_{\theta} f(\theta)$

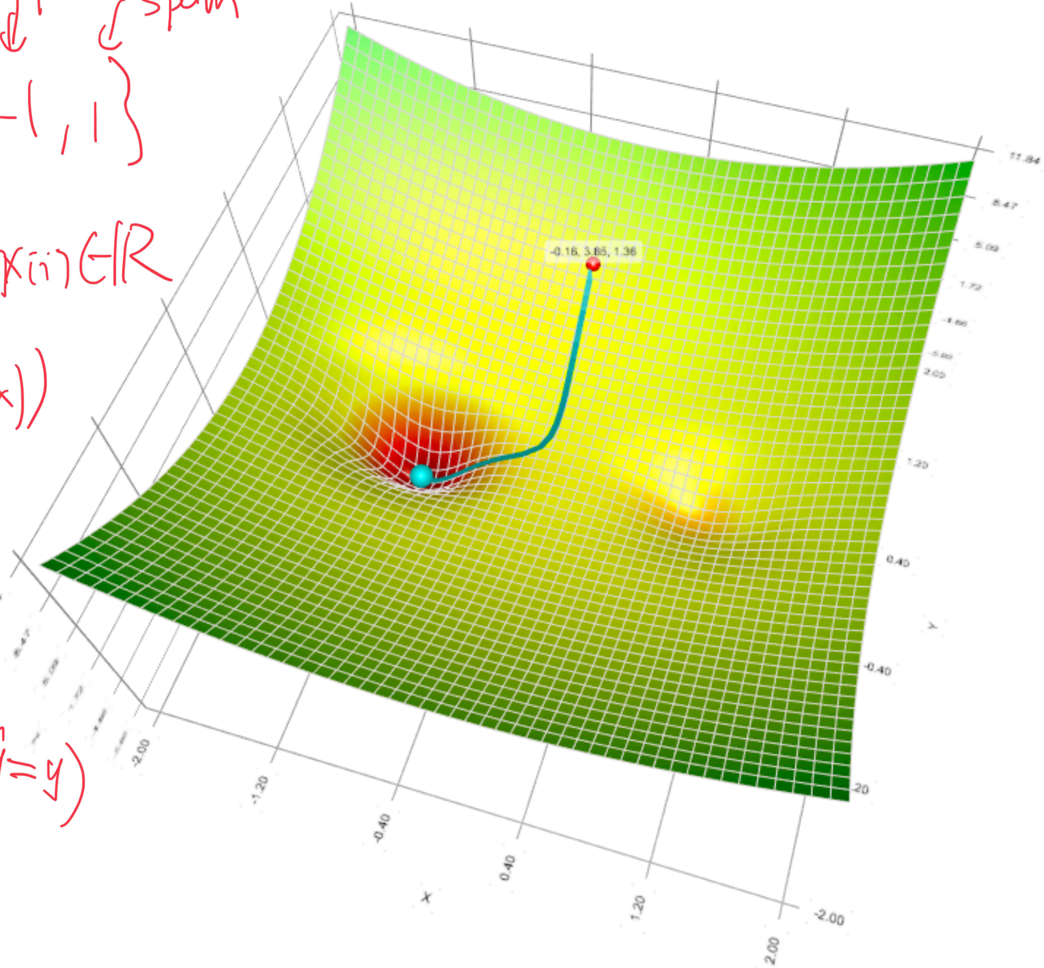
- Let's just optimize it anyway!
  - With gradient descent.

- Assumption: The objective function is differentiable almost everywhere.

$$\theta_{t+1} = \theta_t - \eta_t \nabla f(\theta_t)$$



Gradient Descent Demo



$y \in \{-1, 1\}$ (non-spam, spam)

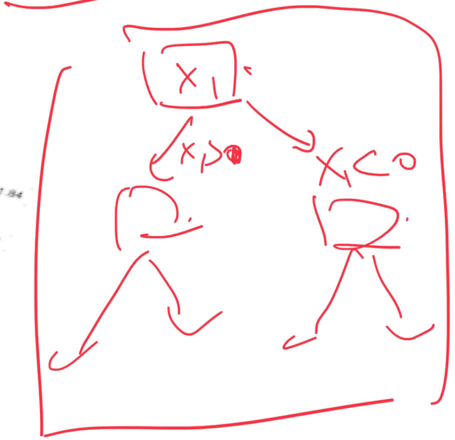
$S_w(x) = \sum_{i=1}^{d} w_i \cdot x(i) \in \mathbb{R}$

$\hat{y} = \text{sign}(S_w(x))$

$y \cdot S_w(x)$

$= |S_w(x)| \cdot \mathbb{1}(\hat{y} = y)$

Decision Tree



K-NN

- Play with this excellent tool yourself to build intuition:

  https://github.com/lilipads/gradient_descent_viz

Gradient of logistic loss for learning a linear classifier

- The function to minimize is

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-y_i \cdot x_i^T w))$$

- How to calculate the gradient?
  - Take out a piece of paper and work on it! (you have 3 min)
  - Hint:
    - Apply the linearity of the differential operator.
    - Apply the chain rule.

Gradient of logistic loss for learning a linear classifier

$$\nabla f(w) = \frac{1}{n}\sum_{i=1}^{n} \frac{\exp(-y_i \cdot x_i^T w)}{1 + \exp(-y_i \cdot x_i^T w)}(-y_i x_i)$$

Gradient of logistic loss for learning a linear classifier

$$\nabla f(w) = \frac{1}{n} \sum_{i=1}^{n} \frac{\exp(-y_i \cdot x_i^T w)}{1 + \exp(-y_i \cdot x_i^T w)} (-y_i x_i)$$

- Question: What is the time complexity of computing this gradient?

Gradient of logistic loss for learning a linear classifier

$$\nabla f(w) = \frac{1}{n} \sum_{i=1}^{n} \frac{\exp(-y_i \cdot x_i^T w)}{1 + \exp(-y_i \cdot x_i^T w)} (-y_i x_i)$$

- Question: What is the time complexity of computing this gradient?

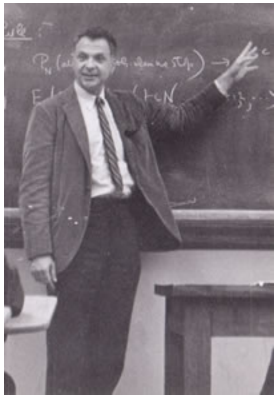
Gradient of logistic loss for learning a linear classifier

$$\nabla f(w) = \frac{1}{n} \sum_{i=1}^{n} \frac{\exp(-y_i \cdot x_i^T w)}{1 + \exp(-y_i \cdot x_i^T w)} (-y_i x_i)$$

- Question: What is the time complexity of computing this gradient?

Can we do better?

Stochastic Gradient Descent (Robbins-Monro 1951)



Herbert Robbins
1915 - 2001

- Gradient descent

$$\theta_{t+1} = \theta_t - \eta_t \nabla f(\theta_t)$$

- Stochastic gradient descent

$$\theta_{t+1} = \theta_t - \eta_t \hat{\nabla} f(\theta_t)$$

- Using a stochastic approximation of the gradient:

$$\mathbb{E}[\hat{\nabla} f(\theta_t) | \theta_t] = \nabla f(\theta_t)$$

$$\mathbf{Var}[\hat{\nabla} f(\theta_t) | \theta_t] \leq \sigma^2$$

Question: What's the time complexity of each iteration in SGD?

One natural stochastic gradient to consider in machine learning

- Recall that

$$\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \ell(\theta, (x_i, y_i))$$

One natural stochastic gradient to consider in machine learning

- Recall that

$$\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \ell(\theta, (x_i, y_i))$$

- Pick a **single** data point i uniformly at random
  - Use $\nabla_\theta \ell(\theta, (x_i, y_i))$

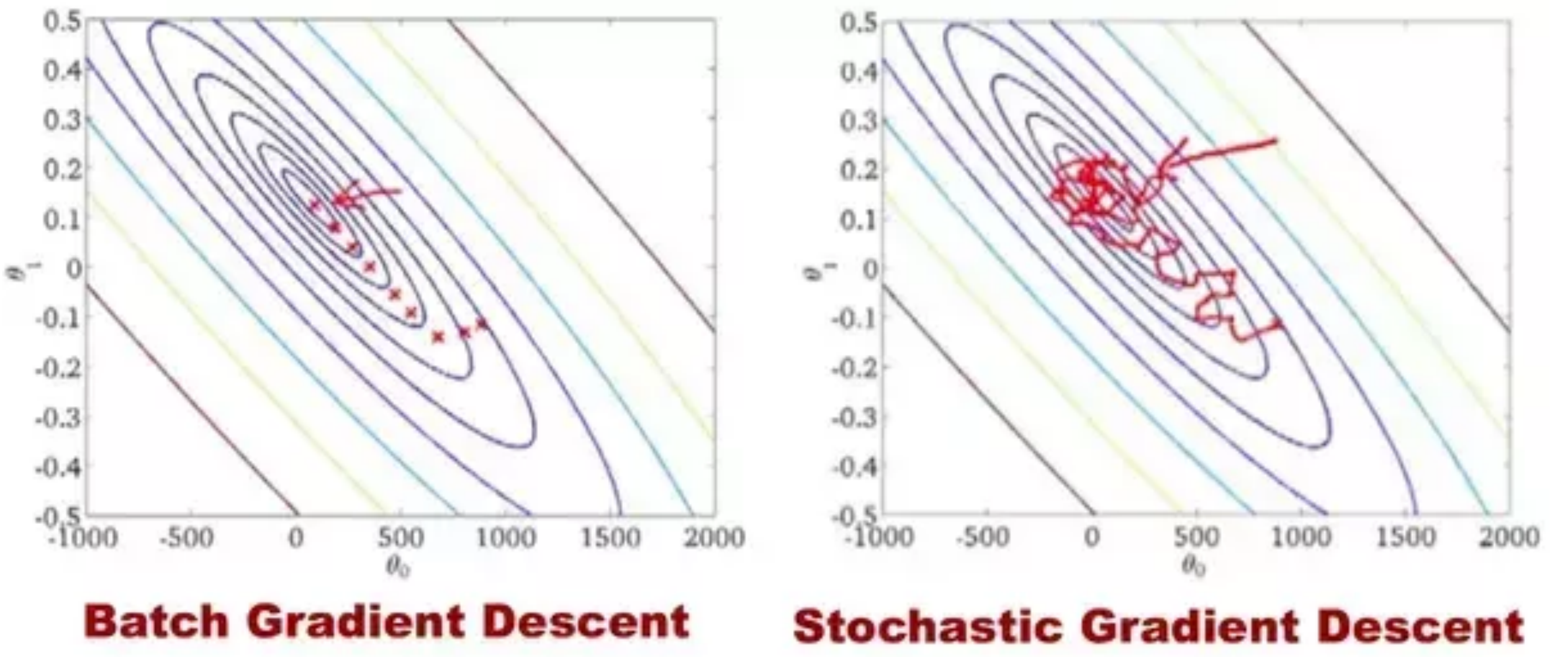
One natural stochastic gradient to consider in machine learning

- Recall that

$$\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \ell(\theta, (x_i, y_i))$$

- Pick a **single** data point i uniformly at random
  - Use $$\nabla_\theta \ell(\theta, (x_i, y_i))$$
  - Show that this is an unbiased estimator!

Illustration of GD vs SGD



Observation: With the time gradient descent taking one step. SGD would have already moved many steps.

Intuition of the SGD algorithm on the "Spam Filter" example

Contains hyperlinks

Proportion of misspelled words

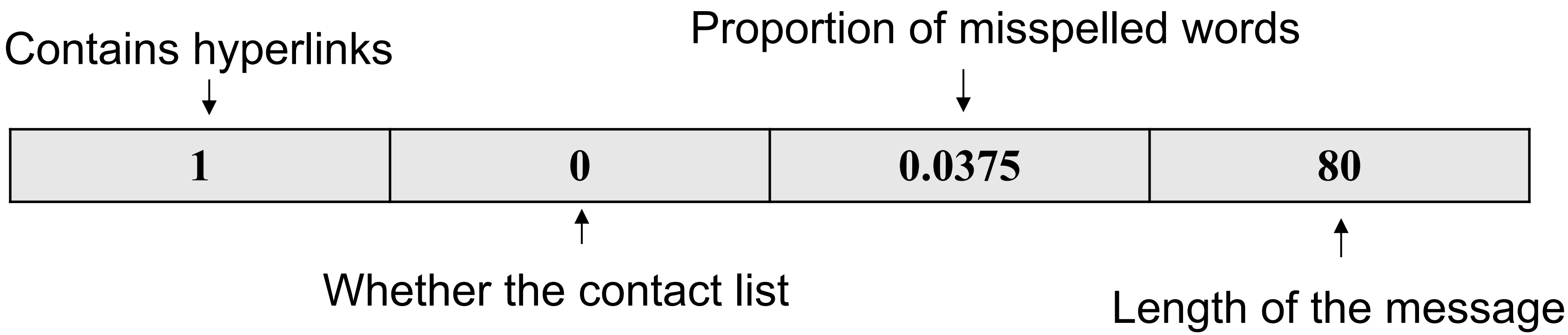
1	**0**	**0.0375**	**80**

Whether the contact list

Length of the message

- Score(x) = $w_0 + w_1$ * 1(hyperlinks) + w_2 * 1(contact list) + w_3 * misspelling + w_4 * length

Intuition of the SGD algorithm on the "Spam Filter" example



- $Score(x) = w_0 + w_1 * 1(hyperlinks) + w_2 * 1(contact\ list) + w_3 * misspelling + w_4 * length$

- Meaning of these weight?

Intuition of the SGD algorithm on the “Spam Filter” example

Contains hyperlinks ↓ | Proportion of misspelled words ↓

1	**0**	**0.0375**	**80**

↑ Whether the contact list | ↑ Length of the message

- Score(x) = $w_0 + w_1$ * 1(hyperlinks) + w_2 * 1(contact list) + w_3 * misspelling + w_4 * length

- Meaning of these weight?
  - The more positive, the more we think the feature is associated with Spam email.

Intuition of the SGD algorithm on the "Spam Filter" example

Contains hyperlinks

Proportion of misspelled words

1	**0**	**0.0375**	**80**

Whether the contact list

Length of the message

- Score(x) = $w_0 + w_1 * 1(\text{hyperlinks}) + w_2 * 1(\text{contact list}) + w_3 * \text{misspelling} + w_4 * \text{length}$

- Meaning of these weight?
  - The more positive, the more we think the feature is associated with Spam email.
  - The more negative, the less that we think the feature is associated with Spam email

Intuition of the SGD algorithm on the "Spam Filter" example

$$\nabla \ell(w, (x_i, y_i)) = \frac{\exp(-y_i \cdot x_i^T w)}{1 + \exp(-y_i \cdot x_i^T w)} (-y_i x_i)$$

Intuition of the SGD algorithm on the "Spam Filter" example

$$\nabla \ell(w, (x_i, y_i)) = \underbrace{\frac{\exp(-y_i \cdot x_i^T w)}{1 + \exp(-y_i \cdot x_i^T w)}}(-y_i x_i)$$

Scalar > 0:
≈ 0 if the prediction is correct
≈ 1 otherwise

Intuition of the SGD algorithm on the "Spam Filter" example

$$\nabla \ell(w, (x_i, y_i)) = \frac{\exp(-y_i \cdot x_i^T w)}{1 + \exp(-y_i \cdot x_i^T w)} (-y_i x_i)$$

Scalar > 0:
≈ 0 if the prediction is correct
≈ 1 otherwise

Vector of dimension d: provides the direction of the gradient

Intuition of the SGD algorithm on the “Spam Filter” example

$$\nabla \ell(w, (x_i, y_i)) = \underbrace{\frac{\exp(-y_i \cdot x_i^T w)}{1 + \exp(-y_i \cdot x_i^T w)}}(\boxed{-y_i x_i})$$

Scalar > 0:
≈ 0 if the prediction is correct
≈ 1 otherwise

Vector of dimension d: provides the direction of the gradient

If we receive an example [1, 0 , 0.0375, 80] like the one before.
And a label y = 1 saying that this is a spam.

How will the SGD update change the weight vector?

Intuition of the SGD algorithm on the "Spam Filter" example

$$\nabla \ell(w, (x_i, y_i)) = \frac{\exp(-y_i \cdot x_i^T w)}{1 + \exp(-y_i \cdot x_i^T w)} (-y_i x_i)$$

Scalar > 0:
≈ 0 if the prediction is correct
≈ 1 otherwise

Vector of dimension d: provides the direction of the gradient

If we receive an example [1, 0 , 0.0375, 80] like the one before.
And a label y = 1 saying that this is a spam.

How will the SGD update change the weight vector?

Then by moving w towards the negative gradient direction, we are changing the weight vector by increasing the weights. i.e., increasing the amount they contribute to the score function (if currently the classifier is making a mistake on this example)

How to choose the step sizes / learning rates?

- In theory:
  - Gradient decent: 1/L where L is the **Gradient Lipschitz constant** of the function we minimize.
  - SGD: $$\sum_t \eta_t = \infty, \sum_t \eta_t^2 < \infty$$
    - e.g. $\eta_t \in [1/t, 1/\sqrt{t})$

- In practice:
  - Use cross-validation on a subsample of the data.
  - Fixed learning rate for SGD is usually fine.
  - If it diverges, decrease the learning rate.
  - If for extremely small learning rate, it still diverges, check if your gradient implementation is correct.

The power of SGD

- Extremely general:
  - Specify an end-to-end differentiable score function, e.g., a complex neural network.
  - Beyond the context of machine learning

- Extremely simple: a few lines of code.

- Extremely scalable
  - Just a few pass of the data, no need to store the data

- People are continuing to discover that many methods are special cases of SGD.

Summary of the lecture

- Data splitting for detecting overfitting
- Distribution shift

- Learning a linear classifier:
  - Surrogate losses and linear logistic regression

- Gradient descent
  - Calculating gradient / making sense of gradient
  - Improving GD with Stochastic Gradient Descent

- Next week: wrap up ML, start probabilistic graphical models