CS5339 Machine Learning

Mathematical Formulations, Practical Issues, Feature Engineering

Lee Wee Sun
School of Computing
National University of Singapore
leews@comp.nus.edu.sg

Semester 2, 2019/20

Fundamental Issues

We will mostly be covering supervised learning with some unsupervised learning. We will look at three fundamental aspects:

- Representation
- Estimation
- Optimization

Representation

We will look at commonly encountered machine learning problems and the function classes used to represent the solutions:

- What are the strengths and weaknesses of those representations?
- What prior knowledge do the representations encode?

We will look at some of the commonly used function classes in more detail:

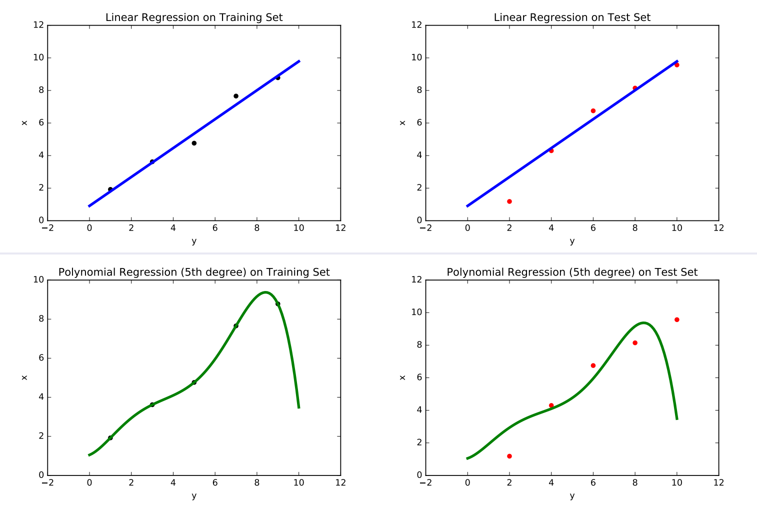
- How many functions are there in the class?
- What are functions that can/cannot be represented/approximated by the function class?
- Are there universal approximators that can approximate all functions?
- What is the rate of approximation? Does it require exponential resources to represent/approximate the function?
- Are there ways that can be used to improve the approximation?

Estimation

We will look at how much information is required to learn function classes.

- Overfitting and generalization.
- What are the problem parameters that determine how many training examples are required?
- How do we balance the complexity of the task with the amount of information available to get the best performance?

Overfitting



Test set mean square error: 0.52 for linear regression, 8.34 for polynomial regression.

- Minimizing error on the training set is not sufficient to ensure good performance on unseen data
  - Being able to reproduce performance on unseen data is often called **generalization**.
- In the example, a 5th degree polynomial can fit the training data (on odd numbers) perfectly compared to using linear regression, but has much poorer generalization (on even numbers).
- Known as **overfitting**.
- For good generalization, need to control complexity of function class.
- Typically balance training error with complexity control: model selection, regularization, etc.

Optimization

- What is the computational difficulty for learning commonly used function classes?
- Are there functions that are computationally difficult to learn, even when the sample complexity required is reasonable?
- Given the criteria for learning, how do we find the best hypothesis?

Outline

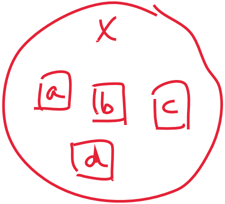
1. Supervised Learning
2. Unsupervised Learning
3. Discriminative/Generative Models
4. Model Selection
5. Features

Definitions

We will use the statistical learning framework for this course. We first formulate supervised learning.

- **Domain set**: An arbitrary set $\mathcal{X}$. Usually represented using features, e.g. vector $\mathbf{x}$ of pixel values for an image, often also called attributes or covariates. Domain points are referred to as *instances* and $\mathcal{X}$ as *instance space*.

Example : Handwritten character recog



- **Label set**: Usually denoted $\mathcal{Y}$. A discrete set for classification, e.g. $\{0, 1\}$ or $\{-1, 1\}$. Real valued for regression. Also called response variable.

Handwritten char $\quad y = \{a, b, c, \ldots\}$

- **Training data**: The training data or training set, $S = ((x_1, y_1), \ldots, (x_m, y_m))$ is a finite sequence of pairs in $\mathcal{X} \times \mathcal{Y}$. This is the input to the learning algorithm.

$$S = ((\boxed{c}, c), (\boxed{d}, d), \ldots)$$

- **Algorithm's output**: The output of the learning algorithm is a function $h : \mathcal{X} \to \mathcal{Y}$. This is often called the *predictor*, or *hypothesis*, or *classifier* in the case of classification problems.

Eg h is a linear predictor
or neural network

- **Data generation model**: We assume that S is sampled from a distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$.

$\mathcal{D}$ is a joint distribution over the domain points and labels.

example : S sampled from $D(x,y)$

$x \in \{ \boxed{a}, \boxed{b}, \dots \}$

Not $D(x)$ or $D(y)$

$y \in \{ a, b, \dots \}$

- Continued ...

  It is sometimes convenient to decompose the distribution into the *marginal distribution* $\mathcal{D}_x$ over $\mathcal{X}$

$$\text{Marginal } D_x(x=\boxed{a}) = \sum_{y=a}^{2} D(x=\boxed{a}, y)$$

  and the *conditional distribution* $\mathcal{D}_{y|x}(y|x)$ of y given x.

$$D_{y|x}(y=a \mid x=\boxed{a}) = \frac{D(x=\boxed{a}, y=a)}{D_x(x=\boxed{a})}$$

  In the case of classification/regression with no noise, $\mathcal{D}_{y|x}(y|x) = f(x)$ is a deterministic function of x.

- **Measures of success**: We will use a loss function to help measure our success. Given a set $\mathcal{H}$ of hypotheses of models, and a domain $\mathcal{Z}$, let ℓ be a function from $\mathcal{H} \times \mathcal{Z}$ to non-negative real numbers $\ell : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}_{+}$. We call such a function a **loss function**.

$Z = X \times Y$

example : Binary classification

$$\ell(h,(x,y)) = \begin{cases} 0 & \text{if } y = h(x) \\ 1 & \text{if } y \neq h(x) \end{cases}$$

The **risk function** is the expected loss of the hypothesis,

$$L_{\mathcal{D}}(h) = E_{z \sim \mathcal{D}}[\ell(h, z)].$$

We are interested in finding a hypothesis h that has small risk, or expected loss. ← Goal of learner

Some commonly used loss functions:

- The **0-1 loss** measures the misclassification error in classification

$$\ell_{0-1}(h,(x,y)) = \begin{cases} 0 & \text{if } h(x) = y \\ 1 & \text{if } h(x) \neq y. \end{cases}$$

More generally

$$\ell(h,(x,y)) = a_{y',y} \quad \text{if } h(x) = y'$$

- The **square loss** is commonly used for regression

$$\ell_{sq}(h,(x,y)) = (h(x) - y)^2.$$

Another example : Absolute loss

$$\ell(h,(x,y)) = |h(x) - y|$$

Empirical Risk Minimization

- The learner does not know $\mathcal{D}$ and only have access to the training set S, a sample from $\mathcal{D}$.

$S=((x_1,y_1), \ldots, (x_m,y_m)$

No access to $D(x,y)$

- For a predictor $h : \mathcal{X} \to \mathcal{Y}$, we can approximate the expected error by using the training set error

$$L_S(h) = \frac{|\{i \in [m] : h(x_i) \neq y_i\}|}{m},$$

where $[m] = \{1, \ldots, m\}$.

Notation

$L_S(h)$ is approximating for $L_D(h)$

- The training set error can be rewritten using the 0-1 loss

$$L_S(h) = \frac{\sum_{i=1}^{m} \ell_{0-1}(h,(x_i,y_i))}{m}.$$

- The training set error is often called the *empirical error* or *empirical risk*.
- Given a hypothesis class $\mathcal{H}$, finding the hypothesis $h \in \mathcal{H}$ that minimizes the empirical risk is a simple learning strategy.

find h that minimizes $L_S(h)$

- This is often called *empirical risk minimization* (ERM).

and hope it approximately minimizes $L_D(h)$

I.I.D. Assumption

- We often make assumptions about the data generation process.
- One common assumption is that the data is **independently and identically distributed (i.i.d.)** according to the distribution $\mathcal{D}$.

z_t is independent of $z_{t-1}, \ldots, z_1$: $p(z_t | z_{t-1}, \ldots, z_1) = p(z_t)$

Example : knowing prev characters does not help predict next char

Identically distr : $p(z_{t+1}) = p(z_t)$

- This is denoted $S \sim \mathcal{D}^m$ where m is the training set size, and $\mathcal{D}^m$ denotes the probability over m-tuples induced by applying $\mathcal{D}$ to pick each element of S independently.

Exercise 1: Design and analysis of machine learning algorithms often assume the i.i.d. assumption. Consider the implications for following problem.

- You are given multiple pages of handwritten text.
- You segment the text into characters and label them, giving you a sequence $S = ((x_1, y_1), ...)$, where x_i is a scanned image of a character and y_i is its label.
- When deployed, you expect to also follow the same process, getting a sequence of unlabeled scanned images of characters that need to be labeled.

1. You train a classifier $h(x)$ that depends only on the current scanned image x. Given a large enough training set, do you expect the long term test error to be similar to the training error? If i.i.d., yes. Training set likely similar to test condition & representative. Real seq likely not iid, but dependence decay over time. long enough training seq likely representative.

2. If you do not assume the data to be i.i.d., how can you exploit it? Classify longer seq each time, e.g classify a word rather than a character. Use predictors with memory, e.g recurrent neural net

Connection to Maximum Likelihood

minimize empirical risk ⟺ maximize likelihood

- Maximum likelihood is a commonly used estimation method in statistics.
- Assume that the data distribution $\mathcal{P}$ is known up to some parameter $h \in \mathcal{H}$. Example: coin tossing

  Bernoulli dist, parameter θ : p(head)

  $P(x=H) = \theta$, $P(x=T) = 1-\theta$

- The maximum likelihood principle suggests to select the h or θ that maximizes the probability of the data S being observed:

$$h_{ML} = \arg\max_{h \in \mathcal{H}} \mathcal{P}(S|h).$$

$S = (0,1,1,0..)$

$P(z_1, \ldots, z_m) = \prod_i P(z_i)$

- For i.i.d. data $S = (z_1, \ldots, z_m) \sim \mathcal{D}^m$, this becomes:

$$h_{ML} = \arg\max_{h \in \mathcal{H}} \prod_{i=1}^{m} \mathcal{D}(z_i | h).$$

Example: Observe 4 heads, 6 tails

$\arg\max_{\theta} P(S|\theta) = \theta^4 (1-\theta)^6$

Usually max $\log P(S|\theta)$

$\log P(S|\theta) = 4 \log \theta + 6 \log (1-\theta)$

$\frac{d}{d\theta} \log P(S|\theta) = \frac{4}{\theta} - \frac{6}{1-\theta}$ Set to zero

$\frac{4}{\theta} = \frac{6}{1-\theta}$

$\theta = \frac{4}{10}$

- For supervised learning, $z_i = (x_i, y_i)$ and we have

$$h_{ML} = \arg\max_{h \in \mathcal{H}} \prod_{i=1}^{m} \mathcal{D}(y_i | x_i, h)\mathcal{D}(x_i).$$

wo h

$$D(x_i, y_i | h) = D(x_i | h) D(y_i | x_i, h)$$

from chain rule of prob

$$P(x_1, \ldots x_m) = P(x_1) P(x_2 | x_1) \ldots P(x_m | x_{m-1} \ldots x_1)$$

- As $\mathcal{D}(x_i)$ does not depend on h, we can drop $\mathcal{D}(x_i)$ from the criterion

$$h_{ML} = \arg\max_{h \in \mathcal{H}} \prod_{i=1}^{m} \mathcal{D}(y_i | h, x_i).$$

- Instead of maximizing the likelihood, it is often more convenient to maximize the log likelihood:

$$h_{ML} = \arg\max_{h \in \mathcal{H}} \sum_{i=1}^{m} \log \mathcal{D}(y_i | h, x_i).$$

Maximum Likelihood and Minimizing Empirical Risk

- For some distributions $\mathcal{D}$, maximizing the log likelihood is equivalent to empirical risk minimization with appropriate loss functions.
- Assume that the observations are generated by a true function plus some additive noise $y_i = h(x_i) + \xi_i$.

$$D(y_i \mid x_i, h) = D(\xi_i = y_i - h(x_i))$$

$$h_{ML} = \arg\max_{h} \sum_{i=1}^{m} \log D(\xi_i = y_i - h(x_i))$$

- If the density of ξ_i is zero mean Gaussian $\frac{1}{\sqrt{2\sigma^2\pi}}\exp(-\frac{\xi_i^2}{2\sigma^2})$, we get

$$\log \mathcal{D}(y_i|h, x_i) = -\frac{(y_i - h(x_i))^2}{2\sigma^2} + C$$

where C is a constant.

- Maximizing the log likelihood is equivalent to minimizing the empirical risk with the square loss

$$L_S(h) = \frac{\sum_{i=1}^m (y_i - h(x_i))^2}{m} = \frac{\sum_{i=1}^m \ell_{sq}(h, (x_i, y_i))}{m}.$$

- More generally, when an equivalent distribution can be found, empirical risk minimization is equivalent to maximum likelihood when for loss function

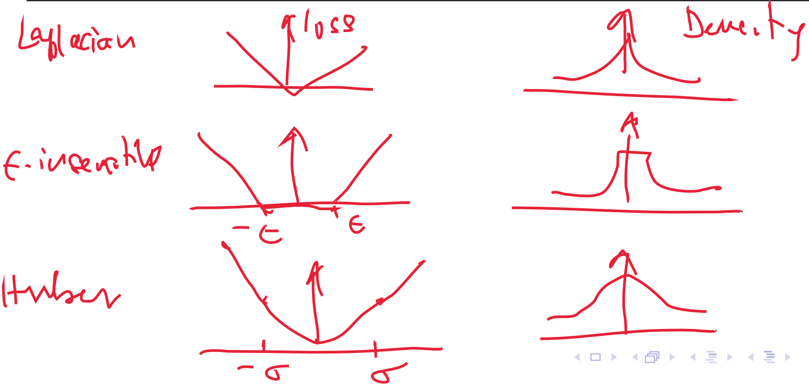
$$\ell(h,(x,y)) = -\ln p(y|x,h).$$

- For regression, assuming $y_i = h(x_i) + \xi_i$, we have

$$\ell(h,(x,y)) = -\ln p_\xi(y - h(x)).$$

Some other commonly used loss function for regression and their corresponding densities:

	loss function	density model
ϵ-insensitive	$\max(\lvert\xi\rvert - \epsilon, 0) = \lvert\xi\rvert_\epsilon$	$\frac{1}{2(1-\epsilon)} \exp(-\lvert\xi\rvert_\epsilon)$
Laplacian	$\lvert\xi\rvert$	$\frac{1}{2} \exp(-\lvert\xi\rvert)$
Huber's robust loss	$\begin{cases} \frac{1}{2\sigma}(\xi)^2 & \text{if } \lvert\xi\rvert \leq \sigma \\ \lvert\xi\rvert - \frac{\sigma}{2} & \text{otherwise} \end{cases}$	$\propto \begin{cases} \exp\left(\frac{1}{2\sigma}(\xi)^2\right) & \text{if } \lvert\xi\rvert \leq \sigma \\ \exp\left(\lvert\xi\rvert - \frac{\sigma}{2}\right) & \text{otherwise} \end{cases}$



Classification Loss Functions

- For classification, we have a finite set of labels. $Y = \{1, \ldots, k\}$
- Assume that the predictor outputs a probability distribution $q = p(y|x, h)$ over the possible classes.
- We can maximize the likelihood, or correspondingly do ERM using the log loss

$$\ell(h, (x, y)) = -\ln p(y|x, h).$$

- Sometimes called the cross entropy loss.

0-1 loss not continuous, not diff
output distr and use log
loss get continuous diff loss

$Y = \{-1, 1\}$

- For binary classification using $h(x) \in (-\infty, \infty)$, we often compose the output of our function $h(x)$ with the logistic (also called sigmoid) function (shown below) to get a probability model $P(y = 1|x, h) = \frac{\exp(h(x))}{1+\exp(h(x))} = \frac{1}{1+\exp(-h(x))}$.

Example: Linear fn

$h(x) = \sum_{i=1}^{d} w_i x_i$

$h(x) \in (-\infty, \infty)$

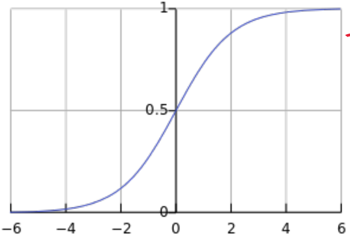


Figure from [6].

$f(x) = \frac{1}{1 + \exp(-h(x))}$

- In this model, $P(y=-1|x,h) = 1 - \frac{\exp(h(x))}{1+\exp(h(x))} = \frac{1}{1+\exp(h(x))}$.

$$P(y|x,h) = \begin{cases} \frac{1}{1+e^{-h(x)}} & \text{if } y = 1 \\ \frac{1}{1+e^{h(x)}} & \text{if } y = -1 \end{cases}$$

- For $y \in \{1,-1\}$, can write $P(y|x,h) = \frac{1}{1+\exp(-yh(x))}$.
- So log loss can be written as

$$\ell_{log}(h,(x,y)) = -\log P(y|h(x)) = \log(1+\exp(-yh(x))).$$

- When $h(x)$ is a linear function, this is often called *logistic regression*.
- Neural networks are also often used as $h(x)$.

- This model can be generalized to multi-class classification by using the model $P(y = k|x, h) = \frac{\exp(h_k(x))}{\sum_{i=1}^{K} \exp(h_i(x))}$.
- When $h_i(x)$ is linear, this is often called *multiclass logistic regression*, *softmax regression* or *maximum entropy classifier*.

$Y = \{1, \ldots, K\}$

MAP Estimation and Regularization

- Empirical risk minimization or maximum likelihood may overfit the data if the hypothesis class used is too powerful.
- The estimate depends entirely on the data.
- One way to reduce overfitting is to balance the dependence on the data with prior knowledge.
- Instead of maximizing the likelihood

$$h_{ML} = \arg\max_{h \in \mathcal{H}} \prod_{i=1}^{m} \mathcal{D}(z_i | h),$$

in *maximum a posteriori (MAP) estimation*, find the parameter that maximizes the posterior probability $P(h|D)$.

- In MAP estimation, we want to find

c.f. max likelihood $\arg\max_{h} P(D|h)$

$$\begin{aligned} h_{MAP} &= \arg\max_{h\in\mathcal{H}} P(h|D) \\ &= \arg\max_{h\in\mathcal{H}} P(D|h)P(h)/Z \\ &= \arg\max_{h\in\mathcal{H}} \prod_{i=1}^{m} \mathcal{D}(z_i|h)P(h)/Z, \end{aligned}$$

↓ independence

where Z is a constant normalization factor.

likelihood

Bayes rule: $P(y|x) = \dfrac{P(x|y)P(y)}{Z}$

prior

evidence

$$Z = \sum_{y'} P(x|y')P(y')$$

- Taking log, for the supervised learning case, we get

$$h_{MAP} = \arg\max_{h \in \mathcal{H}} \left(\sum_{i=1}^{m} \log \underbrace{\mathcal{D}(y_i | h, x_i)}_{\text{likelihood}} + \log \underbrace{P(h)}_{\text{prior}}. \right)$$

- Balance between fitting the data (likelihood) well and fitting the prior well.
- For example, for linear regression, we often put a zero mean Gaussian prior on the weight vector. Assuming additive Gaussian noise, the MAP estimator minimizes a combination of the empirical risk with the square loss and the size of the linear function weights

$$p(w) = \frac{1}{2} e^{-\lambda ||w||^2}$$

$$\frac{\sum_{i=1}^{m} (y_i - \mathbf{w}^T \mathbf{x}_i))^2}{m} + \lambda ||\mathbf{w}||^2,$$

where **w** is the weight vector of the linear regressor.

- This is often called *ridge regression* or *penalized least square*.
- The term $||\mathbf{w}||^2$ is called the regularizer.
- Minimizing a combination of empirical risk and a regularizer is also called regularized loss minimization.
- Under the MAP interpretation, we are maximizing a combination of prior and likelihood functions.
- The regularizer can also be viewed as a measure of complexity, so we are balancing risk minimization with complexity.

Exercise 2:

In this experiment, the training and test examples are generated with the function $y = x$ with Gaussian noise added. We fit a linear function and a 10th degree polynomial.

For the 10th degree polynomial, we fit using polynomial regression and then with ridge regression. In scikit learn, ridge regression finds $\min_w ||Xw - y||_2^2 + \alpha||w||_2^2$.

- Run the experiment.
- Change the variable *data_size* to 10 and run it again.

Comment on the experiment results.

- with size 100 : poly reg overfits slightly
  ridge regression improves performance slightly
- with size 10 : poly reg overfit badly
  ridge regression improves substantially

Bayesian Estimation

- In Bayesian estimation, instead of selecting a single h, we maintain the posterior distribution over the parameters $P(h|z_1,\ldots,z_m)$. ← In MAP, we find arg max of this
- This can be used to make optimal prediction (assuming the Bayesian model is correct) for the variable of interest. For example,

$$\begin{aligned} P(y|x,z_1,\ldots,z_m) &= \int_h P(y,h|x,z_1,\ldots,z_m) \quad \leftarrow \text{marginalizing } P(y) = \int_h P(y,h) \\ &\qquad \downarrow \text{chain rule} \\ &= \int_h P(y|h,x,z_1,\ldots,z_m)\underset{\substack{\uparrow \\ \text{posterior}}}{P(h|z_1,\ldots,z_m)}. \end{aligned}$$

posterior and Often intractable need approx

- For classification, we would predict with the value y that maximizes $P(y|x, z_1, \ldots, z_m)$ to get optimal prediction.
- Bayesian estimation is often computationally more expensive unless there is special structure that can be exploited (e.g. use of conjugate prior).

Exercise 3: What do we mean by optimal prediction?
Consider predicting the label of x, which can take the values 1, 2, or 3. Assume that we know the true probabilities $p(y = 1|x) = 0.3$, $p(y = 2|x) = 0.4$, $p(y = 3|x) = 0.3$. Which label should you predict to minimize prediction error?

1. 1
2. [✓ 2]
3. 3

If pred y, prob of err $1 - p(y|x)$

min prob err : $\arg\min_y (1 - p(y|x))$

Give the formula for the minimum prediction error.

$= 1 - \max_y p(y|x)$

Summary

① Formulate as loss minimization

②

$$\begin{tikzcd}
\text{ERM} \arrow[r, leftrightarrow, "\text{often equiv}"] \arrow[d, "\text{Reduce overfitting}"] & \text{max likelihood} \\
\text{Reg loss minimization} \arrow[r, leftrightarrow] & \text{MAP estimation}
\end{tikzcd}$$

③ Bayesian estimation

Outline

1. Supervised Learning
2. Unsupervised Learning
3. Discriminative/Generative Models
4. Model Selection
5. Features

Unsupervised Learning

In unsupervised learning, $S = \{x_1, \ldots, x_m\}$, no y_i'

- Many (but not all) unsupervised learning problems can be posed as density estimation problems, i.e. learning the distribution of the data. learn $p(x)$
- We can often pose the problem of learning the data distribution as a maximum likelihood (or maximum a posteriori) estimation problem. $\arg\max_{\theta} P_{\theta}(x)$
- For example, we may model data using a mixture of Gaussians and learn using maximum likelihood for clustering. After learning, the means (centers) of the Gaussians can be treated as our cluster centers.

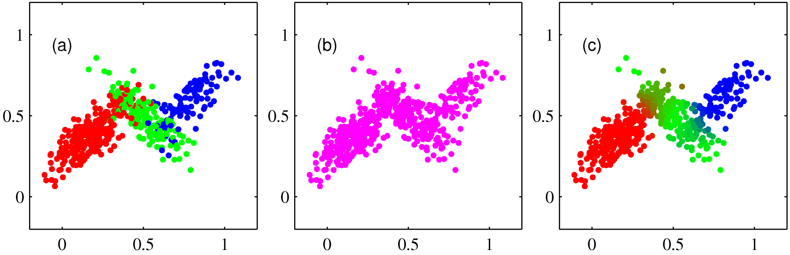


Figure: From [1]. Mixture of 3 Gaussians. Left figure shown in red, green, blue corresponding to the three mixtures. Middle is sample from marginal $p(\mathbf{x})$. Right shows the estimated component for each point using proportions of red, blue and green.

$$P(x) = \sum_{y=1}^{k} p(x,y)$$

In this example, the marginal distribution is

not observed

$$P(X = \mathbf{x}) = \sum_{y=1}^{k} P(Y = y) P(X = \mathbf{x} | Y = y)$$

chain rule

$$= \sum_{y=1}^{k} c_y \frac{1}{(2\pi)^{d/2} |\Sigma_y|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_y)^T \Sigma_y^{-1} (\mathbf{x} - \boldsymbol{\mu}_y) \right).$$

One way to learn c_y, $\boldsymbol{\mu}_y$ and Σ_y is to use maximum likelihood.

Density estimation for sequence $x_1, \ldots, x_m$

- A common type of model is an autoregressive model: From chain rule of probability

$$P(x_1, \ldots, x_m) = P(x_1)P(x_2|x_1)P(x_3|x_2, x_1) \cdots P(x_m|x_{n-1}, \ldots, x_1).$$

For example, language model:

- **the** cat sat on the mat $P(x_1)$
- the **cat** sat on the mat $P(x_2|x_1)$
- the cat **sat** on the mat $P(x_3|x_2, x_1)$
- the cat sat **on** the mat $P(x_4|x_3, x_2, x_1)$
- the cat sat on **the** mat $P(x_5|x_4, x_3, x_2, x_1)$
- the cat sat on the **mat** $P(x_6|x_5, x_4, x_3, x_2, x_1)$

The log likelihood decomposes nicely into a sum.

$$\log P(x_1, \ldots, x_m) = \sum_{i=1}^{n} \log P(x_i | x_{i-1}, \ldots, x_i).$$

Learning is often done by maximum likelihood: select h that maximizes

$$\log P(x_1, \ldots, x_m | h) = \sum_{i=1}^{n} \log P(x_i | x_{i-1}, \ldots, x_i | h)$$

where h comes from a class of functions (e.g. recurrent neural networks) representing conditional distribution.

Relationship with Data Compression/Information Theory

- As described earlier, maximizing the log likelihood $\sum_{i=1}^{m} \log P(x_i|h)$ of the data $x_1, \ldots, x_m$ is equivalent to minimizing the empirical risk using the log loss $-\sum_{i=1}^{m} \log P(x_i|h)$.
- For **lossless compression**, using $P(x_1, \ldots, x_m)$ to compress $x_1, \ldots, x_m$ using Huffman coding or arithmetic coding, gives the shortest code length of $-\sum_{i=1}^{m} \log P(x_i|h)$ (ignoring rounding required for discrete lengths).
  - Maximizing $P(x_i|h)$ minimizes number of bits required.

Another way to think of log loss and log likelihood

- If $Q(x_1, \dots, x_m)$ is the true data distribution and the learning algorithm outputs the distribution $P(x_1, \dots, x_m)$, the expected log loss (code length) is $-\sum Q(x_1, \dots, x_m) \log P(x_1, \dots, x_m)$.
- Hence, many (but not all) unsupervised learning problems can be treated as problems of minimizing the expected risk, just like supervised learning.
- The difference between the expected code length and optimal code length is known as the Kullback-Lieber divergence from P to Q "distance" between distributions

Not a metric. Does not satisfy triangle inequality

$$\begin{aligned} D_{KL}(Q||P) &= -\sum Q(x_1, \dots, x_m) \log P(x_1, \dots, x_m) \\ &\quad + \sum Q(x_1, \dots, x_m) \log Q(x_1, \dots, x_m) \\ &= \sum Q(x_1, \dots, x_m) \log \frac{Q(x_1, \dots, x_m)}{P(x_1, \dots, x_m)}. \end{aligned}$$

Expected code length

Opt code length

Other Optimization Problems

- Density estimation by maximizing likelihood is one common formulation for unsupervised learning.
- Also, often formulated as (approximately) optimizing other objectives.
- For example, K-means is a clustering algorithm for partitioning a set of points in $\mathbb{R}^d$ into k clusters, minimizing

$$\mathcal{L}(\mathbf{C}, \boldsymbol{\mu}) = \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

where $\boldsymbol{\mu}_i \in \mathbb{R}^d$ is a vector of length d acting as a cluster center and $\mathbf{C} = C_1, \ldots, C_k$ partitions the dataset with C_i being a subset of points whose closest cluster center is $\boldsymbol{\mu}_i$.

- Generative adversarial networks (GAN) is a popular recent model that solves the following two player minimax game:

Generator

Discriminator

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1-D(G(z)))],$$

max ← D → min : discriminate real & generated

  - G is a function that generates samples: takes in a random number z and outputs data of the target distribution
  - the first component of the objective tries to maximizes the likelihood of the observed data using D, and
  - the second component uses D to minimize the likelihood of data generated by G. G makes it hard for D to discriminate

GAN generated images from [4].

Summary

Unsupervised learning often formulated as density estimation

- can learn using max likelihood
- same as log loss
- same as minimizing bits to describe data

Common to reduce learning to optimization problem on training data e.g. k-means, GAN, etc.

Outline

1. Supervised Learning
2. Unsupervised Learning
3. Discriminative/Generative Models
4. Model Selection
5. Features

Generative Models

- Learning a density model $p(x)$, e.g. using unsupervised learning methods, is actually sufficient for most machine learning tasks
  - Given any set of observed variables x_o, we can predict any set of target variables x_t by doing probabilistic inference to compute $p(x_t|x_o)$.
- For example, we can ignore the fact that the data for supervised learning comes in pairs $z = (x, y)$ and learn the data distribution (density estimation) for z.
  - After we have learned a model for $p(z) = p(x, y)$, we perform inference to get our estimate of y by computing $p(y|x)$.
- We illustrate this approach using two common learning models: naive Bayes and linear discriminant analysis (LDA).

Naive Bayes

Eg Spam classification

- Consider the problem of predicting the label $y \in \{-1, 1\}$ on the basis of the feature vector $\mathbf{x} = (x_1, \ldots, x_d)$. For simplicity, we assume $x_i \in \{0, 1\}$. x_i is indicator fn for whether word i is in email
- The Bayes optimal classifier is

$$
\begin{aligned}
h_{Bayes}(\mathbf{x}) &= \arg\max_{y \in \{-1,1\}} P(Y = y | X = \mathbf{x}) \quad \text{Bayes rule} \\
&= \arg\max_{y \in \{-1,1\}} P(Y = y) P(X = \mathbf{x} | Y = y) / Z
\end{aligned}
$$

  where we have used Bayes rule in the second equality and Z is a constant normalizing factor.
- In Naive Bayes, we make the (naive) generative assumption that the features are independent of each other given the label, i.e.

$$
P(X = \mathbf{x} | Y = y) = \prod_{i=1}^{d} P(X_i = x_i | Y = y).
$$

- Hence the classifier becomes

$$h_{NB}(\mathbf{x}) = \arg\max_{y \in \{-1,1\}} P(Y = y) \prod_{i=1}^{d} P(X_i = x_i | Y = y)$$

- The learning problem becomes that of learning $P(Y = y)$ and $P(X_i = x_i | Y = y)$. This can be done using maximum likelihood or MAP.

Max likelihood: $P(Y=y) = \frac{\# y}{\sum_{y_i} \# y_i}$

#y is count of y in training set

$P(X_i = x_i | Y=y) = \frac{\#(x_i \wedge y)}{\# y}$

Naive Bayes as a Linear Classifier

- Taking logs, we see that

$$h_{NB}(\mathbf{x}) = \arg\max_{y\in\{-1,1\}} \log P(Y=y) + \sum_{i=1}^{d} \log P(X_i = x_i | Y = y)$$

- Let $\log P(Y=y) = w_0^y$ and $\log P(X_i = x_i | Y = y) = w_{i,0}^y x_{i,0} + w_{i,1}^y x_{i,1}$ where $x_{i,k}$ is the indicator function that takes the value 1 when $x_i = k$ and zero otherwise. $x_{i,0}$, $x_{i,1}$ selects between $w_{i,0}^y$ and $w_{i,1}^y$
- We can then rewrite the Naive Bayes classifier as

$$\begin{aligned} h_{NB}(\mathbf{x}) &= \mathrm{sign}((w_0^1 - w_0^{-1}) + \\ &\quad \sum_{i=1}^{d}((w_{i,0}^1 - w_{i,0}^{-1})x_{i,0} + (w_{i,1}^1 - w_{i,1}^{-1})x_{i,1}) \end{aligned}$$

Subtract equations

- Setting $w_0 = w_0^1 - w_0^{-1}$ and $w_{i,j} = w_{i,j}^1 - w_{i,j}^{-1}$, we have

  $$h_{NB}(\mathbf{x}) = \mathrm{sign}(w_0 + \sum_{i=1}^{d}(w_{i,0}x_{i,0} + w_{i,1}x_{i,1}))$$

  linear classifier!

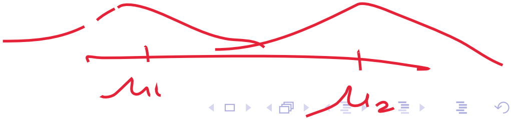
  which is a linear classifier with the features vector $\mathbf{x}$ encoded using indicator functions for the value of each feature (commonly called *one-hot* encoding).
- It is common to add a term w_0 to linear functions to encode the bias (e.g. caused by prior probabilities).
  - Strictly, the function becomes an *affine* function.
- For convenience, we often add a feature x_0 that always takes the value 1 to the feature vector to get $\mathbf{x} = (x_0, x_1, \ldots, x_d)$, so that we can represent a linear classifier as $\mathrm{sign}(\mathbf{w}^T\mathbf{x}) = \mathrm{sign}(\sum_{i=0}^{d} w_i x_i)$.

Naive Bayes can be viewed as linear classifier.

Linear Discriminant Analysis

- In LDA, we are again doing binary classification with $y \in \{-1, 1\}$ based on feature vector $\mathbf{x} = (x_1, \ldots, x_d)$. The generative assumptions however are difference.
- Let $\pi_1 = P(Y = 1)$ and $\pi_{-1} = P(Y = -1)$. We assume that $P(X|Y)$ is Gaussian with the same covariance matrix Σ but different means $\mu_{-1}, \mu_1 \in \mathbb{R}^d$ for the two classes

$$P(X = \mathbf{x}|Y = y) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{\left(-\frac{1}{2}(\mathbf{x}-\mu_y)^T\Sigma^{-1}(\mathbf{x}-\mu_y)\right)}.$$



- Recall the optimal Bayes classifier

$$h_{Bayes}(\mathbf{x}) = \arg\max_{y\in\{-1,1\}} P(Y=y)P(X=\mathbf{x}|Y=y)$$

- The classifier will predict $h_{Bayes}(\mathbf{x}) = 1$ if

$$\log\left(\frac{P(Y=1)P(X=\mathbf{x}|Y=1)}{P(Y=-1)P(X=\mathbf{x}|Y=-1)}\right) > 0.$$

- This is called the log likelihood ratio. In this case it is

$$\frac{1}{2}(\mathbf{x}-\mu_{-1})^T\Sigma^{-1}(\mathbf{x}-\mu_{-1}) - \frac{1}{2}(\mathbf{x}-\mu_1)^T\Sigma^{-1}(\mathbf{x}-\mu_1) + \log(\pi_1/\pi_{-1})$$
$$= \underbrace{(\mu_1-\mu_{-1})^T\Sigma^{-1}}_{w^T}\mathbf{x} + \underbrace{\frac{1}{2}\left(\mu_{-1}^T\Sigma^{-1}\mu_{-1} - \mu_1^T\Sigma^{-1}\mu_1\right) + \log(\pi_1/\pi_{-1})}_{w_0}$$

which is also a linear classifier.

Discriminative Models

- Both Naive Bayes and LDA end up giving us linear classifiers.
    - But learning method is to do density estimation to learn $p(x, y)$, then use Bayes rule to obtain the linear classifier.
- In contrast, we previously defined supervised learning as minimizing the expected loss for some loss function.
    - For example, in logistic regression, we compose a linear function with the logistic (also known as sigmoid) function to represent the conditional probability of y given $\mathbf{x}$, $p(y|\mathbf{x}) = \frac{1}{1+\exp(-y\mathbf{w}^T\mathbf{x})}$. We then try to minimize the expected log loss $E[-\log p(y|\mathbf{x})] = E[\log(1 + \exp(-y\mathbf{w}^T\mathbf{x}))]$.
- This approach of directly optimizing for the loss that we are interested in is called **discriminative learning**.
- Which is better for learning the linear function: logistic regression, Naive Bayes, or LDA?

Discriminative Learning in Practice

Discriminative learning:

Aim: find h to min $L_D(h)$

Use $L_S(h)$ to approx $L_D(h)$

+ Regularize so that h approx min $L_D(h)$

Generative learning:

Learn $p(x,y)$ then

do inference to get $p(y|x)$

Exercise 4: In this experiment, we plot the learning curves for data generated from two isotropic Gaussians with centers $(1, 1, 1, 1), (1, 1, 1, 1)$ and standard deviations of 1. In this case, both models are able to represent the optimal decision boundary and we would like to see which method converges faster.

Comment on the learning curves. Give your reasons for the differences you observe.

- Generative model with correct model often converges faster than corresponding discriminative model
- In this case, both should converge to same accuracy

Exercise 5: We will use the 20 Newsgroup dataset in the experiment. According to the website `http://qwone.com/~jason/20Newsgroups/`: *The 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. To the best of our knowledge, it was originally collected by Ken Lang, probably for his paper Newsweeder: Learning to filter netnews, though he does not explicitly mention this collection. The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering.*

- Run the experiment comparing the learning curves of Naive Bayes and logistic regression in classifying documents from 'alt.atheism' and 'comp.graphics'.
- Replace 'comp.graphics' with 'soc.religion.christian' and rerun.

Comment on the learning curves. What do you think are the reasons for the differences you observe?

- Naive Bayes depends on model being nearly correct. ~~Okay~~ Not too bad in one case, not as good in another
- Logistic regression tries to do the best it can for both cases. Better than NB in one case
- NB gets to its final accuracy faster.

Discriminative Vs Generative Classifiers

- Which is better?
- Discriminative classifiers are more robust against misspecification of the model.
  - If the model is wrong, they will usually converge to the best approximation within the model class as the data size increases. If the model is correct, they will normally converge to the optimal solution.
  - On the other hand generative classifiers depend on the model being correct. If the model is correct, it converges to the optimal solution. But if the model is incorrect, no guarantee of getting a best approximation within the model class.
  - It is often difficult to get correct models in practice.
- It is usually easier to add features to discriminative models.
  - Adding features adds to the approximation capabilities of the discriminitive model (at some risk of overfitting if there is not enough data).
  - For generative model, we need to try to find the correct generative model for the feature, which can be difficult.

- Generative classifiers have advantages in some situations.
  - It is sometimes computationally cheaper to train, e.g. Naive Bayes mostly requires only counting.
  - It may require fewer training examples to converge for some models, e.g. Naive Bayes converges faster than logistic regression [7].
  - It handles missing variables naturally. In doing inference, missing variables are marginalized away. In discriminative models, there is no standard way for handling missing variables.
  - It also handles unlabeled training data naturally (marginalizing the missing labels away). Learning with labeled and unlabeled data is called **semi-supervised learning**.

Outline

1. Supervised Learning
2. Unsupervised Learning
3. Discriminative/Generative Models
4. Model Selection
5. Features

- Most algorithms have parameters to tune in order to get good generalization performance.
- Many algorithms optimize a regularized loss function consisting of the empirical risk plus a regularization term that penalizes for complex functions example: Ridge regression

$$L_S(\mathbf{w}) + \lambda R(\mathbf{w}) \quad \frac{1}{m}\sum_{i=1}^{m}(y_i - w^T x_i)^2 + \lambda\|w\|^2$$

  for some function R, e.g. when doing MAP estimation.
  - One parameter to tune is the value of λ.
- Structural risk minimization (to be covered later) is able to provide performance bound for some of these methods.
  - The bounds suggest the form of the objective function.
  - Provides further reassurance that the algorithm is likely to be well-bahaved.
  - But in practice, the bounds are too loose to be used directly as the objective function – usually the parameter λ needs to be tuned.

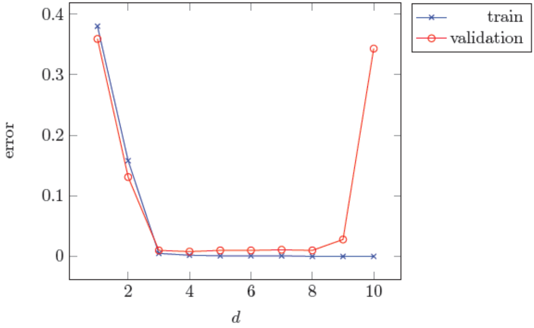


Figure from SSBD. Using validation set to select degree of polynomial in polynomial regression.

- In practice, it is common to leave some (randomly selected) data out of the training set to use to select the model.
- This is called the validation set. Also called development set

- The validation set is used to evaluate and select the different models.
    - For continuous parameters, a few discrete values may be selected for evaluation.
    - Grid search is sometimes done, with a coarse grid first, then zooming into the correct region to do a finer grid.
- The validation error is a reasonable estimate for the true error if
    - the validation set is large enough
    - the number of models being evaluated is not too large
    - the training data is representative of the test distribution
- After the model is selected, the validation set is often combined back into the training set and use to retrain the selected model with all the data.
- In solving a problem, typically we split the data into three parts: a training set (e.g. 60%), a validation set (e.g. 20%), and a test set (e.g. 20%).
- The test set is used to do the final evaluation of performance.

- If there is not enough data to train a good model after setting aside a validation set, k-fold cross validation is often used.
    - The training data is partitioned into k sets of size m/k.
    - The following is done for each partition
        - Use the partition to evaluate while the rest of the data for training
    - The validation errors are averaged and used to select the best model.
- In the extreme case, only one example is left out each time, with the rest of the training set used for training.
    - This is called leave-one-out cross validation.

Exercise 6

In this experiment, we look at the effect of the number of folds and the training set size when doing cross validation.

- On the digits data set, run 10-fold cross validation
- Change the number of folds to 2-fold and observe the effect.

Comment on the outcomes of the experiments.

- 10-fold gives more accurate estimate of accuracy of training with all training data

What to do when learning fails

- If training error is large.

Do error analysis - look at validation error instead to find better features, architectures, etc.

  - Possibly underfitting. Look to reduce approximation error. Look for more useful features, use more powerful approximators.
  - Check that your are not regularizing too much.
  - It is also possible that the approximating function class is powerful enough, but the optimization algorithm is failing. Usually less likely with convex problems. For non-convex problems
    - Tuning your optimization parameters may help, e.g. learning rate in neural networks
    - Better initialization may help, particularly if you can use prior information to get a good starting point.
- If validation error is high but training error is low
  - Possible overfitting. Look to reduce the estimation error. Do feature selection, or more regularization, etc. If possible, get more data.

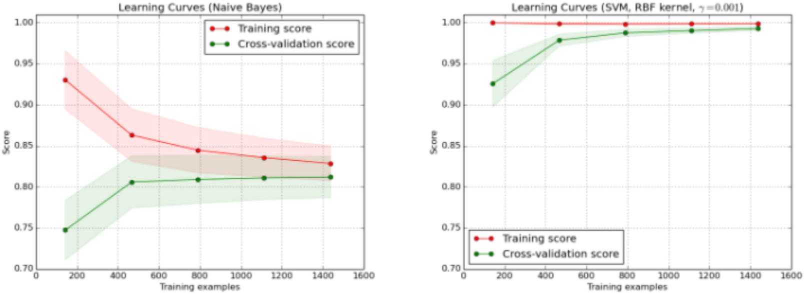


Figure from [5]

- The learning curve shows the validation and training score for varying number of training samples.
    - Plotting the learning curve can be helpful in telling us whether getting more data will likely be helpful for an algorithm.

Outline

1. Supervised Learning
2. Unsupervised Learning
3. Discriminative/Generative Models
4. Model Selection
5. Features

How to Win on Kaggle [2]

For most Kaggle competitions the most important part is feature engineering, which is pretty easy to learn how to do. (Tim Salimans)

The features you use influence more than everything else the result. No algorithm alone, to my knowledge, can supplement the information gain given by correct feature engineering. (Luca Massaron)

Feature engineering is certainly one of the most important aspects in Kaggle competitions and it is the part where one should spend the most time on. There are often some hidden features in the data which can improve your performance by a lot and if you want to get a good place on the leaderboard you have to find them. If you screw up here you mostly can't win anymore; there is always one guy who finds all the secrets. However, there are also other important parts, like how you formulate the problem. Will you use a regression model or classification model or even combine both or is some kind of ranking needed. This, and feature engineering, are crucial to achieve a good result in those competitions. There are also some competitions where (manual) feature engineering is not needed anymore; like in image processing competitions. Current state of the art deep learning algorithms can do that for you.
(Josef Feigl)

- Feature engineering involves using domain knowledge to define features that may be useful. For example, in text/NLP applications, may use
  - unigrams, bigrams, trigrams, constructed from the word sequences.
  - parts of speech, perhaps components of the parse trees.
  - specially selected words from dictionary.
  - other NLP components such as named entity detector, sentiment analysis, etc.
- Then can do feature selection.

word embedding

- May also do feature generation by unsupervised learning.
- In certain domains, e.g. vision applications, learning the features e.g. using neural networks from raw data using supervised learning may work better.

Feature Selection and Generation

- We assume that instance space $\mathcal{X}$ that we are given is a subset of $\mathbb{R}^n$.
- This may be the raw input that we get, e.g. pixels in an image. Or it may be the output of a set of variables or features of the problem domain.
- For simplicity (and to be consistent with the term feature selection), we call each input component a feature.
- We may want to select only a subset of features
  - Sparse solutions may be more interpretable, e.g. we want to know which genes are likely responsible for a disease.
  - Using fewer features would speed up prediction.
  - Feature selection can improve generalization for some learning algorithms.
- We may also want to transform the features by weighting, normalization, and other transformations.
- We may want to construct other features out of the current features.

Filter

- Assess each feature independently of other features.
- Select k features that achieves the highest score.
- For example, Pearson correlation coefficient is defined as:

  $$\frac{cov(X_j, Y)}{\sqrt{var(X_j)var(Y)}},$$

  where *cov* is the covariance and *var* is the variance, and we want to maximize the correlation coefficient.

  - Ranks the features according to how well the single variable X_j can minimize the mean square error when used in a linear predictor:

    $$\min_{a,b\in\mathbb{R}} \frac{1}{m}\sum_{i=1}^{m}(ax_{ij}+b-y_i)^2.$$

- For classification, *mutual information* is often used:

$$
\begin{aligned}
I(X_i;Y) &= \sum_{x_i}\sum_{y} p(x_i,y)\log\frac{p(x_i,y)}{p(x_i)p(y)} \quad \begin{array}{l}\leftarrow x_i, y \text{ dependent}\\[1ex] \leftarrow x_i, y \text{ independent}\end{array}\\
&= \sum_{x_i}\sum_{y} p(x_i,y)\log\frac{p(\cancel{x_i})p(y|x_i)}{p(\cancel{x_i})p(y)} \quad \leftarrow \text{chain rule}\\
&\qquad\qquad \searrow \text{chain rule}\\
&= \sum_{x_i}\sum_{y} p(x_i)p(y|x_i)\log p(y|x_i) - \sum_{x_i}\sum_{y} p(x_i,y)\log p(y)\\
&= \sum_{x_i} p(x_i)\sum_{y} p(y|x_i)\log p(y|x_i) - \sum_{y}\log p(y)\sum_{x_i} p(x_i,y)\\
&\qquad\qquad\qquad \swarrow \text{entropy}\\
&= -\sum_{x_i} p(x_i)H(Y|x_i) - \sum_{y} p(y)\log p(y) \quad \leftarrow \text{marginalize}\\
&= -H(Y|X) + H(Y)\\
&= H(Y) - H(Y|X).
\end{aligned}
$$

- $H(Y) - H(Y|X)$ is the difference between the entropy of the label Y and the conditional entropy of Y given input X, and is also called the *information gain*.
- For feature selection, we rank features according to information gain, and select the k features with the largest information gain.
- Another commonly used feature selection method is the χ^2 (chi square) feature selection method which tests whether the feature is independent of the label.

Wrapper

- Filter approaches do not take into account dependence among the features.
- Wrapper approaches search for subset of variables that will perform well. Often, the learning method is treated as a black box that is optimized by searching for a subset of variables.
- Common greedy methods include:
  - *Forward selection:* Features are progressively added as learning is done. For example, the Viola-Jones face detector is trained using Adaboost, which may be considered as a greedy method that adds one feature at a time[1].
  - *Backward elimination:* In backward elimination, we start with all variables and progressively eliminate the least promising ones.
- Combinations of forward selecton and backward elimination and other search methods are also used in practice.

[1] We will look at Adaboost in more detail later in the course

Viola-Jones Face Detector

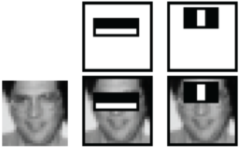


Figure 10.2 The first and second features selected by AdaBoost, as implemented by Viola and Jones. The two features are shown in the top row and then overlaid on a typical training face in the bottom row. The first feature measures the difference in intensity between the region of the eyes and a region across the upper cheeks. The feature capitalizes on the observation that the eye region is often darker than the cheeks. The second feature compares the intensities in the eye regions to the intensity across the bridge of the nose.

- Figure from SSBD. For real-time face detector (detect face in your camera while you are taking photo), need to be very fast.
- Viola-Jones face detector achieves the speed by selecting and using very few features and a cascade where subwindows that are labeled by simpler classifiers as potentially positive are further processed using a more complex classifier.

Sparsity-Inducing Norms

- We can formulate the feature selection problem as:

$$\min_{\mathbf{w}} L_S(\mathbf{w}) \text{ s.t. } ||\mathbf{w}||_0 \leq k,$$

optimization problem

where $||\mathbf{w}||_0 = |\{i : w_i \neq 0\}|$.

  - $||\mathbf{w}||_0$ is often called the ℓ_0 norm, even though mathematically, it is not a norm.

- This is often computationally hard, so we often relax the ℓ_0 norm into a ℓ_1 norm to get

$$\|w\|_1 = \sum_{i=1}^{d} |w_i|$$

$$\min_{\mathbf{w}} L_S(\mathbf{w}) \text{ s.t. } ||\mathbf{w}||_1 \leq k,$$

  - If L_S is convex, then this is a convex optimization problem and can be solved efficiently.

- A related and more common way to encourage a sparse solution is to use the ℓ_1 norm of the weight as a regularizer,

$$\min_{\mathbf{w}}(L_S(\mathbf{w}) + \lambda||\mathbf{w}||_1).$$

  - Again, this is convex if L_S is convex, e.g. in logistic regression.
- Adding ℓ_1 regularization to linear regression with the squared loss gives us the LASSO algorithm,

$$\min_{\mathbf{w}}(\frac{1}{2m}||X\mathbf{w} - \mathbf{y}||^2 + \lambda||\mathbf{w}||_1).$$

- Can derive Lasso as MAP with each w_i independently distributed with Laplacian prior $\Pr(w_i) = \frac{\lambda}{2}\exp(-\lambda|w_i|)$.
- Under some assumptions, can recover the true sparse solution – studied under compressed sensing.

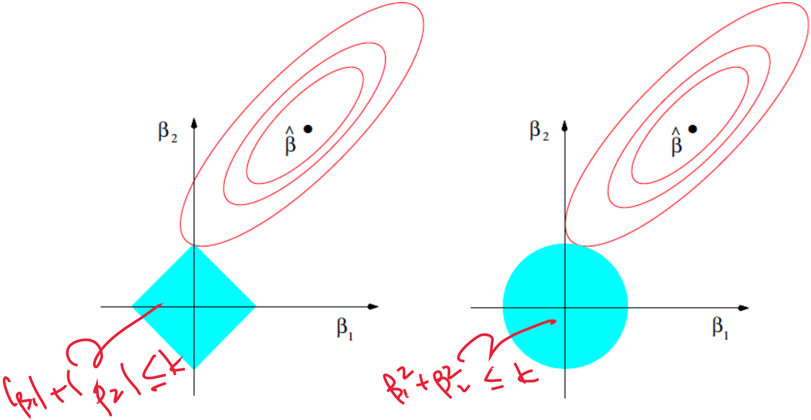


Figure from [3]. Lasso vs ridge regression.

Unlike the ℓ_1 norm, the ℓ_2 norm does not induce sparse solutions.

Exercise 7:

Regularizing using the ℓ_1 norm rather than the ℓ_2 norm induces sparsity and can serve as a method for feature selection. This is often called Lasso. The optimization objective for Lasso is $\frac{1}{2m}||y - Xw||_2^2 + \alpha||w||_1$. We will compare the two methods using polynomial regression with a 10th degree polynomial using noisy training data from a linear function to see if we get a sparse solution.

Comment on the outcome of the experiment.

- Lasso zeroes out most coeffs
- Ridge regression does not have sparse solution in terms of features

Feature Transformation and Normalization

Example: Ridge regression with x_1 and x_2 in meters

True fn: $w_1 x_1 + w_2 x_2$ with $w_1 = w_2 = 1$

If instead, receive x_2 in kilometers, true fn becomes $w_1 = 1$, $w_2 = 10^3$

- Some simple transformations on the features can often affect performance.
- For example, if the scales of the features are very different, a feature that has very small range may need very large weights to have an effect.
  - With regularization, this feature may be overpenalized as it would cost a lot to use a large weight.
  - In this case, normalizing the features may be helpful.

problems in optimization

generalization

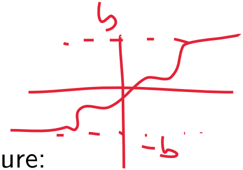
Let $\mathbf{f} = (f_1, \ldots, f_m) \in \mathbb{R}^m$ be the value of feature f over the m training examples and $\bar{f} = \frac{1}{m}\sum_{i=1}^m f_i$ be the empirical mean of the feature.

Some common transformations:

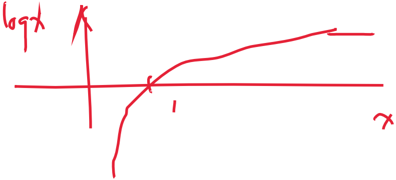
- **Centering:** Transform the feature to have zero mean: $f_i \leftarrow f_i - \bar{f}$.
- **Unit Range:** Set the range of the feature to $[0, 1]$. Let $f_{\max} = \max_i f_i$ and $f_{\min} = \min_i f_i$. Then the transformation is: $f_i \leftarrow \frac{f_i - f_{min}}{f_{\max} - f_{\min}}$.
- **Standardization:** Transform the feature to have zero mean and unit variance. Let $\nu = \frac{1}{m}\sum_{i=1}^m (f_i - \bar{f})^2$. Then the transformation is: $f_i \leftarrow \frac{f_i - \bar{f}}{\sqrt{\nu}}$.

variance

std dev



- **Clipping:** Clips the high low values of a feature: $f_i \to \text{sign}(f_i)\max(b, |f_i|)$ where b is a user specified parameter.
- **Sigmoidal transformation:** Apply sigmoid function to the feature: $f_i \leftarrow \frac{1}{1+\exp(bf_i)}$ where b is a user specified parameter.
- **Logarithmic Transformation:** Used when the difference for small values, e.g. between 0 and 1, is more important than the difference for large values, e.g. 1000 and 1001: $f_i \leftarrow \log(b + f_i)$, where b is a user specified parameter.

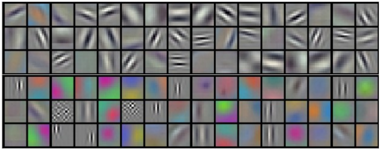


Transformation may be domain specific. For example, in text processing, a word that appears many times in a document may be important to the document, but if the word appears in all documents, it is unlikely to be useful in telling documents apart. This motivates the *term frequency-inverse document frequency* (TF-IDF) representation.

- Let $f(w,d)$ denote the number of times word w appears in document d.
- Let $g(w)$ denote the number of documents where w appears in within the document collection with m documents.
- Then $\mathrm{tf\text{-}idf}(w) = f(w,d)\log\frac{m}{g(w)}$.

Feature Learning

- Instead of selecting a subset of predefined features, we can also learn a feature mapping $\psi : \mathcal{X} \mapsto \mathbb{R}^d$ which maps instances in $\mathcal{X}$ to d dimensional feature space.
- Such learned features can be more informative that the original features for some domains, e.g. pixels in images are not very informative but a feature that can tell if a set of pixels comes from a part of a face would be more useful.
- One way would be to learn the features directly as part of supervised learning, e.g. by using hidden units in neural networks.



Features learned by AlexNet.

Image source: `http://cs231n.github.io/convolutional-networks/`.

- But supervised learning requires labeled data, which can often be expensive.
- If we have a lot of unlabeled data, may be useful to do unsupervised feature learning, sometimes called *dictionary learning*.
- Also self-supervized learning. Construct supervized learning tasks from unlabelled data for purpose of learning useful features. e.g remove part of an image and use remaining parts to predict removed parts.

Dimension Reduction

- One approach for dictionary learning is to construct an *autoencoder*.
    - An autoencoder constructs a function pair: an encoder $\psi : \mathbb{R}^d \mapsto \mathbb{R}^k$ and a decoder $\phi : \mathbb{R}^k \mapsto \mathbb{R}^d$.

      if $d >> k$, then dim reduction

    - The goal of learning is to minimize the reconstruction error: $\sum_i ||\mathbf{x}_i - \phi(\psi(\mathbf{x}_i))||^2$ is often used as the reconstruction error.

- Continued ...
  - Principal Component Analysis (PCA) is an example where $k < d$. In PCA, we map the input into a smaller number of dimensions (dimensionality reduction) in a where that the reconstruction error using the smaller number of dimensions is minimized.
  - In PCA, $\psi(\mathbf{x}) = V_k^T \mathbf{x}$ is a linear transformation. The decoder $\phi(\mathbf{u}) = V\mathbf{u}$ is also a linear transformation.
  - In general, k need not be less than d. It is also possible to use non-linear transformations such as deep neural networks for ψ and ϕ.

Reading

Some slide material are taken directly from SSBD.

1. SSBD section 9.3 on logistic regression
2. SSBD chapter 24.1 on maximum likelihood
3. SSBD chapter 24.2 on naive Bayes
4. SSBD chapter 24 on discriminative vs generative learning
5. SSBD Chapter 11 on model selection
6. SSBD Chapter 25 on feature selection and generation.

References I

[1] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] *David Wind, Learning from the Best*. [Online: http://blog.kaggle.com/2014/08/01/learning-from-the-best/, accessed July 2017].

[3] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. Springer series in statistics New York, 2001.

[4] Tero Karras et al. "Progressive growing of gans for improved quality, stability, and variation". In: *arXiv preprint arXiv:1710.10196* (2017).

References II

[5] *Learning curve*. [Online: http://scikit-learn.org/0.15/auto_examples/plot_learning_curve.html, accessed July 2017].

[6] *Logistic Regression*. [Online: https://en.wikipedia.org/wiki/Logistic_regression, accessed July 2017].

[7] Andrew Y Ng and Michael I Jordan. "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes". In: *Advances in neural information processing systems*. 2002, pp. 841–848.