06_Feature_Reduction_and_Classifier_Combination

1. High-Dimension Space

Problem of High-Dimensional Space

There are a lot of problem when using a very high-dimension space, three problems are stressed in the course

- For a high-dimensional space, **more samples are needed** to catch same proportion of data
- For a given number of samples, in a high-dimensional space, more data will become **boundary points** (no points pareto outside of it)
- Come to a higher-dimension space, points tend to have **equal distances**

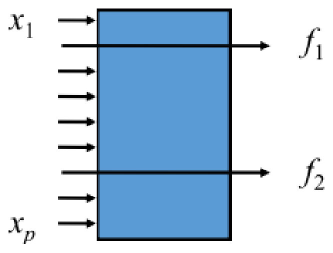
Method: Feature Reduction

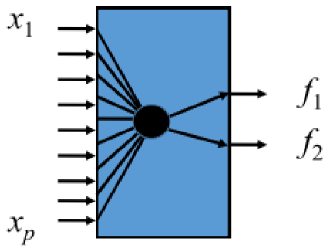
To be concluded, for a high-dimensional space, we need sufficient many sample points. Otherwise we need to **drop some features**.

There are mainly two ways to drop features

- **Feature Selection**: **select** d out of p measurements

- **Feature Extraction**: **map** p measurements to d measurements





Notes: Think of selection and extraction as finding a **mapping,** for linear cases, that is find a linear matrix A then transform features x to Ax.

For General procedure of Feature Reduction, we need to find:

- **Criterion** Function
- **Optimization** or search algorithm

For criterion, there are two main trends:

- **Optimal Criterion**: final performance of the entire system
- **Approximate performance**: some performance that intuitively indicate final performance.

In this lecture ,we mainly focus on how to optimize feature before we actually train something based on our data. That means, **we mainly focus on the second trend.**

2. Feature Extraction

Here we will introduce two classical Linear Feature Extractors

- Unsupervised: **Principal Component Analysis (PCA)**
- Supervised: **Linear Discriminant Analysis (LDA)**

Unsupervised: PCA

Intuition

There are three intuition behind PCA:

- Retain as much [total] variance as possible
- Make projected data uncorrelated
- Minimize squared reconstruction error

Model

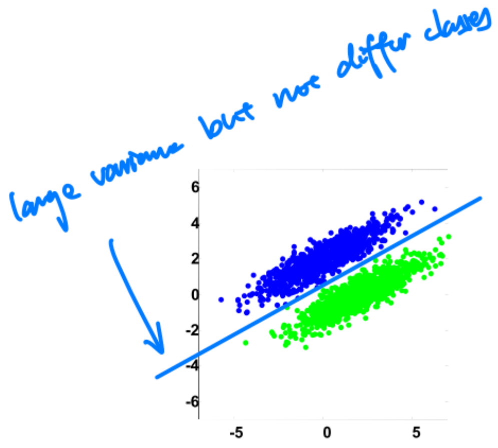
Here we will use a reduces to 1D example to illustrate the model of PCA.

Let a be a projection vector that reduces to 1D and our data has covariance matrix C. Then $a^T Ca$ equals to variance of the projection of data.

Then we assume the constraint $||a||^2 = 1$, then we can solve it (for example use Lagrangian Multiplier: $a^T Ca - \lambda \left(\|a\|^2 - 1 \right)$

Remarks

- Global and Linear
- Need estimate covariance well
- Criterion is not necessarily related to the goal



Kernel PCA

For some cases with apparent nonlinear boundary, we can use **kernelized PCA**

Supervised: Linear Discriminant Analysis

Scatter Matrices

First we will use following symbols:

- $m, S_T = \Sigma$: mean ad covariance of all samples
- m_i, Σ_i: mean and covariance of class i

Then based on these symbols, we can further define:

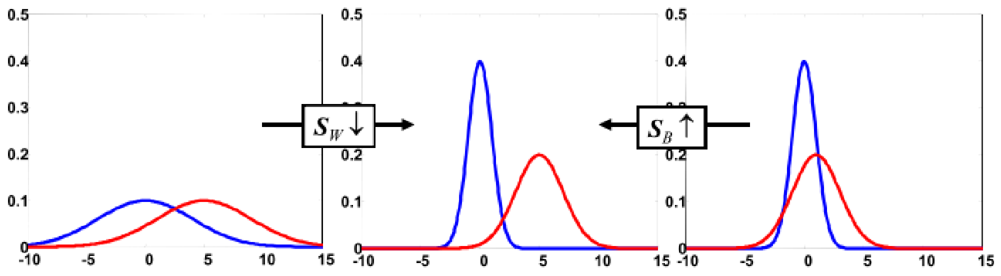
- **Within Scatter:** $\boldsymbol{Sw} = \sum_{i=1}^{C} \frac{n_i}{n} \Sigma_i$
- **Between Scatter:** $\boldsymbol{SB} = \sum i = 1^C \frac{n_i}{n} \left(\boldsymbol{mi} - \boldsymbol{m} \right) \left(\boldsymbol{mi} - \boldsymbol{m} \right)^T$

And we further have: Total scatter (Σ) equals sum of within and between

For S_W and S_B, we have:

- S_W: "average class width"; the smaller the better;

- S_B: "average distance between class means"; the larger the better



Model

Here we will use a reduces to 1D example to illustrate the model of PCA.

Find projection vector a such that classes are **maximally separated**, that is, choose a to maximize the **Fisher Criterion**

$$J_F(a) = \frac{a^T S_B a}{a^T S_W a}$$

We can use Eigenanalysis to solve it

Combination of Feature Extraction

One interesting property of feature extraction is: **the combination of feature extraction is still a feature extraction** (easily to understand).

3. Feature Selection

In previous chapters, we has already seen one feature selection algorithm: the **sparsity regularization**

Criterions

Here we will mainly discuss approximate performance predictors: calculate easily to measure that give indication of real performance.

Probabilistic Criteria

Probabilistic Criteria use **probabilistic distance**, which expresses distance between two distributions. But because it often needs estimates of class-conditional densities, it is potentially difficult and expensive.

Scatter Matrices

We can still use Scatter Matrices we have used. For judgement we can use following options:

$$\begin{aligned} J_1 &= \operatorname{trace}\left(\boldsymbol{S}_W + \boldsymbol{S}_B\right) = \operatorname{trace}(\Sigma) = \operatorname{trace}\left(\boldsymbol{S}_T\right) \\ J_2 &= \operatorname{trace}\left(\boldsymbol{S}_B / \boldsymbol{S}_w\right) \\ J_3 &= \det(\Sigma) / \det\left(\boldsymbol{S}_w\right) \\ J_4 &= \operatorname{trace}\left(\boldsymbol{S}_W\right) / \operatorname{trace}\left(\boldsymbol{S}_B\right) \end{aligned}$$

Mahalanobis Distance

- Judge the distance between two classes

$$D_M = (m_1 - m_2)^T X^{-1}(m_1 - m_2)$$

- For multi-class cases: take sum or minimum
- When come to 1D case, we will have **Fisher Criterion**

$$J_F = \frac{(m_1 - m_2)^2}{\sigma_1^2 + \sigma_2^2}$$

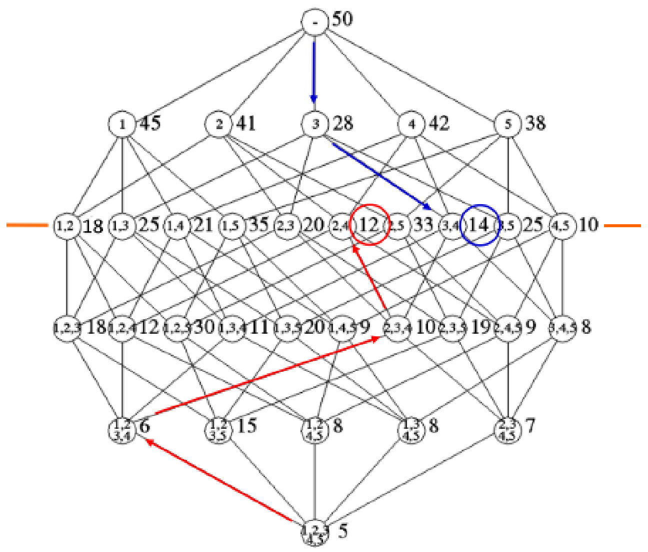
Illustration

> https://zhuanlan.zhihu.com/p/46626607

Search Algorithms

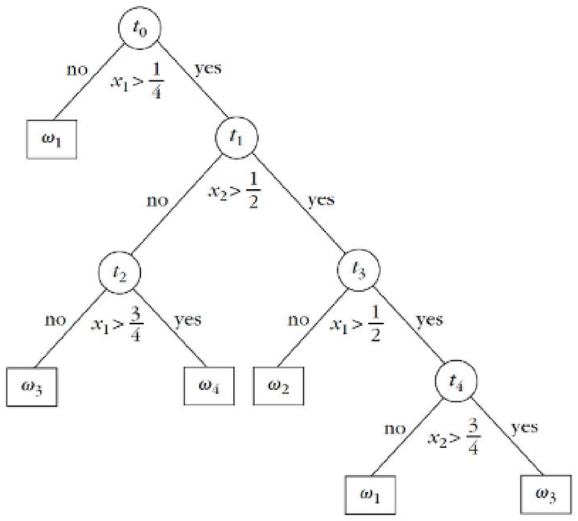
There are several intuitive methods:

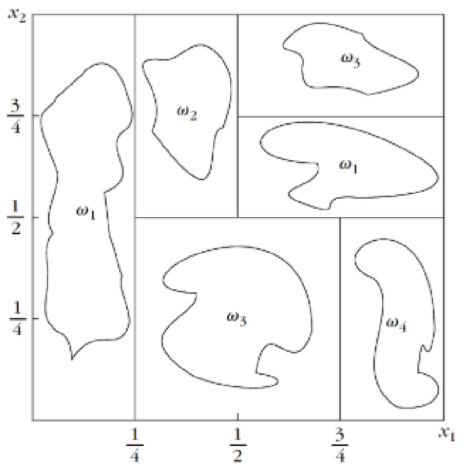
- Look at all possible subsets
- select one feature at a time
- **Forward Selection**: Start with empty feature set, One at a time, keep adding feature that gives best performance considering entire chosen feature set
- **Backward Selection:** Start with full feature set, One at a time, keep feature that gives best performance considering entire chosen feature set
- **Plus-l-take-away-r**: Keep adding best l and removing worst w
- **Branch and Bound**: Works well when criterion is monotonic in number of features d
- **Floating Search**



Decision Tree and Bagging

A Decision Tree Example





Ingredients

- Set of **Questions** to be asked (can be nonlinear)
- **splitting criterion**: there are many ways to split a decision tree based on given data, which criterion should we follow to split a tree
- **stop-splitting rule**: when should we stop splitting a tree?
- **class assignment rule** in leaves (most times, we cannot generate the whole tree to classify every object correctly, which always means overfitting)

Trade-Off

- **size:** too large → overfitting, too small→not enough
- **variance:** small changes in data may lead to large changes in tree

Bagging

Bagging is one way to reduce this variance, it is a **combining methodology**

- Bootstrap data sample to train different classifier
- average classifier outcomes

Bootstrap Sampling:

Bootstrap Sampling is a method that involves drawing of sample data repeatedly with replacement from a data source to estimate a population parameter

4. Classifier Combination

Even if one classifier performs better than other, former may still be better in particular part of feature space

Implementation

How to generate different **base classifiers:**

- different type of classifiers
- different object sampling
- different feature samplings

Fixed Combination

A lot rules can be used: averaging rule, max rule, min rule, product rule, median rule, voting rule, etc

Things to be combined can be class labels, posteriors, rankings or maybe other classifier outputs.

Generally, theoretical analysis of fixed combination is very hard

Condorcet Paradox

Assume there are 3 classifier:

Classifier 1 prefers class A to B to C

Classifier 2 prefers class B to C to A

Classifier 3 prefers class C to A to B

Of course, every choice is as good as every other

But it is worse: choose one preferred class, say A, this leaves us with C, that is preferred by the two other classifiers...

Trained Combination

One can take outputs of base classifiers as features. And then we prepare a new Classifier trained on to of these is the combiner.

Boosting

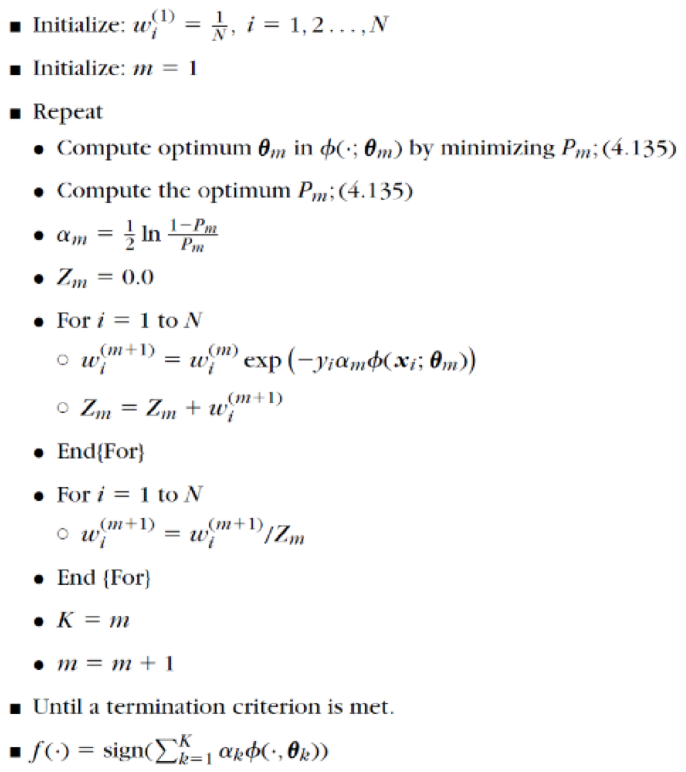
In machine learning, **boosting** is an ensemble **meta-algorithm** for primarily reducing bias, and also variance in supervised learning, and a family of machine learning algorithms that **convert weak learners to strong ones**

AdaBoosting

One important intuition behind **AdaBoosting** is sequentially train different classifiers instead of parallelly train different classifiers. **Later training should pay more attention on the area in which previous classifiers perform poorly.**

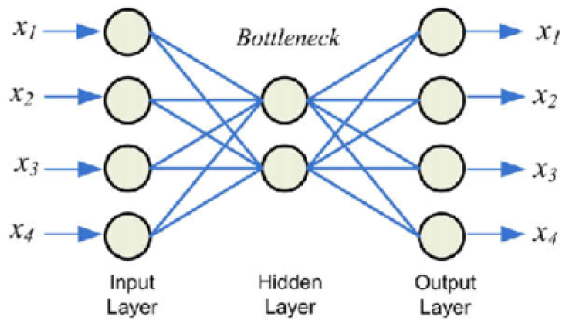
In order to do that, in each round, different data will **modify its weights** (poorly performance data will with higher weight) and different classifier will **automatically modify its weight** based on performance.

AdaBoost



```
■ Initialize: $w_i^{(1)} = \frac{1}{N}$, $i = 1, 2 \dots, N$
■ Initialize: $m = 1$
■ Repeat
    • Compute optimum $\boldsymbol{\theta}_m$ in $\phi(\cdot; \boldsymbol{\theta}_m)$ by minimizing $P_m$; (4.135)
    • Compute the optimum $P_m$; (4.135)
    • $\alpha_m = \frac{1}{2} \ln \frac{1-P_m}{P_m}$
    • $Z_m = 0.0$
    • For $i = 1$ to $N$
        ○ $w_i^{(m+1)} = w_i^{(m)} \exp\left(-y_i \alpha_m \phi(\boldsymbol{x}_i; \boldsymbol{\theta}_m)\right)$
        ○ $Z_m = Z_m + w_i^{(m+1)}$
    • End{For}
    • For $i = 1$ to $N$
        ○ $w_i^{(m+1)} = w_i^{(m+1)} / Z_m$
    • End {For}
    • $K = m$
    • $m = m + 1$
■ Until a termination criterion is met.
■ $f(\cdot) = \text{sign}(\sum_{k=1}^{K} \alpha_k \phi(\cdot, \boldsymbol{\theta}_k))$
```

5. Autoencoder: a classifier combination perspective



The autoencoder focus on a NN that can **maps high-dimensional input to lower dimension-data**, based on which, another high-dimensional output can be generated and $\text{input} \approx \text{output}$. The autoencoder can be widely used in **feature reduction**.

Notes: Actual it can be seen try to find a kind of mapping between high-dimensional input and low-dimensional representative. If we have high-dim input with very few data, any invertible mapping can be with $J = 0$, but it is always not the case. So the J always cannot be zero, so the process try to find a mapping that very closed to invertible mapping, which means preserve more information.

Summary

- High-Dimension Feature may cause a lot of problems
- Feature can be extracted
  - PCA: try to keep covariance
  - LDA: keep the ratio of distance among classes and the covariance in the classes
- Feature Selected
  - Build an intuitive criterion: probabilistic, scatter matrices, mahalanobis distance
  - Search for the best combinations
- Decision Tree
  - Consider: Size and Covariance
  - Bagging
- Feature Combinations
  - Fixed Combinations
  - Trained Combinations
    - Boosting
    - AdaBoosting
- Autoencoder do feature extraction