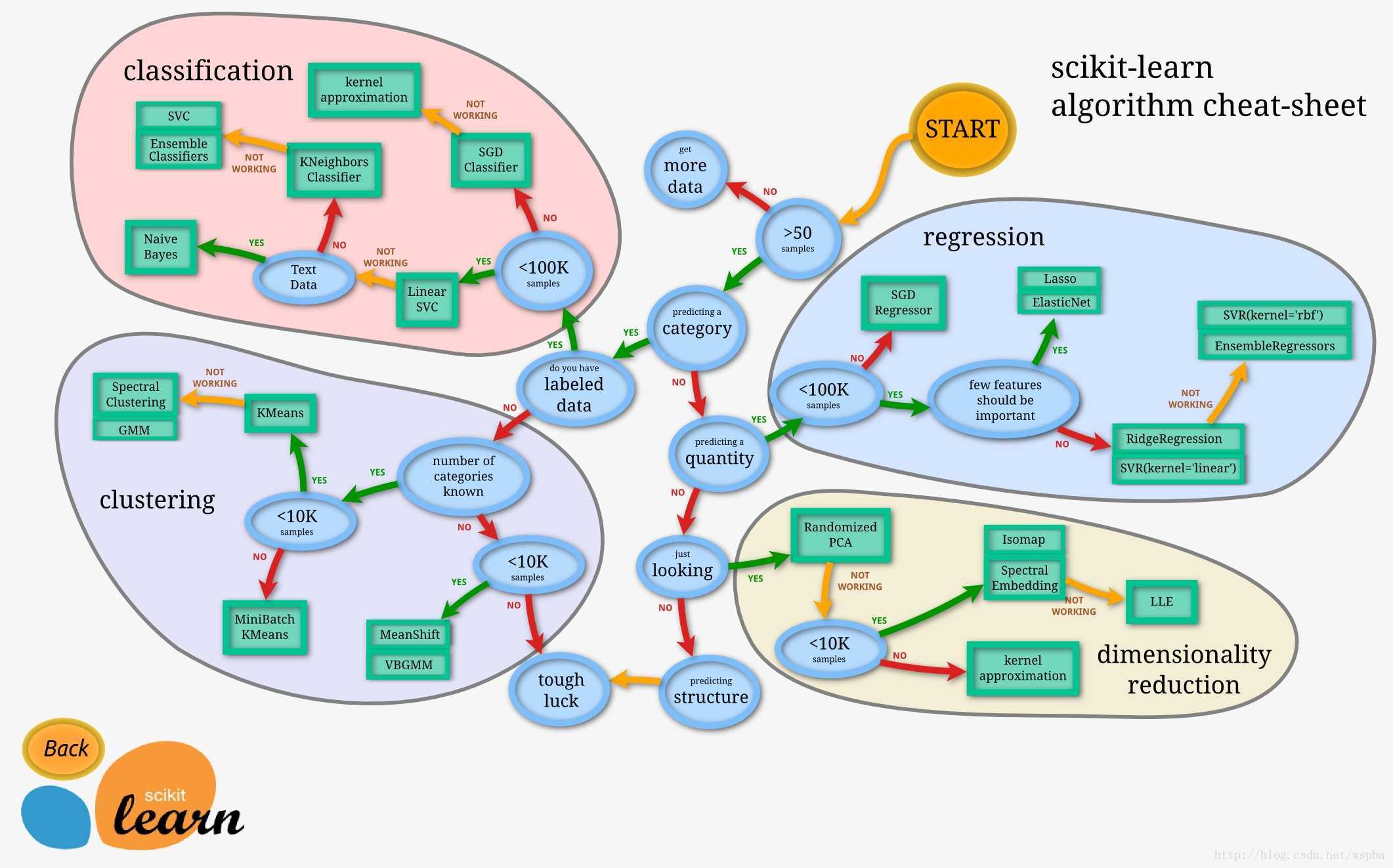
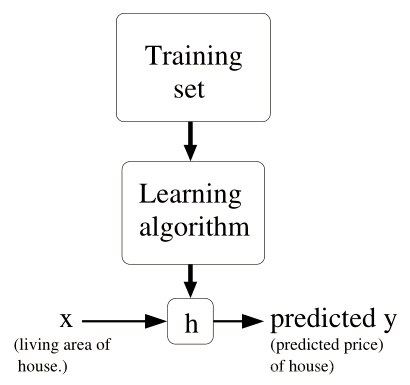
ML



**Lesson 1**

① Supervised Learning (SL)



Continuous 🡪 Regression Question (to predict)

Discrete 🡪 Classification Question (to ‘chose’ whether or not)

② Unsupervised Learning (UL)

No comparative sets or no certain answer 🡪 Clustering

③ Reinforcement Learning (RL)

‘Good Dog’ & ‘Bad Dog’ Question 🡪 Maximize ‘Reward Signal’ (decision making, like SLAM)

**Lesson 2**

① Optimization Question 🡪 least square method 🡪 ***Gradient Descent Algorithm***

② GDA associated

called ‘learning rate’, need to set by experience because too large can lead to go over the minimum, accordingly, too small can lead to convergent slowly.

(1) Batch gradient descent (e.g. linear regression)

.

**🡪** :

🡪🡪:

∴ *Batch Gradient Descent called*

(2) Stochastic gradient descent (suit large quantities of training sets)

‘accelerate’ algorithm, local minimize feature not as good as (1), just swing around it.

*(Q: How to check convergence? Any optimized solution?)*

③ Normal Equation

GDA iteration complex 🡪 Matrix-related presentation (trace properties & equation deductions)

①

🡪

②

🡪

③

🡪 ,

④

try to concise iteration by upwards definition & properties

cannot inverse?

1. features have extra linear relation 🡪delete!

2. m<n 🡪regularize training examples!

**Lesson 3**

*Outline this Lesson:*

1. Linear regression🡪locally weighted regression

↓

2. Probabilistic interpretation ( probabilities’ expression)

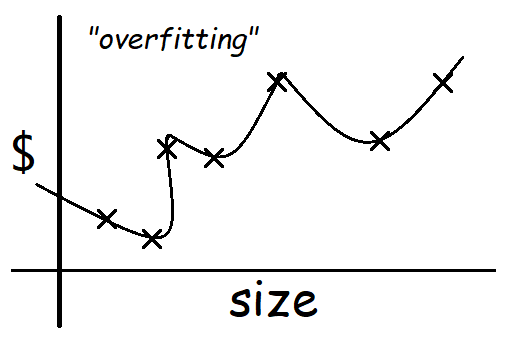
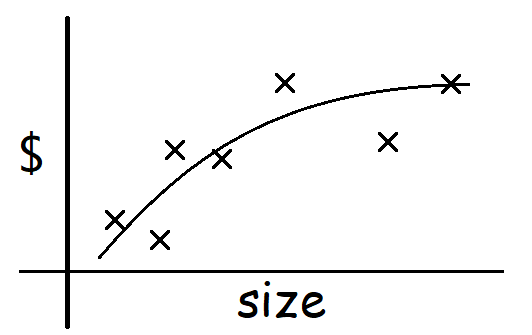
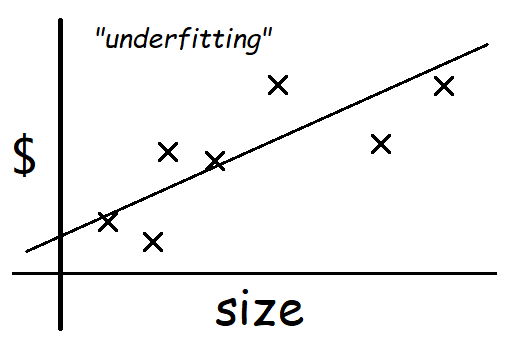
↓

3. Logistic regression 🡪perceptron algorithm

① linear regression extension (polynomial regression)

linear regression parameter can be represent like this:

🡪*underfitting* & *overfitting*



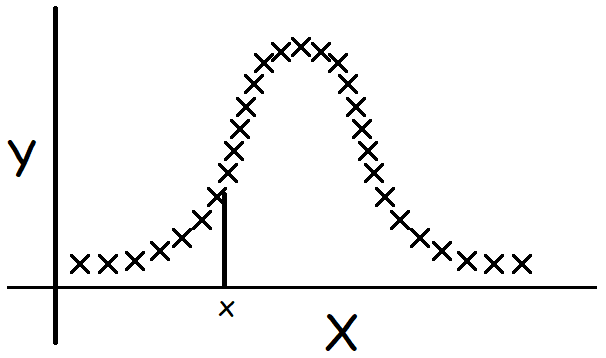
”Parametric” learning algorithm: training examples - get s - fixed set of parameters

🡪”Non-parametric” learning algorithm:

In another word, parameters rely the whole training sets, even after learning.

🡪 *locally weighted regression*(loess/lowess)

e.g.



② Probabilistic interpretation

🡪

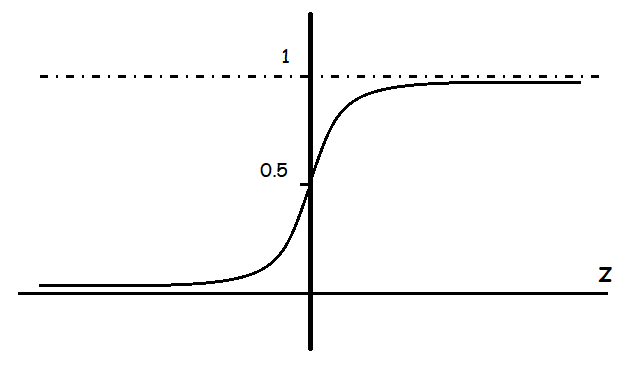
PS: means the probability of given (independent variables);

means the probability of given and parameterized by

**🡪** :

③ Logistic regression (Classification)

🡪

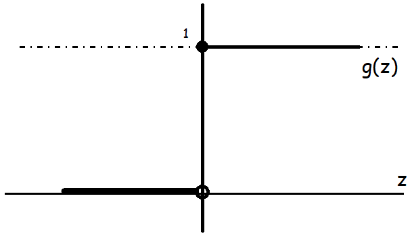


🡪 ,

🡪

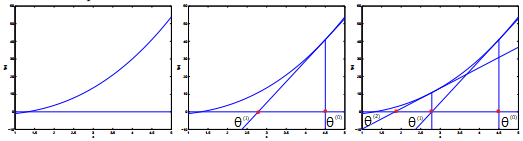
🡪

perceptron algorithm



**Lesson 4**

① Logistic regression – Newton’s Method



🡪

② Exponential Family

🡪

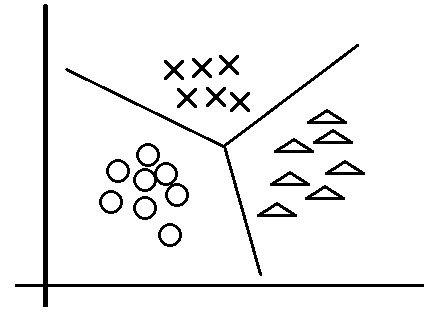
🡪

🡪

③ Generalized Linear Models (GLMs)

🡪

🡪

🡪

🡪

Softmax regression:

🡪

🡪

**Lesson 5**

*Outline this Lesson:*

1. Generative learning algorithms

2. GDA (Gaussian discriminant analysis)🡪digression: Gaussians

3. Generative & Discriminative comparison (GDA & logistic regression comparison)

4. Naive Bayes

5. Laplace Smoothing

① generative learning algorithms (e.g. logistic regression)

🡪

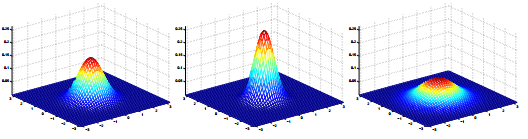
-

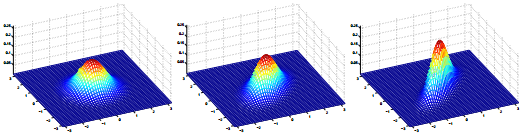
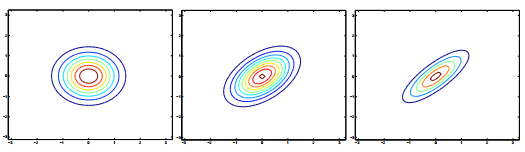
-

🡪

-

② GDA (Gaussian Discriminant Analysis)



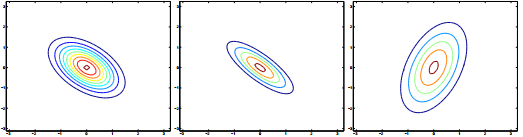


figure: different covariance of Gaussian functions (2-D)

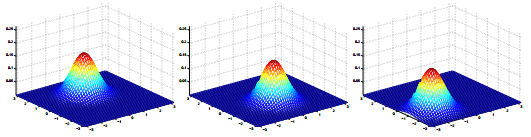
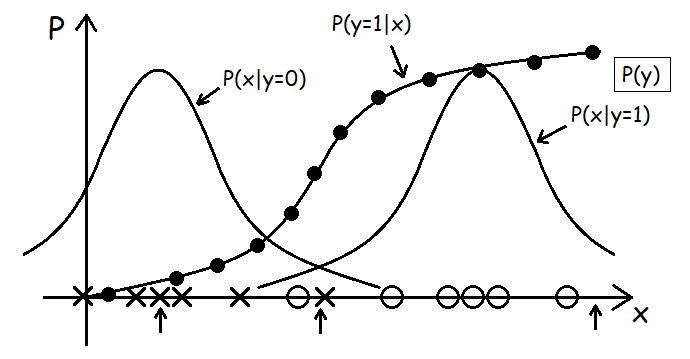


figure: different means of Gaussian functions (2-D)

🡪



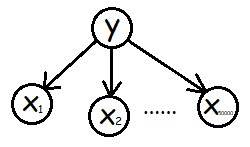
③ GDA & logistic regression comparison

It shows that logistic regression is robust for different classification questions.

GDA makes stronger modeling assumptions, and is more data efficient (i.e., requires less training data to learn “well”) when the modeling assumptions are (approximately) correct.

Logistic regression makes weaker assumptions, and is significantly more robust to deviations from modeling assumptions.

④ Naive Bayes (e.g. text classification: anti-spam)



⑤ Laplace Smoothing

When a brand new word (e.g. “NIPS”, which is the 30,000th word in dictionary) inputs, the prior probabilities are as follows:

To avoid this problem, we use Laplace Smoothing to replace estimate with

**Lesson 6**

*Outline this Lesson:*

1. Naive Bayes – event models

2. Neural Networks (Intro.)

3. Support Vector Machines

① event models

called *“Multi-variate Bernoulli* *event model”*

(hererepresents one of samples,is one of words in dictionary whether inor not)

*without using number information?* 🡪

*Multinomial event model*

e.g.



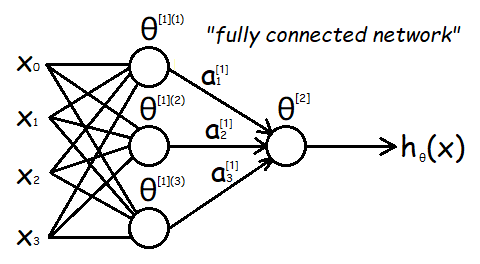
back to primal example, we can change the model with

(is a vector with different length in emails,represents the number where in dictionary)

🡪

Because Naive Bayes algorithm is built by Bernoulli or Multinomial (i.e. or other distributions), it still belongs to exponential family (GLMs), so finally, it is still a *linear classifier*.

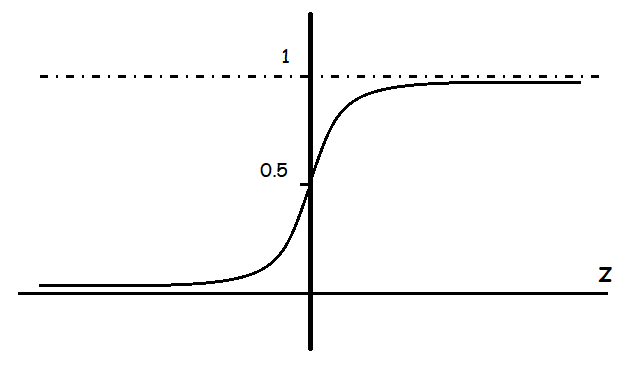
② Neural Networks



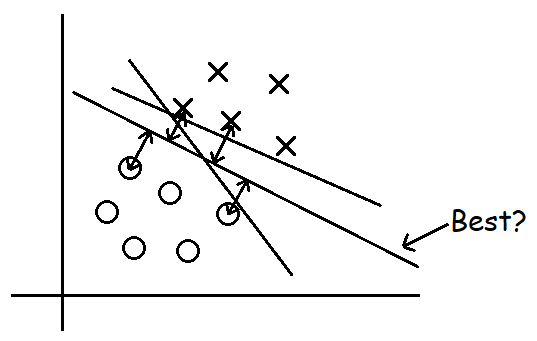
*Q: How to build a basic neural network and compute all the parameters?*

*- references: cs229-notes-deep\_learning & cs229-notes-backprop*

③ Support Vector Machines

(1) Functional margins (e.g. logistic regression)

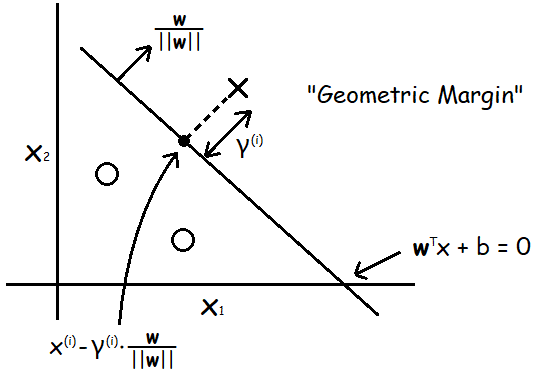
(2) Geometric margins



(3) notation

(4) definition & deduction

🡪



🡪

So to the whole samples, we have the definition of “geometric margin” as follows:

**Lesson 7**

*Outline this Lesson:*

1. Optimal Margin Classifier

2. Primal/Dual Optimization problem (KKT)

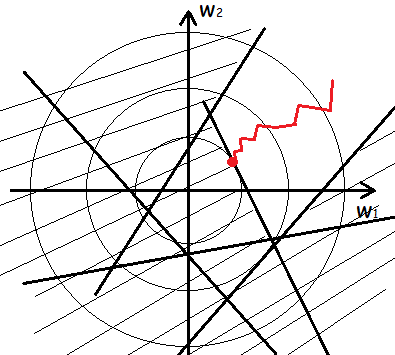
3. SVM dual

4. Kernels

① Optimal margin classifier

from last lesson, we know that change the constraints like:

will not change the original optimization problem. so,

🡪

so, original problem can re-describe into:

② Primal/Dual Optimization problem

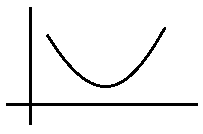
here's a function of that

we can build a *“Lagrange equation”* to re-describe it as:

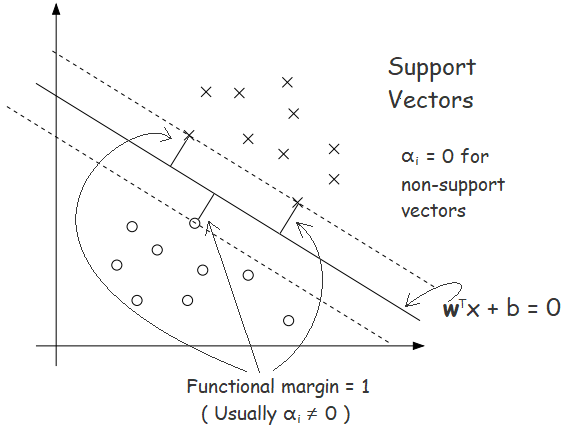
🡪

- *why?*

🡪



③ SVM dual (& KKT conditions)



SVM cost function as:

Lagrange duality

From above, we know that just associate with **.** So,

Once we solve the dual problem, we can easily solve for :

\*Finally, to predict new sample(s), we just need calculate:

④ Kernels

*\*Lagrange duality problem must be a convex optimization problem!*

proof as follows:

(<https://www.cnblogs.com/xubing-613/p/5941549.html> )

**Lesson 8**

*Outline this Lesson:*

1. Kernels

2. Soft margin

3. SMO algorithm

① Kernels

e.g.

e.g. 2

e.g. 3

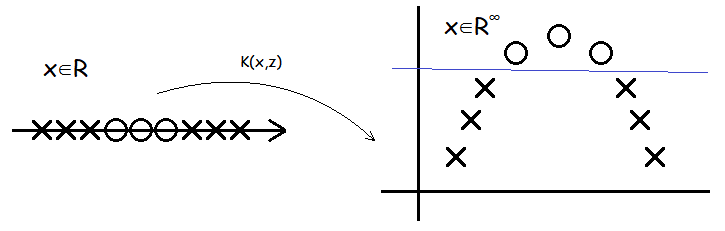
e.g. 4 (polynomial kernel)

e.g. 5 (Gaussian kernel/radial basis function)

*- ?*

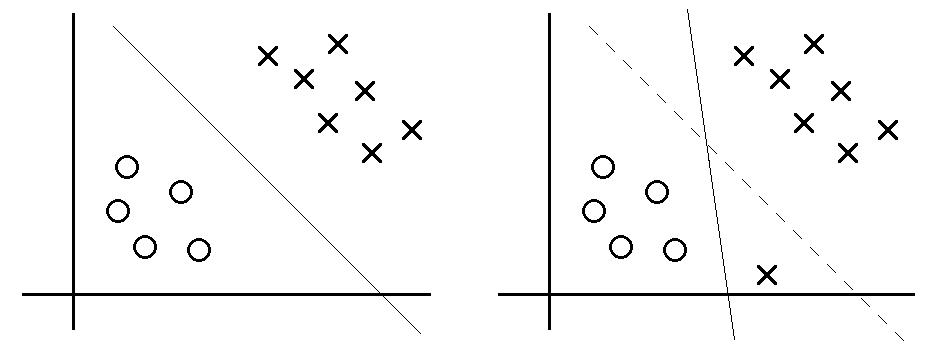
e.g.

*- How to use kernels?*



② Soft margin (Regularization)

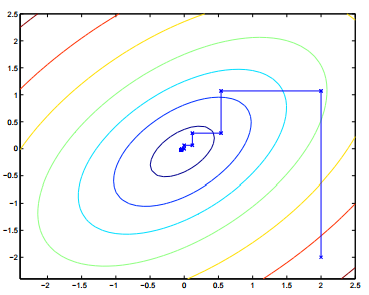
“L1 (or L2) norm soft margin SVM”



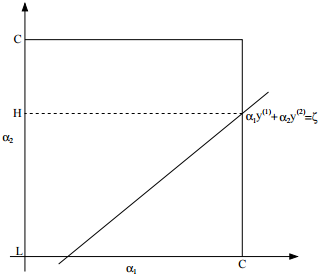
③ SMO algorithm

- Digression: Coordinate ascent

Considering an unconstrained optimization problem:



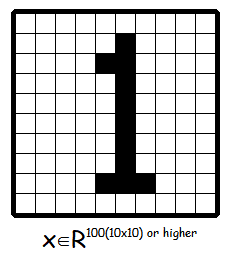
- Sequential Minimal Optimization (SMO)



reference of how to choose& update: *John Platt’s paper of SMO*

- two brief examples of using SVM

#1 Handler’s Integer Recognition



#2 classify protein sequences

*(even though only 20 to human’s protein XD)*

- *how to represent?*

**Lesson 9**

*Outline this Lesson:*

Learning Theory

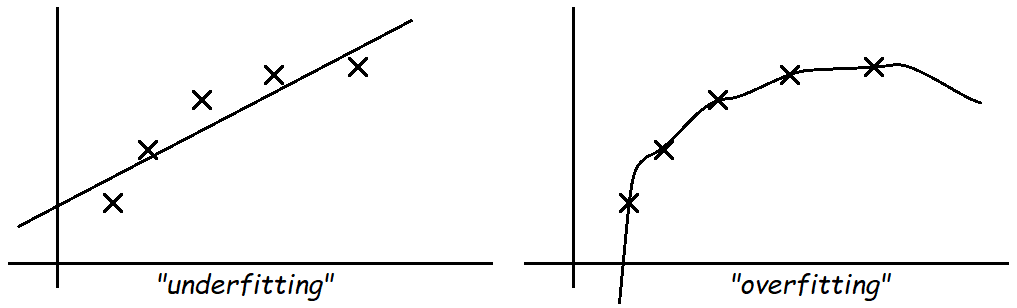
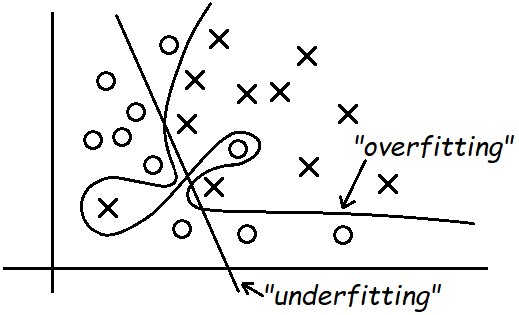
1. Bias/Variance

2. Empirical Risk Minimization (ERM)

3. Union Bound & Hoeffding inequality

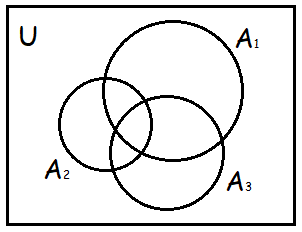
4. The case of finite (Uniform convergence)

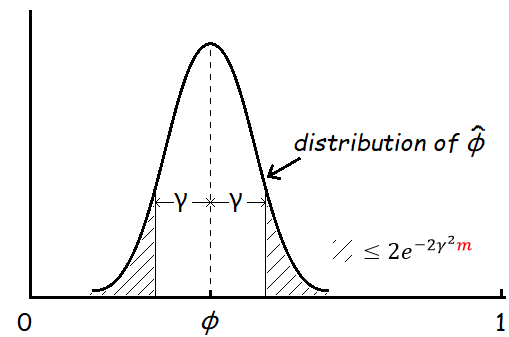
① Bias/Variance

② Empirical Risk Minimization (ERM)

All above is just curious about training with training sets, but our ultimate goal is the prediction, which means the accuracy of what we want to classify or seize. So we define

③ Union Bound & Hoeffding inequality



④ The case of finite (Uniform convergence)

So we have the *uniform convergence*:

From above, we can know that there are 3 quantities of interest: #samples , bound and the probability of error. Here’s some other ways to re-describe this solution where we can bound either one in terms of the other two!

(1)

Transform to solve m with this equation, we have the *sample complexity* bound:

In practical, almost for finite ’s length k in Computer Science, it has “”, which means we don’t worry too much about of vary kinds of hypothesis functions.

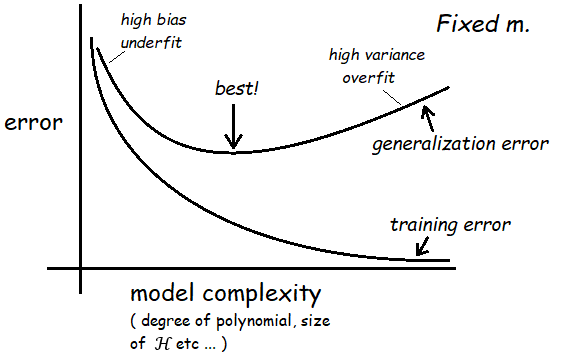
(2)

Similarly, we have the *error bound*:

From above, we have proved a so-called *bias variance tradeoff* theorem:

E.g., we have some larger hypothesis class , switching to means

(bias) can only decrease, meanwhile, the term will increase (variance) by increasing k.



PS: is the *size* of training set, is the *gap* between training error & generalization error, is the *probability* of this condition isn’t satisfied.

**Lesson 10**

*Outline this Lesson:*

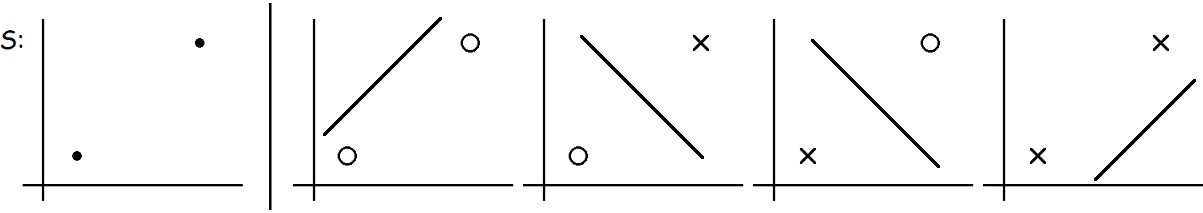
Learning Theory

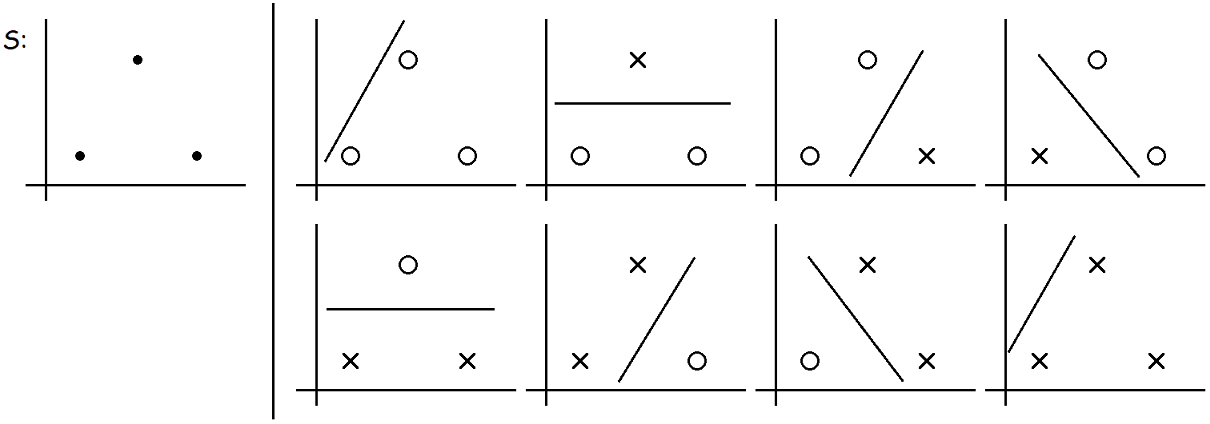
1. The case of infinite (VC dimension)🡪digression: tie up some loose facts

2. Model selection: Cross validation & Feature Selection

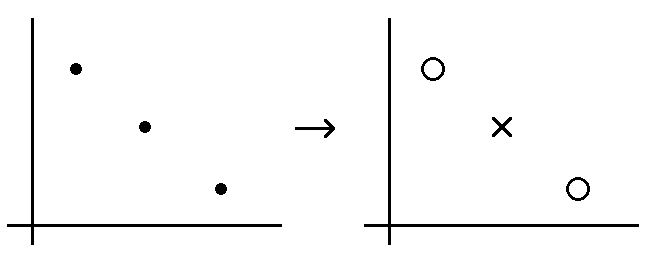
① The case of infinite (VC dimension)

In this case, the number of training set almost linear with the parameter, but not all the hypothesis class own this linear feature (generally, only suit for linear hypothesis).





*Q: this cannot shattered bysituation?*



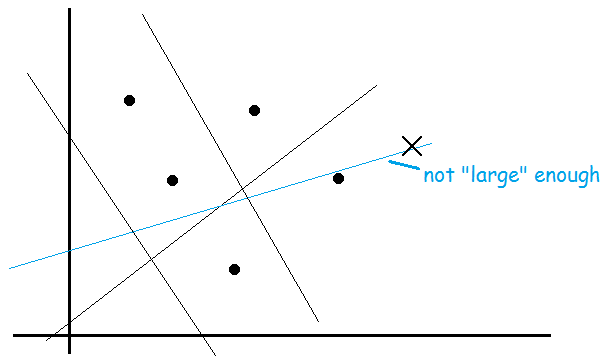
It doesn’t matter. Under the definition of the VC dimension, in order to prove that is at least , we need to show only that there’s one set of size that can shatter.

With the definition of VC dimension, we can prove a theorem of the infinite (the progress of proof quite complicated, here just conclusion).

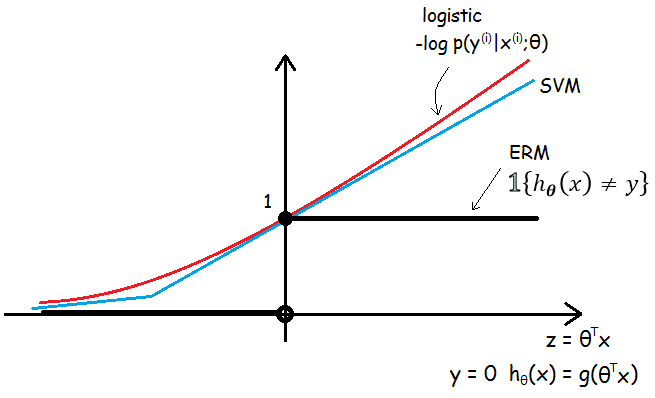
It means that, if we want to minimize the gap between training error with generalization error whether the hypothesis is finite or infinite, the least number of training set must be the *same order* with (i.e. change with the same rate, cause it’s just a very loose bound that has implicitly considered worse or even the worst condition) the number of hypothetical features (VC dimension).

- digression: tie up some loose facts

#1 Large margin linear classifier (e.g. SVM), i.e.,



#2 ERM refers to logistic regression and SVM, i.e.,

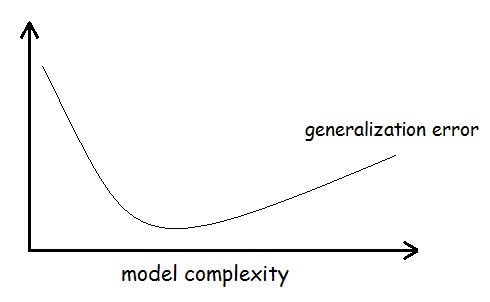


Considering only one parameter’s condition, logistic regression and SVM can be both viewed as the convex approximation of ERM(can also generalize), because the linear classifiers minimizing the training error is an NP-hard problem(not convex).

② Model selection: Cross validation & Feature Selection

(1) cross validation

Relationship between generalization error and model complexity is re-drawn as:



Consider 3 kinds of problems of selecting among several models for a learning problem:

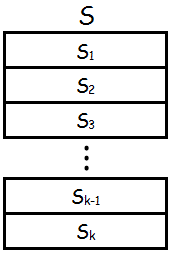
🡪 Practically, these kinds of problems we assume we have some finite set of models

Here can consist of SVM, neural network, logistic regression etcetera.

🡪 *Hold-out cross validation* (or *simple cross validation*):

- Easy and fast to use especially in quite large sets, but not sufficiently enough.

🡪 *-fold cross validation*:



- Suit for data-scarce sets (e.g.) and be more sufficient in using data, but inevitably accompany with more computational expense.

🡪 *Leave-one-out cross validation:* when, with the extremely rare data.

(2) feature selection

Some special learning problems could be described as: the number of featuresis (extremely) larger than the number of training set, but there is only a small number of features that are “(strongly) relevant” to the learning task. It means that

However we almost cannot search for the very features through comparing allmodels, here we have got some heuristic search procedure as follows.

🡪 *Forward search*:

- The algorithm can be finally terminated in 2 situations: all features are in, exceeds some pre-set threshold (e.g. pre-set).

PS: It’s a kind of *“wrapper” model feature selection*.

*Backward search*:

🡪 *”Filter” feature selection*:

E.g.

Here we can define *mutual information* (mainly in text problems):

is all the results thatcan get (usually), is the same. The probabilities can be estimated from training set.

Through information theory, we can also re-write the mutual information as a *KL(Kullback-Leibler)-divergence*:

Informally, this equation gives a measure of how different the probability distributions are. It is clearly that if and are independent random variables, the KL-divergence between will be 0.

Finally when we pick top features, we can also choose them using cross validation as measure!

**Lesson 11**

*Outline this Lesson:*

Learning Theory

1. Bayesian statistics & regularization

2. Digression: Online learning

3. Advice for applying ML algorithms

① Bayesian statistics & regularization

*“Frequentist”* view (e.g. Linear regression):

*\*“Bayesian”* view:

Practically,

*\*Regularization can always be transformed by “loss + prior” type!*

proof as follows:

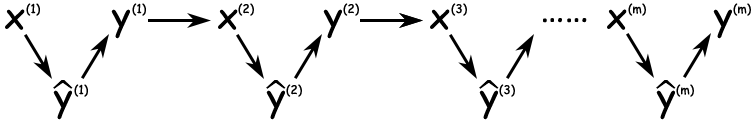
through [**Lesson 3**](#MLE), we can accordingly know that

🡪

🡪

🡪

② Digression: Online learning



Online error can also solve some incredible problem, such as when we use perceptron algorithm but, if it is proved that positive and negative examples are separated by a margin, perceptron algorithm can still converge to digital dimensional space. Total online error is at most

\*③ Advice for applying ML algorithms

Notes here are experiential for applying, might not suitable for researching. Some of these advice is debatable, but still make a dent in parts of problems.

(1) diagnostics for debugging learning algorithms

#1 Someone used Bayesian logistic regression to build up an anti-spam, which contains a small set of words as features, but got unacceptable 20% (or more) test error.

a) Common approach: Just try in different ways (quite randomly)

- Try getting more training examples.

- Try a smaller set of features.

- Try a larger set of features.

- Try changing the features: Email header vs. email body features.

- Run GD for more iterations.

- Try Newton’s method.

- Use a different value for .

- Try using an SVM.

This approach is time-consuming, gambly, sometimes it might work though.

b) Better approach: Diagnose and repair

- Run diagnostics to figure out what the problem truly is.

- Fix (or precisely optimize) whatever the problem is.

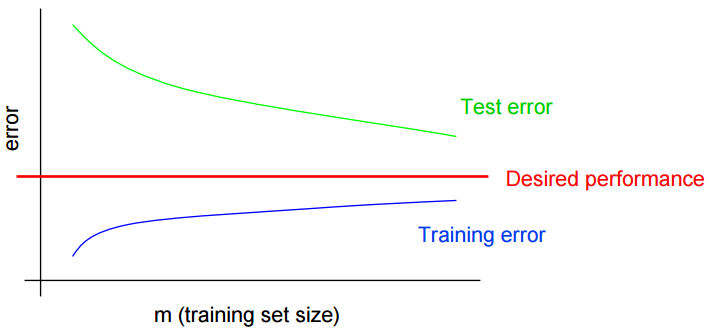
🡪 1) Suppose the problem is either (*bias vs. variance diagnostics*):

- Overfitting (high variance).

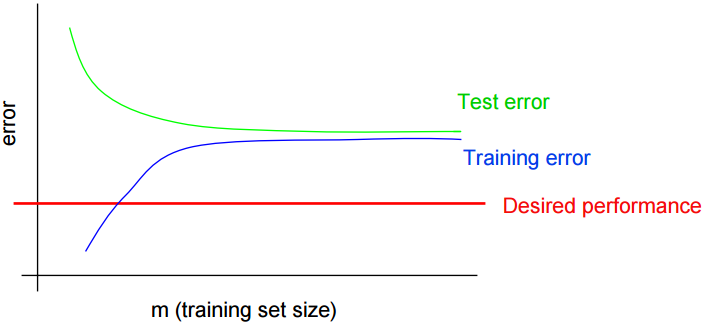
- Too few features to classify spam (high bias).

2) Diagnostic:

- Variance: Training error will be much lower than test error.



- Bias: Training error will also be high (although both are very closed of each other).



here, we can know what fixes to try:

- Try getting more training examples. Fixes high variance.

- Try a smaller set of features. Fixes high variance.

- Try a larger set of features. Fixes high bias.

- Try email header features. Fixes high bias.

For other problems, it’s usually up to ingenuity to design effective and unique diagnostics to figure out what’s wrong.

#2 Someone uses Bayesian logistic regression to build up an anti-spam, which gets 2% error on spam, and 2% error on non-spam. (Unacceptable error on non-spam.) He also applies SVM using a linear kernel, which gets 10% error on spam, and 0.01% error on non-spam (acceptable). Because of some reasons (like computational efficiency etc.), he chooses to use the former algorithm.

We can figure out lots of potential improvements (or questions) like:

- Is the algorithm converging?

- Optimize the right function?

- If using weight, need weights higher for non-spam than spam?

- Correct value for in Bayesian logistic regression?

- Correct value for in SVM? …

And whatever reason, we really want to deploy Bayesian logistic regression, even though SVM does much better for this application.

*Q: What to do next?*

BLR tries to maximize:

Diagnostics:

- Problem is with optimization algorithm.

It means that BLR tries to maximizebut fails, because using SVM’s parameterscan be larger than BLR’s. Obviously algorithm doesn’t converge well.

- Problem is with optimization objective function.

It shows that the SVM, which does worse on, actually does better on weighted accuracy. In other words, maximizingdoesn’t really correspond that well to maximizing. This confirms that if you care about,is the wrong function to be maximizing.

here, we can know what fixes to try:

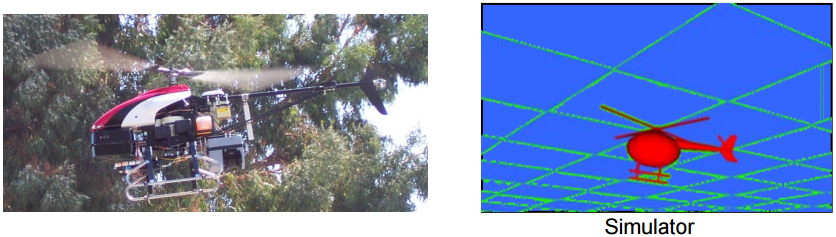
- Run GD for more iterations. Fixes optimization algorithm.

- Try Newton’s method. Fixes optimization algorithm.

- Use a different value for . Fixes optimization objective.

- Try using an SVM. Fixes optimization objective.

#3 Andrew Ng’s project on reinforcement learning: The Stanford Autonomous Helicopter.



Build a simulator of helicopter 🡪 Choose a cost function🡪 minimizeto get

Suppose the resulting controller parameters gives much worse performance than human pilot. *What to do next?*

- Improve simulator?

- Modify cost function?

- Modify RL algorithm?

Diagnostics:

- Ifflies well in simulation, but not in reality. 🡪 Problem in simulation.

Letbe the human control policy.

- If, cost function is failed to minimize. 🡪 Problem in RL algorithm.

- If, it means that minimizing doesn’t correspond to good autonomous flight. 🡪 Problem in cost function.

Conclusions:

We’ve got 3 different diagnostics of different learning problems. Practically, it’s just a little bit useful in some ways because it is quite often to come up with our own diagnostics (via ingenuity) to figure out what’s happening in different learning problems.

Moreover, even if a learning algorithm is working well, we can also run diagnostics to help us understand the algorithm further. This is useful for:

- Understanding application problems further, even getting an intuitive understand of what works and what doesn’t work.

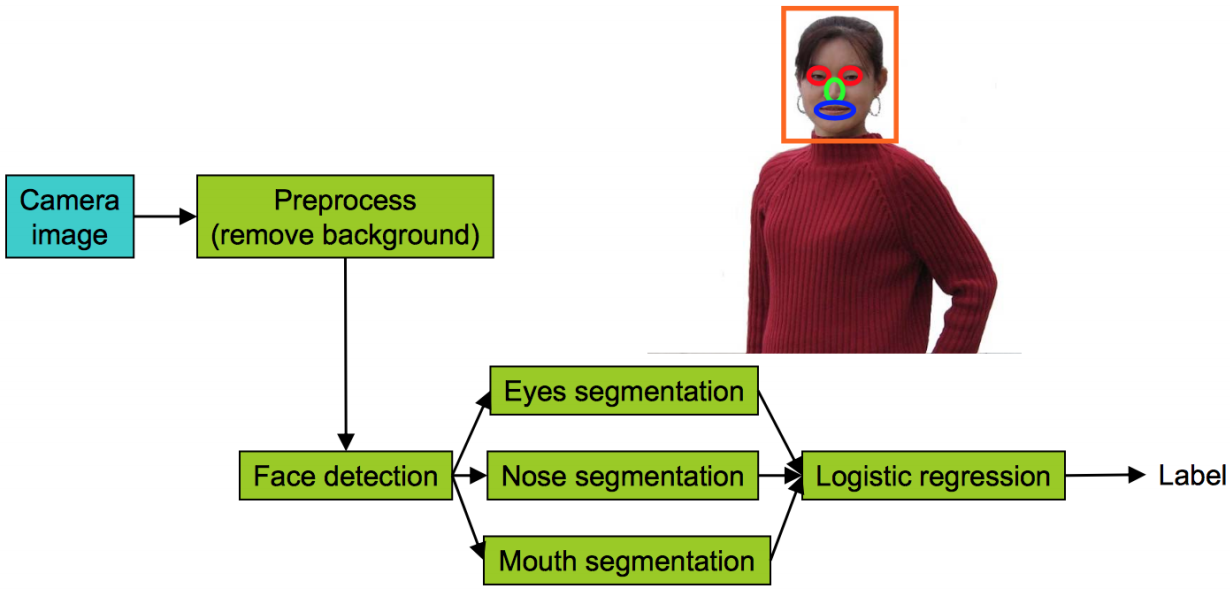
- Writing research papers: enrich insight about the problem and justify research claims.

- Making sense and being convincible when explain the core algorithm to others.

🡪 So, we need *error analysis* to understand what sources of error are!

(2) error analysis & ablative analysis

a) error analysis



E.g. A pipeline of Face recognition from images (not quite formal)

Suppose we have a pipeline like above picture. Now the recognition accuracy of overall system is only 85%.

What we can do in brief, plug in *ground-truth* for each component, which means the perfect output of each component however it’s got (like using PS, coding by hand etc.), see how accuracy changes. Perhaps the changes can be noted as:



And then, find the several maximum gaps between and improve them first. In this case, we know that we have most room for improvement in face detection and eyes segmentation.

b) ablative analysis

Compared with error analysis, ablative analysis is the opposite strategy which tries to explain the difference between some baseline and current performance.

E.g. A good anti-spam classifier by adding lots of clever features to logistic regression:

- Spelling correction.

- Sender host features.

- Email header features.

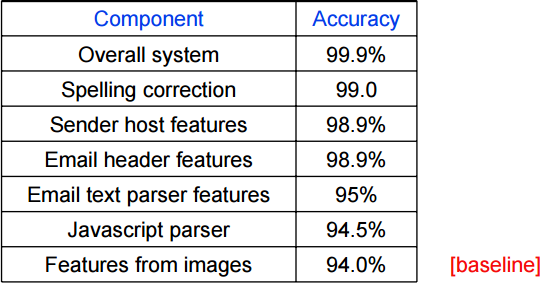
- Email text parser features.

- Javascript parser.

- Features from images.

Q: How much did each of these components really help?

Suppose we apply a simple logistic regression without any clever features get 94% performance. In ablative analysis, just remove components from the system **one at a time** (not only one-by-one, one-out-others-in if suitable is ok) to see how it breaks. Perhaps the changes can be noted as:



Accordingly, we can also find out the several maximum gaps between and figure out what is the core component of the system. In this case, it shows that email text parser features contribute for the most of the improvement.

From the discussion above, we can reorganize the both analysis briefly as:

|  |  |  |
| --- | --- | --- |
| Type | **Error Analysis** | **Ablative Analysis** |
| How | 1. Select suitable components  2. Make each “perfect” (*ground*  *-truth*), record the changes  3. Find out the most improving parts, focus on those | 1. Select suitable components  2.Remove each in some rules, record the changes  3. Find out the most decreasing parts, pay attention on those |
| Situation | Improve the algorithm or debug it. | Analysis improved algorithm, find out the core change(s). |

(3) how to get started on a learning problem

There are two typical approaches to applying learning algorithms:

#1 Design very carefully, then implement it.

Benefit: Nicer, perhaps more scalable algorithms. May come up with new, elegant, learning algorithms; contribute to basic research in machine learning.

Risk: *Premature* (*statistical*) *optimization*.

#2 Build a quick-and-dirty prototype, diagnose and fix it.

Benefit: Will often get application problem working more quickly. Faster time to market.

Risk: Time-consuming debugging and testing (if not quite experienced in some part).

**Lesson 12**

*Outline this Lesson:*

Unsupervised Learning

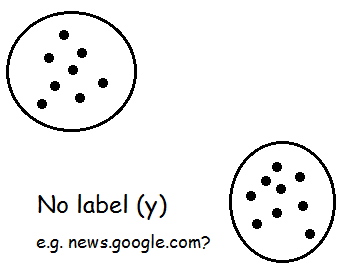
1. Clustering (k-means)

2. Mixture of Gaussians

3. Jensen’s inequality

4. EM (Expectation-Maximization) algorithm

① Clustering (k-means)



*K-means algorithm*:

This algorithm guarantees convergence by *distortion function*:

Here is a label of to decide one examplebelongs to which cluster centroid related. Moreover, (# clusters) is assumed already known, even though it is quite a problem to choose practically. If we are just curious about the minimum of , we can choose this parameter “randomly” and choose that can minimize . (I.e. the exactmay vague as below.)

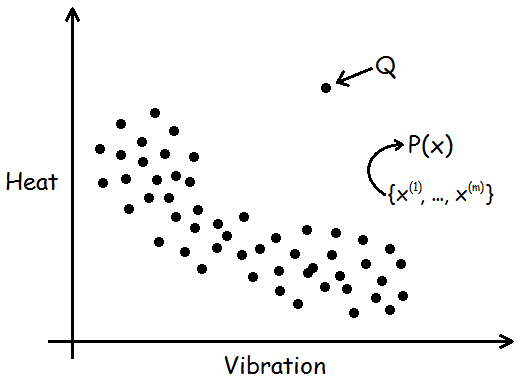


Algorithm above do those things in two steps: (i) “Assigning” each training exampleto the closest cluster centroid . (ii) Moving each to the mean of the points assigned to it.

As a matter of fact, *k*-means is exactly coordinate descent on . Because is a non-convex function, *k*-means can always converge, but not guarantee to converge to the global minimum. (Initialize cluster centroids randomly and choose the minimum may help.)

🡪 Density estimation

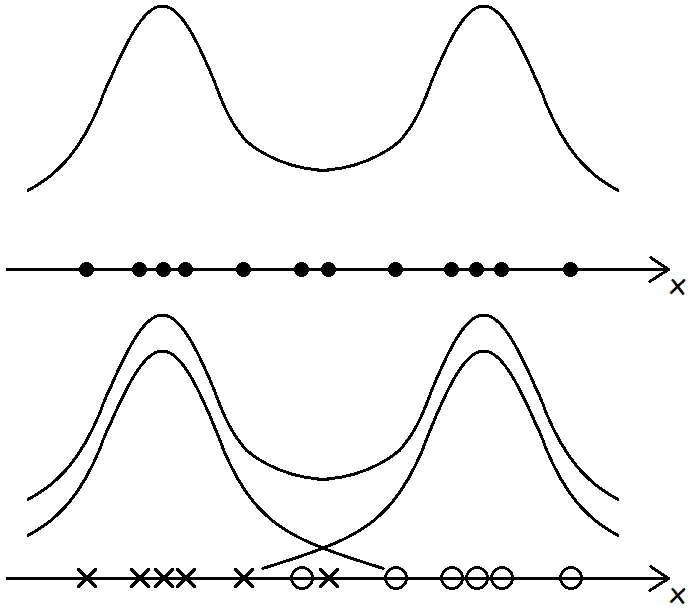
Here’s a practical example, we need to detect whether an aeroengine is broken or not (briefly) in such two potential factors: vibration and heat. The data all we have and the detected one, point *Q*, are shown as:



We can divide all training set into several clusters by *k*-means, then check *Q* belongs to none of them to figure out *Q* is the one need further inspected. Some problem like this are called *anormaly detection*. For brevity, we can describe the algorithm to solve those as:

In this part, we are curious about *step 1*: how to get, so called *density estimation*. Considering this situation, we can hardly build a model with all the basic distributions we have already known like Gaussian or Poisson etc. So a usual way to build complex models like this is to apply the mixture of Gaussians.

② Mixture of Gaussians



As it’s shown above, it is quite similar with GDA in [**Lesson 5**](#GDA)(here each Gaussian may vary because of the difference of). However, the fact is that we actually do *not* know. In order to satisfy this condition, we need special type of EM algorithm to help.

🡪 *EM* *algorithm (with mixture of Gaussians model)*:

We can apply Bayes’ Rule to expand *E-step* as:

Despite of computational difficulty, we can update easily. Practically, *k*-means can also be described as EM algorithm. Moreover, EM algorithm converges to the local minimum instead of the global minimum.

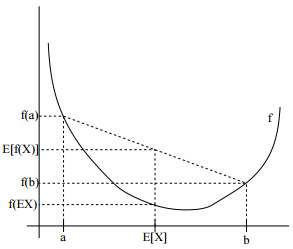
*Q: Does EM algorithm can always converge like k-means?*

- Need some other factors first, like Jensen’s inequality.

③ Jensen’s inequality

*Jensen’s inequality* actually is the property of convex functions or distributions, and it has lots of different forms in different ways. Here’s one useful form of them:

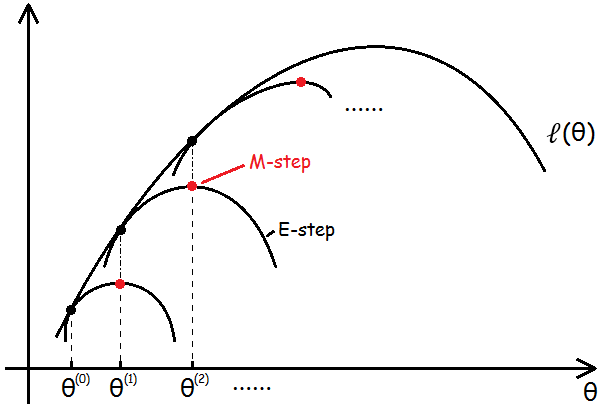
We can draw a picture to help us understand and remember it:



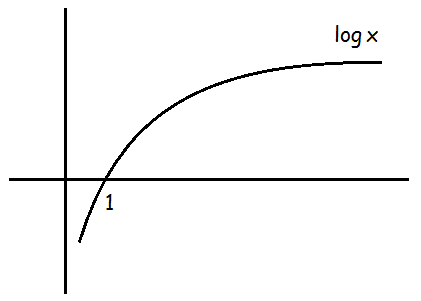
④ EM (Expectation-Maximization) algorithm

From the mixture of Gaussians part, we can conclude it and generalize such problem as follows:

Now the problem is transferred to find suitableto maximize the likelihood. The general EM algorithm gives an efficient method instead of maximizingstraightly: (i) repeatedly construct a lower-bound on(*E-step*); (ii) optimize that lower-bound (*M-step*). The overall process can also be drawn as a picture:



For each , let be some distribution over the , we get:

Here we use some little tricks:

According to the above equations, we know that for ***any***satisfied those given conditions, we have a lower-bound on.

*Q: How to choose the very?*

- Want fit the inequation’s equality sign, in other words, make the lower-bound **tight**!

Since we know ,

Thus, we simply set the to be the posterior distribution of the given and parameterized by .

Finally, we can give out the *generalized EM (Expectation-Maximization) algorithm*:

**Lesson 13**

*Outline this Lesson:*

EM

1. Mixture of Gaussians

2. Mixture of naive Bayes

3. Digression: Gaussians

4. Factor analysis

① EM: Mixture of Gaussians

- Another perspective of EM (in optimization)

- Mixture of Gaussians

According to the process above in [**Lesson 12**](#EM_MoG), we know that in *E-step*:

So, *M-step*:

#1 maximize with respect to

🡪 Compute the update rule of :

#2 maximize with respect to

Considering the constraint of , we can construct the Lagrangian (is naturally satisfied because of log function):

Similar to the equation in [**Lesson 12**](#EM_ratio), we have so it’s easily to figure out the Lagrange multiplier

🡪 Compute the update rule of :

#3 maximize with respect to

🡪 Figure out the update rule of :

② EM: Mixture of naive Bayes

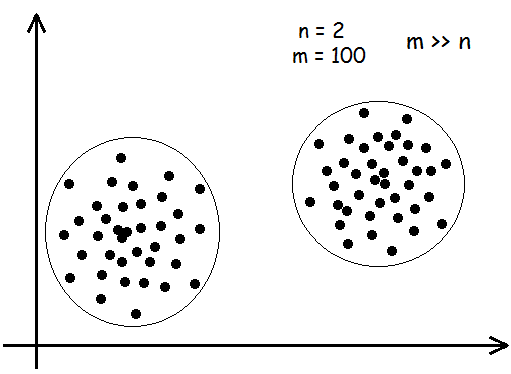
Here’s a Text Clustering example (Multi-variant Bayes event model)

So, EM algorithm of this model

The parameter here are almost similar to 1 or 0 (e.g. 0.999/0.00001). Obviously, in *M-step* we treat as GDA’s and keep updating all the parameters till convergence.

③ Digression: Gaussians

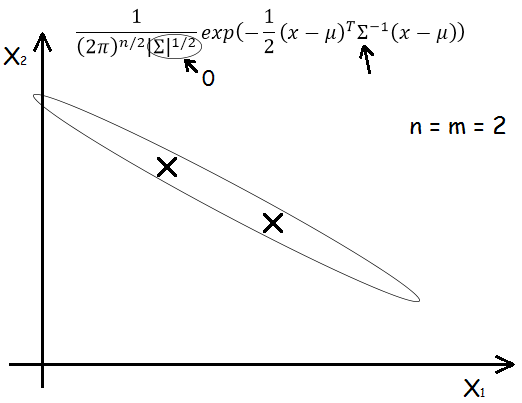
Most models we have discussed above in unsupervised learning satisfy. For instance, here’s a simplified GMM data:



And we have

Whenor even, we’ll find that the matrixis singular. Because of this problem,anddo not exist, which means the model itself cannot describe this situation to solve the core problem we really concern in this way.

Here’s a quite impressive example to describe such a problem, the numbers of training set and features are both 2, the potential Gaussian model would be a very narrow ellipse (infinite width and very thin height).



*Q: How to deal with it?*

- Use some restrictions of.

The right side restriction is stricter than left side if necessary. Even both method can rebui-ld model to fit, however, they will model the orthogonal features of data which means each fe-ature is uncorrelated and independent. Often, we are curious about the correlation structure in the data. We need to build a *factor analysis model*, which use parameters than the diagonal and captures some correlations in the data, without having to fit a full covariance matrix.

- Marginals and conditionals of Gaussians

Since marginal distributions of Gaussians are themselves Gaussian and above shows that

, we can also figure out the conditional distribution ofgivenas , where

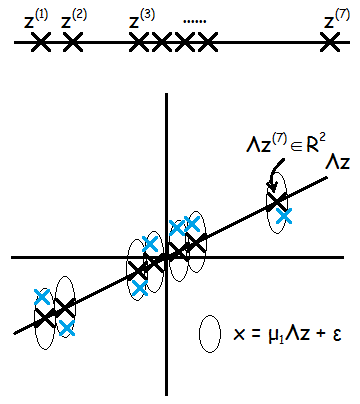
It’ll be helpful in the next part. (the details of conditional distribution parameters’ proof at *references: more\_on\_gaussians Pg. 8*. PS: pay very attention to get, partitioned matrix!)

④ Factor analysis

Posit a joint distribution onas follows, hereis a latent random variable:

Equivalently, we can also define that

E.g.



So,andhave a joint Gaussian distribution

🡪

🡪

Moreover, we can also figure out the marginal distribution ofis given by

So, we can write down the log likelihood through a given training set

Accordingly, *EM algorithm of factor analysis model* is described as follows:

\*\*Herein *M-step* of one loop is a *fixed* part just after *E-step* finished.

**Lesson 14**

*Outline this Lesson:*

1. Factor analysis: EM steps

2. Principal Component Analysis (PCA)

① Factor analysis: EM steps

From [**Lesson 13**](#EM_FA), we can figure out that *EM algorithm of factor analysis model* can be described as follows:

Here, we want to get the exact *M-step*, the update form of exact parameters instead ofstep by step**.** First, we write down the complete form of “max” goal.

#1 get:

Hereis a fixed parameter when update.

#2 get:

Meanwhile, we have

#3 get:

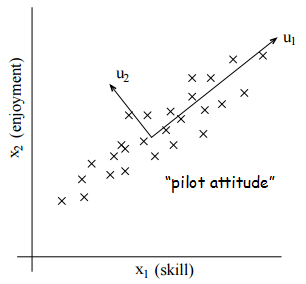
Similarly, we have (here should use some [***tricks***](#trick_grad) to getand):

🡪 Final *EM algorithm of factor analysis model*:

② Principal Component Analysis (PCA)

PCA is another dimensionality reduction algorithm just with a little bit “lack” of original in-formation. This algorithm directly do eigenvector calculations instead of EM. The main progre-ss can be described as follows:

E.g. A case of helicopter pilots’ skill level associated with their enjoyment. As the picture below, maybe either skill or enjoyment is our concern. As a matter of fact, the inner relation of both features, named “pilot attitude” (vector), is the very feature we are truly interested in. And the other quadrature componentjust means the noise or other nonsense part.

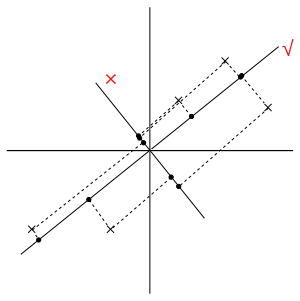


Before we develop the PCA algorithm, we need to do pre-processing to normalize its mean and variance first.

In some case, e.g. a grayscale image, which has the same scale of each feature don’t need to do *steps 3-4*.

*Q: How to compute the “major axis of variation”?*

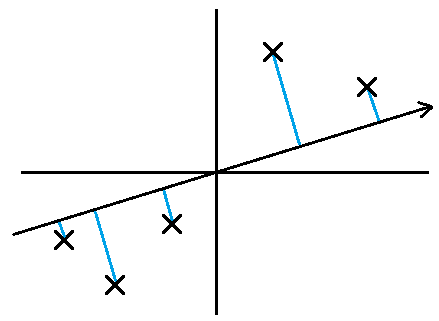
- Finding a unit vectorso that when data is projected onto the direction corresponding to, the variance of the projected points is maximized.



We can describe it in a specific way as follows:

Sometimes, the several eigenvalues may be very closed to themselves. And it’s quite dangerous to choose only one or two top eigenvectors to represent it instead of subspace. Because the several “similar” eigenvalues may lead to those associated eigenvectorsrotate freely within subspaces. Choosing topeigenvectors (usually containsoriginal info) is usually about same.

Here’s another way to explain PCA, trying to minimize the sum of the distance’ squares between each origin data point and accordingly projected point (sum of squares of blue parts).



Some applications by PCA:

- Visualization.

To draw pictures of high dimensional data, use PCA to reduce it to 2-d or 3-d.

- Compressor.

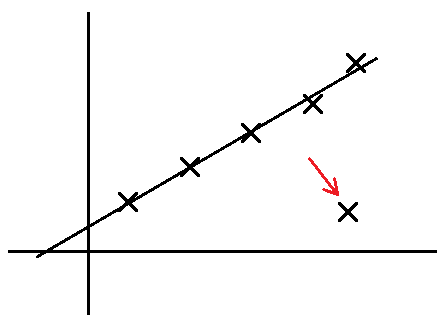
Use much lower dimensions to save original main information, without too much loss.

- Learning problems.

1. avoid overfitting, e.g. linear regression; 2. reduce dimensions to simplify the model.

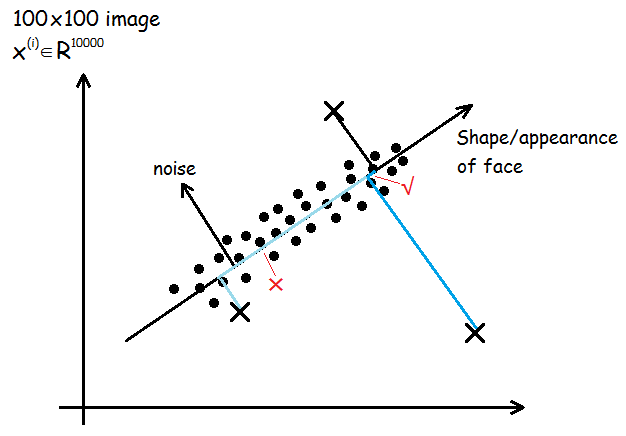
- Anomaly detection.

When a new prediction is quite far away from the subspace in some case.



- Matching/distance calculations.

E.g. face detection. When positions of faces vary in different pictures or the origin features almost have 10,000-d.



Original faces Eigenfaces

**Lesson 15**

*Outline this Lesson:*

PCA

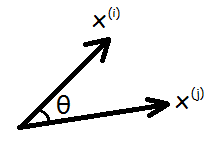
1. Latent Semantic Indexing (LSI)

2. Singular Value Decomposition (SVD) implementation

ICA

3. Independent Component Analysis (ICA)

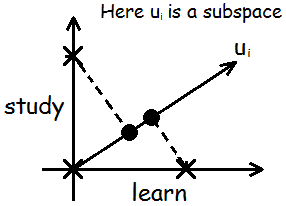
① Latent Semantic Indexing (LSI)

Here’s an example. We want to compare lots of text article to find the most similar pairs of them or just scatter them into different clusters by their themes. The first thing we can do is represent all the data set as the vector of a solid vocabulary. Such as:

🡪

Observe the equation above, the numerator ofactually is

And here comes a problem. When two documents have different words with the same meaning, thehere must become 0. To avoid this situation, we can apply PCA to project such features into, and then calculate. That’s LSI really matters in brief. (E.g. the words “learn” and “study”)



② Singular Value Decomposition (SVD) implementation

When the number of training set features is very large (e.g.), the covariance of it () may become a very high dimensional matrix, not easily figured out the principle components. One way to solve such a problem is applying SVD to decompose this very large matrix into three matrix product.

In PCA, we have two ways to implement SVD.

#1 Implement SVD directly to training set (when).

#2 Implement SVD to(when).

The last part is some advice and comparisons, when training set satisfies as:

🡪

Here’s a conclusion table of when to use such unsupervised algorithms so far:

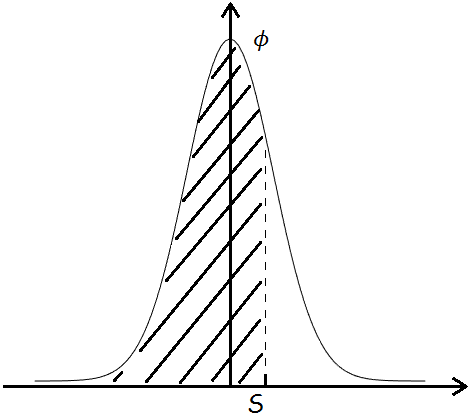
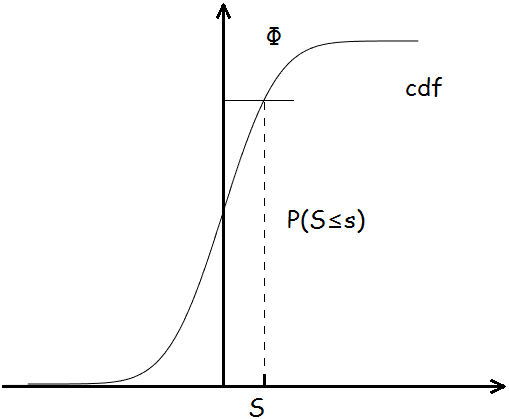
|  |  |  |
| --- | --- | --- |
|  | **Model** | **Not probabilistic** |
| ***“Subspace”*** | Factor Analysis | PCA |
| ***“Clumps”/***  ***“Groups”*** | Mixture of Gaussians | K-means |

③ Independent Component Analysis (ICA)

*Cumulative distribution functions (Cdf)*:

It means when we specifyor, we can get one with another because

E.g. Gaussian

Considering the “cocktail party problem”, here we havemicrophones placed in the room to record severalspeakers different combinitions of voices. The goal is separating out the original speakers’ speech signals. We can observe:

Here’s some so-called “ICA ambiguities”. E.g. assume. Because of the symmetry and, the scaling (voices volume), permutation (speakers sequence) and the sign of speakers’ voices may vary but finally don’t matter. (If Gaussian, we cannot decompose even one of them because the density is rotationally symmetric.)

*ICA model*:

*Q: How to choose ?*

- E.g. . We can also use Laplacian or others.

We can implement, and maximize it via stochastic gradient ascent (or can be transferred to SGD):

Finally, we can get the parameter matrix(though it’s not very accurate practically, because the truth iss are dependent) to rebuild the origin unmixed.

PS: Some applications of ICA:

- EEG (Electroencephalogram) data analysis preprocess.

- Small natural image patches (Humanlike recognition).

**Lesson 16**

*Outline this Lesson:*

Reinforcement Learning

1. MDPs (Markov Decision Processes)

2. Value Function

3. Value iteration

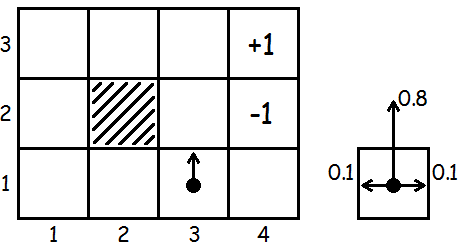
4. Policy iteration

① MDPs (Markov Decision Processes)

(not formal):

Whether we get a positive or negative reward, figure out what we *actually* did right or did wrong to cause the reward, so we can do more of the right things and less of the wrong things.

Here is an example to describe how to model with simplified MDP, not considering.

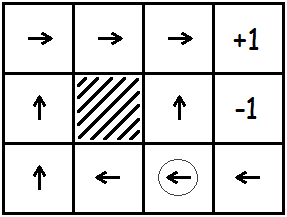


We can know that

Suppose that when we let the android go north, the probability of going north is just 0.8 along with 0.1 going east or west (assume not stand still), and when it doesn’t move to the final sections, it will take some consumption by -0.02. Then we can write down every element in this step tuple just like

*Q: How to understand the meaning of “policy”?*

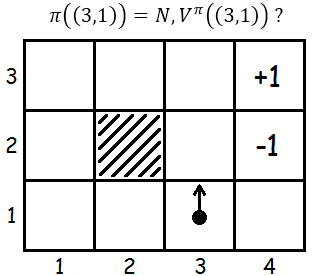
- E.g.



Policy here means a function mapping from the states to the actions. Concretely, we execute some policyif, whenever we are in state, we take action.

② Value Function

e.g.



So, we can also have the “*optimal policy”* :

Obviously, we can get the relationships among those 3 functions that

There is one meaningful property that we can use the same policyno matter what the initial state of our MDP is!

③ Value iteration

*Value iteration algorithm*:

The inner-loop of this algorithm can be updated both in synchronous and asynchronous ways. No matter what the inner loop is actually realized, value iteration will cause to converge to , and we can finally find the optimal policy .

④ Policy iteration

Instead of calculating the optimal value, we can also straightly find an optimal policy for an MDP.

*Policy iteration algorithm*:

Both value iteration and policy iteration are standard algorithms for solving MDPs, and there isn’t currently universal agreement over which algorithm is better. However, for MDPs with large state spaces, value iteration maybe preferred. For this reason practically, value iteration seems to be used more often than policy iteration.

*Q: What if we don’t know ?*

- We can estimate them from data via:

Reinforcement learning problems may have different policies to reach to the final goal. E.g. using value iteration with unknown :

**Lesson 17**

*Outline this Lesson:*

Continuous state MDPs

1. Discretization

2. Models/Simulators

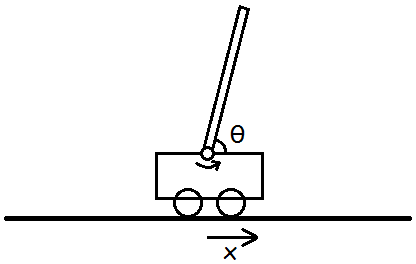
3. Fitted value iteration

① Continuous state MDPs: Discretization

In [**Lesson 16**](#finite_MDPs), we have focused on the MDPs with a finite number of states. We now need to discuss some algorithms for solving MDPs with an infinite number of states.

E.g.

PS: ”*Inverted pendulum*” problem can be shown as picture below, what we’re care about is how to keep dynamic balance of the pole with the little car’s moving controls.

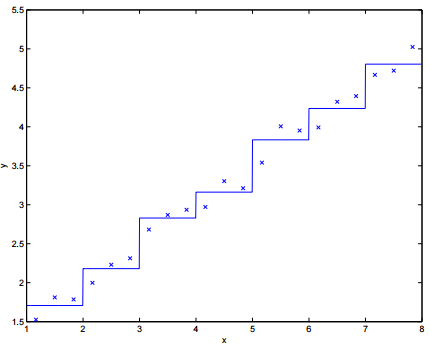


One easy way to solve this kind of problems is to discretize those state space. E.g. we have 2-d states, we can use a grid to discretize the state space like:



Here’s two downsides when using this discretization algorithm:and are assumed fairly naive which means the value function (policy function) is piecewise constant in each of the gridcells.

To better understand such a limitation, we can take a supervised learning problem as an example. Obviously, linear regression would do fine on this problem. However, if we use a representation of discretization intervals, then it’ll look like this:

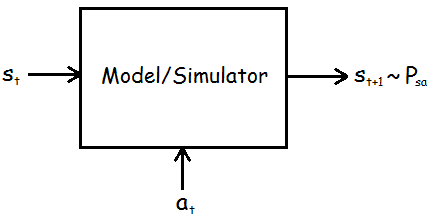


When it happens, we would also need a very fine discretization, which means discretize into very small grid cells, to get a good approximation. Meanwhile, if we do discretize quite much smaller, we can cause another problem called the *curse of dimensionality*.

Owing to this exponential growth, discretization does not scale well to large problems. E.g., with a continuous state, if we discretize each state variable intovalues, we would havediscrete states, pretty high computational expense!

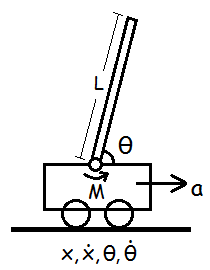
As a rule of thumb, discretization usually works extremely well for& problems, along with being simple and quick to implement.& problems may also works with some extremely careful designs. When, it will very rarely works to solve the problem.

② Models/Simulators



We have several ways to get such a model. One is to use *physics simulation*. Here’s an example of the inverted pendulum mentioned above.

E.g. *physics simulator*



We can easily get , then the other parameters can come out naturally.

An alternative way to get a model is to *learn a model* from data straightly. Suppose we execute**trials** in which we repeatedly take actions in an MDP, each trial fortimesteps.

Thestate sequences like the following:

We can use learning algorithm to estimateas a **function** of.

E.g.

Here we can use both *deterministic* model and *stochastic* model as follows:

*Q: Non-linear model?*

-

③ Fitted value iteration

Recall the discrete value iteration, we would like to perform the update here as:

In this algorithm, we just assume satisfies stochastic model. When using deterministic model, we can simplify it by setting the parameter to be equal to 1.

*Q: How to get the action ?*

- Because of the continuous states and the discrete actions, we just take action at some specific states by

may hardly come out in some case when we have to sample state. We can apply the approximation when the simulator satisfies some deterministic type, i.e. , is a deterministic function such as and or using the approximation

then choose action

**Lesson 18**

*Outline this Lesson:*

1. State-action rewards

2. Finite horizon MDPs

3. Linear dynamical systems: Models

4. Linear Quadratic Regulation (LQR)

5. Riccati equation

① State-action rewards

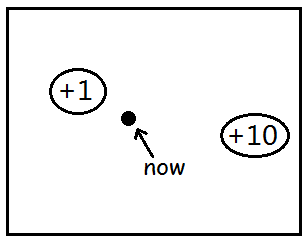
General (discrete) MDPs:

Here the expectation part means estimating the next state by the distribution. And we have a more general setting to fit continuous MDPs precisely, we need to apply some variation of previous assumptions.

② Finite horizon MDPs

*Q: Why here the optimal policy happen to be non-stationary?*

- E.g.

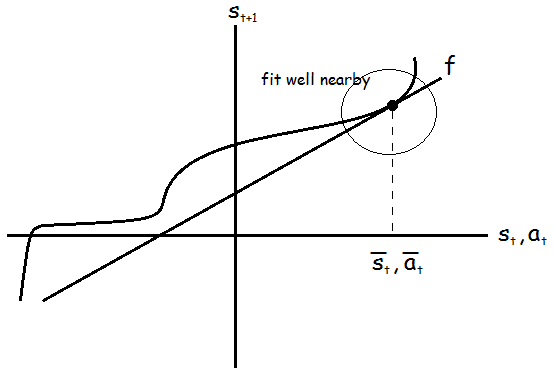


As the cartoon shown above, we want to take actions to aim for the +10 goal at the very beginning. However, when we don’t have enough time steps left to reach the +10 goal along with a closer +1 goal, choose the latter is better.

③ Linear dynamical systems: Models

We can calculate every value and policy sequence with *dynamic programming* like:

E.g. inverted pendulum



*Q: How to solve the problem ifhas a constant?*

- *Let*.

④ Linear Quadratic Regulation (LQR)

E.g. Helicopter control

To simplify this model, we can also suppose that

⑤ Riccati equation

Using, we can finally get the *Discrete Ricatti equations*:

If we just need to figure out the optimal policy, we can run this algorithm just to update only!

**Lesson 19**

*Outline this Lesson:*

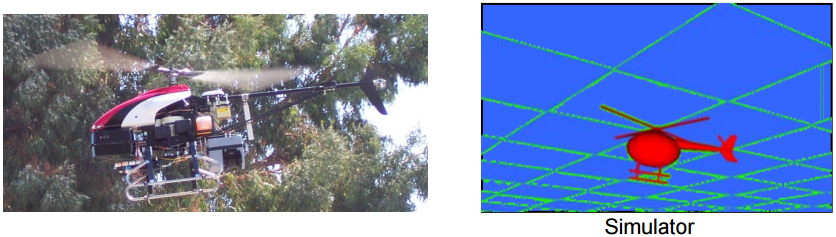
1. Debugging RL algorithms

2. LQR - Differential dynamic programming (DDP)

3. Kalman filter & Linear Quadratic Gaussian (LQG)

① Debugging RL algorithms

This example we have talked about in [**Lesson 11**](#RL_debugging), here we pay attention to the analysis of such a problem rather than diagnostics before.



Build a simulator of helicopter 🡪 Choose a cost function🡪 minimizeto get

Suppose the resulting controller parameters gives much worse performance than human pilot. *What to do next?*

- Improve simulator?

- Modify cost function?

- Modify RL algorithm?

\*

Diagnostics:

- Ifflies well in simulation, but not in reality. 🡪 Problem in simulation.

Letbe the human control policy.

- If, cost function is failed to minimize. 🡪 Problem in RL algorithm.

- If, it means that minimizing doesn’t correspond to good autonomous flight. 🡪 Problem in cost function.

② LQR - Differential dynamic programming (DDP)

The generate steps can be described as:



Here we just give out the steps of the algorithm, and we will discuss the exact example in the next lesson.

③ Kalman filter & Linear Quadratic Gaussian (LQG)

In the real world, we can hardly observe the full state, we just observe the partial state. To solve such a problem, we have a new model called *POMDP* (Partially Observable MDPs). In this case, we cannot straightly get current state, but with the observation, we can indirectly get it by:

Meanwhile, the tuple of a finite-horizon POMDP is

In this section, we can get the LQG, which is a special type of LQR through the following assumptions (is the observation of the *belief state*):

Using the marginal formulas of Gaussians, we would get (theoretically)

Unfortunately, the matrices of the marginal distribution always change in time! When time goes, the final matrices will be computationally expensive, about a cost in! To avoid this situation, we use the *Kalman filter* algorithm.

So, is the “best” estimate for.

Through this algorithm, we can obviously figure out that we don’t need to calculate the formertime steps! The update steps only depends on the previous distribution.

To put it all together, algorithm first runs a *forward pass* to compute the . Then it runs a *backward pass* (LQR updates) to compute . Finally, we can get the optimal policy .

**Lesson 20**

*Outline this Lesson:*

1. POMDPs (Partially Observable MDPs)

2. Policy search

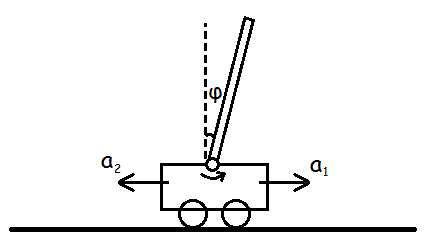
3. Conclusion

① POMDPs (Partially Observable MDPs)

② Policy search

(1) Reinforced

E.g. An inverted pendulum system just like



So the *reinforced algorithm* (gradient ascend):

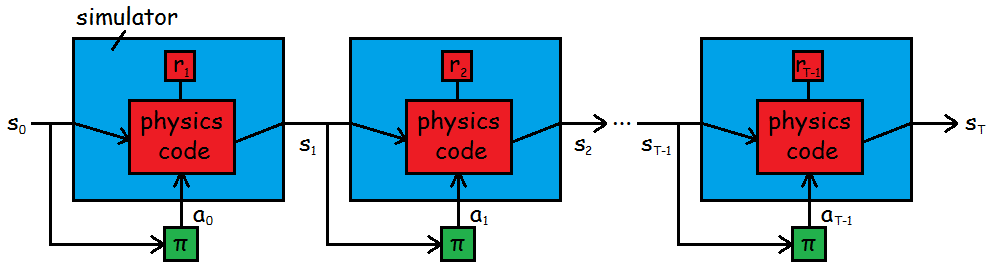
If we only have an approximationof(could befrom Kalman Filter), we can also use it with just a little variation:

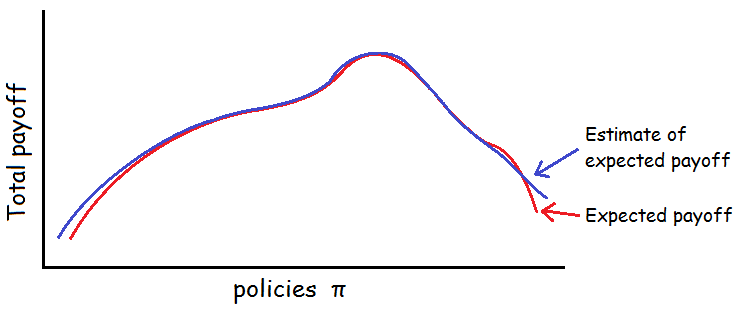
As a gradient ascend algorithm, the reinforced also has the negative property such as: Because of the observed noise, the sampled sequence in essentially a sort of random direction, only satisfy the expectation; As the parametergrows and the noise, it may take a lot of time to sample the different sequence (see the pseudocode), converge with million iteration step etc.

This algorithm is more useful in just simulator rather than the actual physical device. So, we have another algorithm called *PEGASUS* to simplify it.

(2) PEGASUS (Policy Evaluation of Gradient And Search Using Scenarios)

This algorithm totally based on the inspiration of simulator way to build models (rather than sample the physical data). The core method is using a random number generator ***only once*** to generate the stochastic sequence. Then fix it in advance and always use the same sequence of random numbers.





③ Conclusion

*Policy search* algorithms are easier to the low level control tasks (just like reflexes), such as helicopter flight, driving or controlling various robots.

Problems of long path plannings (like chess, Tetris etc.) prefer to apply a *value function* method instead.

The key of RL problems is really *sequential decision making*, which means we need to make decisions step by step, and each step may have long-term consequences.