Machine Learning

2017

Book:

* Machine Learning: A Bayesian and Optimization Perspective, by Sergios Theodoridis

# Matrix Calculus Notation

Remember that the vectors are column vectors, even though I sometimes write them as row vectors for convenience. So goes on the first row, goes on the second row, and so on.

# Measures of Performance

## Learning Curve

This is a graph showing how fast the model is learning from the data.

Underfitting situation:

error

validation error

training error

training set size

The training error and the validation error converges quickly, and it's relatively high (compared with the overfitting case).

Overfitting situation:

error

validation error

training error

training set size

The training error is initially very low, and the validation error is initially much higher than the training error.

With larger and larger training sizes, the overfitting problem is gradually solved, and the two errors converge, to a level that is lower than the underfitting situation.

## Generalization Errors

**Bias** - error due to wrong assumptions, such as using a linear model when the data is actually quadratic. The model is underfitting the data.

**Variance** - error due to excessive sensitivity to small variations in the data. The model has many degrees of freedom and is overfitting the data.

**Irreducible error** - error due to noise in the data

# #2. Probability and Stochastic Processes

## Probability and Random Variables

**Conditional Probability**

|  |  |
| --- | --- |
|  | (2.10) Conditional Probability |
|  | (2.12) Product Rule |
|  | (2.14) Bayes Theorem |

**Mean and Variance**

|  |  |
| --- | --- |
|  | (2.24) Mean Value |
|  | (2.25) Variance |
|  | (2.28) Covariance of two random variables |
|  | (2.31) Covariance matrix of multiple variables |

Covariance matrices are symmetric and positive semidefinite.

**Transformation of Random Variables**

Let two random vectors and be related via an invertible transform .

|  |  |
| --- | --- |
|  | (2.44) The Jacobian matrix is a matrix of all possible first order derivatives. |
|  | (2.45) Relationship between the pdf of and . |

## Examples of Distributions

**Discrete PDFs**

|  |  |
| --- | --- |
|  | (2.50) Bernoulli distribution |
|  | (2.53) Binomial distribution |
|  | (2.58) Multinomial distribution |

**Multivariate Gaussian**

|  |  |
| --- | --- |
|  | (2.69) Multivariate Gaussian pdf |

**(hyper)ellipsoid set of points**

For the Multivariate Gaussian PDF, an ellipsoid set of points forms the level set --- they generate the same value.

Reasoning:

All points that score the same density value obey

The covariance matrix Σ is symmetric, which guarantees the orthogonally diagonalizable factorization

where are eigenvalues  
and are orthonormal eigenvectors

Use to simplify

into

use property of orthogonal matrix

where

To go from to require a translation to and then a rotation using UT.

For the three variable case:

So inconclusion, the points on an ellipsoid, generate a constant .

For 2D Gaussian PDF, the level contours are ellipses.

**Uncorrelated variables mean independence for the multivariate Gaussian**

Normally, uncorrelated variables DO NOT imply independence.

But the multivariate Gaussian is special in that uncorrelated variables do mean independence.

Reasoning:

If

The PDF

becomes

So the PDF can be separated into products

Therefore, the variables are independent.

**Central Limit Theorem**

Suppose there are *N* mutually independent random variables , with means and variances , then the sum

will be Gaussian if the *N* is large enough.

The variables are independent, so the mean and variance of the sum is

**Continuous PDFs**

|  |  |
| --- | --- |
|  | (2.85) Exponential distribution |
|  | (2.87) Beta distribution - multiple shapes.  is a hill.  is a tub. |
|  | (2.92) Gamma distribution - multiple shapes.  is the decay that will eventually dominate.  is a decay that adds to the decay.  can create an initial increase. |

**Dirichlet distribution**

This is the multivariate version of the beta distribution.

The *x* in the beta distribution has a 0 to 1 restriction.

Similarly, there is a restriction on the vector :

The *K* random variables lie on a (K-1) dimensional simplex.

So if *K=2*, then the 2D random variables lie on the line.

If *K*=3, then the 3D random variables lie on the plane.

The Dirichlet PDF:

|  |  |
| --- | --- |
|  | (2.95) |

## Stochastic Processes

**stochastic process**

The stochastic process is a sequence of random variables, also known as **random signal**.

Let *un* be the signal *u*, at a particular time *n*. This is a random variable because it can change from experiment to experiment.

The full stochastic process can be described by a joint pdf - the signal *u* at time indices *{n, m, ..., r}*. The pdf can have some kind of dependency on the time index.

**Mean and Autocorrelation**

Mean:

The *u­n* is a signal at a particular time *n* in the sequence. Its value can change from one experiment to the next. The E[*un*] gives the across an infinite number of experiments, which is why it's integrated against the *un* variable.

Autocovariance and autocorrelation:

These metrics relate the random variable *u­* at two different time indices.

The r(n,m) is the signal processing variation of the autocorrelation definition - the general autocorrelation definition normalizes the data by subtracting off the mean and dividing by the variance.

It is common to assume the signal has already been centered, that it has zero mean. Under this assumption, cov(n, m) = r(n, m).

**Strict-Sense Stationary**

A strict-sense stationary (SSS) stochastic process has a PDF that is not time sensitive.

**Wide-Sense Stationary (WSS)**

A wide-sense stationary (WSS) stochastic process has a constant mean and an autocorrelation that depends only in the difference of the time indices.

The autocorrelation sequence r(k) has special properties.

#1.

The \* means complex conjugate.

#2.

The r(0) is average power.

#3.

#4. The autocorrelation sequence is positive definite.

**Cross-correlation**

Let *un* and *vn* be two WSS processes. Define a new process

where

If two processes are uncorrelated,

Uncorrelatedness is a weaker condition than independence.

For WSS stochastic processes,

**Power Spectral Density S(w)**

The power spectral density S(w) is the Fourier transform of its autocorrelation sequence r(k).

The S(w) of a WSS stochastic process is real and nonnegative.

The area under the graph of S(w) is proportional to the power of the stochastic process.

**Convolution Sum**

The convolution sum describes input-output relationship for linear time invariant (LTI) systems.

The convolution pairs a weight *w* with an observation *u*. The weight and observation indices run in opposite direction. This is to support the casual situation, when the observations are all restricted to the past:

The weight *w0* is paired with the current time index:

The weights are the impulse response, because they show up when the input *u* is an impulse function. For example:

**(Theorem 2.1) - The power spectral density S(w) of the output**

The power spectral density S(w) of the output, *dn*, for a WSS stochastic process, *un*, is given by

**Equation (2.132) - Computation of the output autocorrelation**

The \* means convolution.

Theorem 2.1 is then obtained by the Fourier transforms:

## Information Theory

I(x)

1

P(x)

Rare events give more information. If P(x) = 1, then the event will always happen, and seeing it occur gives zero information. At the other extreme, if P(x) = 0, the event will never happen, and seeing it occur gives infinite information.

The use of the logarithm function allows information from two independent events to be added together.

Any base for the logarithm can be used. Base 2 logarithm means the information is in bits.

Suppose there's a single binary random variable . Both possibilities are equally likely, so .

If there are *k* binary random variables (*k* bits), so that the possibility of a particular word is

then the information conveyed is

Compared with before, having a longer word means more possibilities. Then each word is more rare and conveys more information.

**Mutual Information I(x, y)**

This quantity can be negative, if *P(x|y) < P(x)*. That is, when *y* makes *x* less likely, the mutual information becomes negative.

If *P(x|y) = P(x)*, then observing *y* does not help to explain *x*, and therefore *x* and *y* are independent. The mutual information would be log(1) = 0.

If *P(x|y) = 1*, then I(x,y) = I(x). This is the highest "mutual" information possible, when seeing *y* always leads to *x*.

**Conditional Information I(x|y)**

**Entropy**

Entropy is the average information over all possibilities:

Average mutual information between two random variables *x*, *y*:

Iav(x, y) is zero if *x* and *y* are independent, which results in P(x, y) = P(x) P(y).

Conditional entropy:

**Maximum Entropy**

Entropy is maximum for p(x) being a uniform distribution, when all possible values are equiprobable.

Entropy is zero if p(x) = 0 everywhere, except one value xi, where p(xi) = 1. There is no average information because the state of the world is "stuck".

# #3. Learning in Parametric Modeling

## Linear Regression

The equation is actually

The data has a 1 appended to it, so the whole thing is expressed as .

**(Normal Equations)**

The least squares loss function

Take the gradient of and set equal to zero:

Use: and

Note: you don't divide both sides by . There is actually not a single , but instead we have . The whole Σ (sum) expression is zero, but the individual expression might not be zero.

Putting things into matrix notation:

, this is due to being a constant so it can be moved to the back side.

The has no N index so it can come out of the summation expression.

Let n = 1 through 3:

Generalization:

where:

This is a convention in data science, where the data appears as the rows of a matrix (table).

matrix sizes:

*X* is [N x (m+1)]

is [(m+1) x 1]

is [N x 1]

**Classification**

One scheme is to use as the decision boundary. One set of data are assigned y = +1 and the other set of data is assigned y = -1.

As long as there is clear separation between two classes, this scheme works well.

Overlapping classes require alternative schemes.

**Discriminative Learning:**

For classification problem with input and label

In discriminative learning, only is considered.

**Generative Learning:**

The model is

There is one per class, which has to be learned.

For the final prediction

**Other Documentation**

**Linear Regression - Normal Equations**

For , the least square solution is .

See linear algebra notes CH 6, for more detail.

**Gradient Descent**

For the cost function , the gradient is .

See numerical algorithm notes CH 4, for more detail.

If there is too much data, run the gradient descent algorithm on a random subset of the data. This is known as stochastic gradient descent.

## Regularization

**(Ridge Regularization)**

A new loss function that penalizes the coefficients if they get too large:

The DC bias term, θ0, is not regulated. The is like the except the θ0 is missing.

Take the gradient of and set it to zero. This is very similar to the linear regression derivation, with the new addition of .

Use

Use the same derivation as before, except insert at the very end.

Combining with :

The XTX results in a square matrix the size of . The 3x3 situation looks like

**Inverse problems, ill conditioning, overfitting**

Machine learning is an inverse problem - where the goal is to find the model parameters based on input and output data.

Inverse problems are typically ill-posed.

Well-posed problems have a unique and stable solution.

Overfitting - a different set of training data causes wildly different (model parameters). We say this model has high variance.

**Bias-variance trade off example**

A zero variance example would be to just state that , without considering the data at all. No matter what the data actually is, . This model has high bias.

A high variance example would be to use a 10th degree polynomial to fit a regression problem that has 10 data points. The fit is perfect, but a different set of training data would likely generate different (model parameters).

**Other Documentation**

Ridge Regression (Tikhonov Regularization) is also documented in "Numerical Algorithm" notes, CH 4.

# #4 Mean-Square Error Linear Estimation

## Statistical View of Normal Equations

The following derivation of the normal equations takes a statistical viewpoint.

Let the prediction be

and the loss function be

Notice how it's no longer a long sum, but an expected value operation. Think of the E[ ] operator as an infinite sum that got averaged out.

**Expand the :**

The is a constant in the view of the E[ ] operator, so it can be factored out.

**Gradient of the loss function :**

This time around, it's the E[ ] operator that can be treated as a constant in the view of the operation.

Use for .

Use for .

The is a covariance matrix that is symmetric and positive semi-definite.

where is the input-output cross-correlation vector,

and is the covariance matrix.

**Setting the gradient to zero to find the optimal :**

|  |  |
| --- | --- |
|  | (4.5) Normal Equations |
|  | (4.6) |

**Derivation of :**

The goal here is to compare and , to see what is special about being at .

Previously, the expression has been expanded to be

Simplify and further using the and defined in (4.6):

Using equation (4.5) ,

|  |  |
| --- | --- |
|  | (4.9) |

Compare and by evaluating the expression :

reduce using equation (4.5)

to make the factorization work, the last two terms need to look like .

, where the covariance matrix Σx is assumed to be symmetric.

|  |  |
| --- | --- |
|  | (4.8) |

**Remarks 4.1 about**

Equation (4.9),

says that , due to Σx being positive semi-definite.

Equation (4.8),

says that , again due to Σx being positive semi-definite.

## Frequency Domain

**Normal equations for infinite impulse response (IIR) filters**

For input sequence *un* and output *dn*, the IIR filter model is

The optimal filter coefficients satisfy

|  |  |
| --- | --- |
|  | (4.45) |

The idea is E[(error)(any input)] = 0, so the error is not correlated with the input.

Derivation of the normal equations from equation (4.45):

The processes are WSS and the autocorrelation depends only on the time difference.

|  |  |
| --- | --- |
|  | (4.47) |

**Solution of IIR normal equations using the Fourier transform**

The term is a convolution.

Applying Fourier Transform to both sides:

|  |  |
| --- | --- |
|  | (4.48) |

## Kalman Filtering

**State-space model**

|  |  |
| --- | --- |
|  | (4.112) State equation |
|  | (4.113) Output equation |

The *n* is a time index and the various signals are time varying.

The is unobservable.

The F, H, and variables are observable, and the goal is to estimate the .

**Noise model**

The and are noise variables, with zero mean.

The processes are WSS stochastic processes, and certain covariances are assumed to be known:

Other covariances are assumed to be zero.

For all *n, m*:

where O is a matrix of all zeroes.

For :

**Estimators**

The estimation of occurs in two stages. The first stage produces and the second stage produces .

The first index is the current iteration and the second index is the data being used. means only data up to the (n-1) iteration is being used.

In terms of order of generation, it's:

**Error covariance matrices**

The (n)|(n-1) convention applies to other variables in the algorithm, the error vector and error covariance matrix *P*.

The and *P* are related to the unobservable , so these are initialized to something and then hopefully converges to their true values.

In terms of order of generation, it's

**Estimation equations**

**Partial Derivation** ( not shown)

The main goal is to from to

The estimate is purely assuming that is correct. There is also no noise in this estimation.

The estimate updates the with an error term . This estimate is using the latest observation .

The *K* value governs how much error feedback occur. This *K* value comes from computations involving the *P* matrices and has been mathematically shown to be optimal.

The Pn-1|n-1 is related to , while Pn|n-1 is related to . So first we need an expression relating to .

Now that Pn|n-1 is in terms of , which can be combined to form Pn-1|n-1.

The first and last term are definitions:

The middle two terms are zero due to differences in time index.

The generation is not shown in the book.

# #5. Stochastic Gradient Descent

## MS Gradient Descent Convergence

**Mean Square Gradient Descent:**

In chapter 4, the cost function for a prediction is

and the gradient was found to be

where is the input-output cross-correlation vector,

and is the covariance matrix.

Gradient descent on the mean square cost function would be

It's θnew = θold + (step size) \* (negative of the gradient).

**The gap between and :**

Define a quantity that represents the gap between the current and the final optimal .

The optimal comes from the normal equations defined in chapter 4.

Use the mean square gradient descent equation to get an expression for .

use

use

**Convergence of MS Gradient Descent depends on the Eigenvalues of Σx**

The covariance matrix Σx is symmetric and positive definite. It is guaranteed to have the orthogonal eigen decomposition:

, all eigenvalues are positive

*Q* = orthogonal matrix of eigenvectors

The Σx is a square matrix, which allows both and .

This is related to via a coordinate transform. When the gradient descent algorithm converges, both and goes to .

Expanding the :

For the j-th component, you have

If µ is too small, say µ = 0, then , and there is no convergence.

Alternatively, a µ that is too large, say , then , and there is no convergence either because the magnitude is not shrinking.

Conceptually, only a range of µ can lead to convergence:

µ

0

Each eigenvalue λj is establishing an upper (2/λj) limit on µ. The lowest of these upper limits is (2/λmax).

|  |  |
| --- | --- |
|  | (5.15) Condition for convergence |

For the j-th component, choosing will cause that component to go to zero - but only that j-th component will go to zero. Convergence requires all components to go to zero.

If sticking with a single step size µ all the time, then the optimal step size is a compromise between λmax and λmin:

µ

0

The best point is at the intersection of the falling λmin line and the rising λmax line.

|  |  |
| --- | --- |
|  | (5.16) |

Wider spread therefore results in slower convergence.

At the other extreme, if , then convergence can happen in one step with the optimal step size . In this case all components of is being set to zero in one step.

## LMS Algorithm

For the linear model , you have an estimate , and a new piece of data just came in.

Define error as

The LMS algorithm is minimizing error squared using gradient descent.

The iterative equation is then

where the "2" has been absorbed into the step size µ.

## Normalized LMS

For the linear model , you have an estimate , and a new piece of data just came in.

The NLMS solution is to fit the latest piece of data, while minimizing the deviation from the current estimate.

while requiring

The Lagrangian function is:

The optimized solution requires :

The minimum point satisfies both and the constraint.

Solve for the variable λ1 in terms of the other variable:

Use this expression for λ1 to get an iterative solution for :

For the actual algorithm, the book recommends:

|  |  |
| --- | --- |
|  | (Algorithm 5.3) |

The δ is to prevent division by zero and the µ is to cope with noise by not trusting the new data too much.

## Affine Projection Algorithm

For the linear model , you have an estimate , and a new piece of data just came in.

The affine projection algorithm solution is to fit the *q* latest pieces of data, while minimizing the deviation from the current estimate.

while requiring

which is abbreviated as

The solution mirrors the one for normalized LMS - you get the Lagrangian function, set its gradient to zero, find and expression for , and use that expression to get an iterative relation for .

The Lagrangian function:

This is like the NLMS derivation, except that the constraint is different. It's instead of just λ1 since there are multiple constraints.

:

Notice how the term has been rewritten. This enables the use of the property.

Ultimately, we only care about , not the value of . Therefore it's okay to absorb the "2" into and simplify the Lagrangian constraint as:

The system of equations for the minimum point:

Solving for :

Iterative relation for :

For the actual algorithm, the book recommends

|  |  |
| --- | --- |
|  | (Algorithm 5.2) |

The δ makes the inversion more numerically stable.

The µ is to cope with noise by not trusting the new data too much.

# Linear Models

**Lasso Regression**

The minimization problem is

solve it using gradient descent. It's the gradient plus . This second gradient is , depending on the sign of *xi*.

The lasso regression has a tendency to reduce weights to zero.

**Elastic Net**

This is a combination of the ridge and lasso regression.

**Early stopping regularization**

This uses the gradient descent for optimization, but stops iterating as soon as the error in the validation set starts to rise.

error

validation error

training error

iterations

stop here

# #7. Classification

## Logistic Regression

based on Theodoridis section 7.6

**Two-class model**

where is the data, is the coefficients, and the *w* are the class labels.

This model actually has theoretical justifications - if the data are Gaussian distributions with a common covariance matrix, then the log ratio of the posteriors is a linear function.

**Probability σ(t)**

It's two class, so .

Let ,

So let ,

**Total Probability P**

Adding subscripts: for the *n-th* data point we have output label , and .

The probability of the *n-th* event is

The likelihood function (total probability of all events):

**Maximize P 🡪 minimize**

Maximizing *P* is the same as maximizing . Stating things as a minimizing problem:

We want to choose that minimize *L*, so we say .

We are minimizing , so the gradient is .

So this gradient can be divided into two problems, and

**Hessian**

The book also calculates the Hessian and shows that this problem is convex.

**M-Class model**

The algorithm can be extended to M classes. However, the algorithm becomes much more complex, and separating 10 classes still require 9 vectors. So the benefits of the M-Class model is non-obvious.

## Fisher's Linear Discriminant

**Projection interpretation of the linear separator**

Given a linear separator:

the is interpreted as projecting the data point along the normal vector .

The goal is to find a such that the projected points have a large between-class distance and small within-class variance.

For a two-class scenario, the projected points have means *µ1* and *µ2*, and variances *σ12* and σ22. The mathematical goal is to maximize the Fischer's Discriminant Ratio (FDR):

**FDR numerator - the between-class scatter matrix Σb**

Let class 1 points have a mean of . The projected mean that the FDR is talking about is:

The notation is somewhat confusing --- keep in mind that there is a scalar mean µ1 that post projection, and there is a vector mean that is before projection.

The quantity is related to a quantity called the between-class scatter matrix Σb. More generally, for *m* classes:

where *µ0* is the overall mean:

The P(wm) is the probability of the class wm.

When both classes are equally probable, the FDR numerator is

|  |  |
| --- | --- |
|  | (7.54) |

**FDR Denominator - the within-class scatter matrix Σw**

Let *y* be the data points in class 1 projected onto the normal.

Again, the above variables are post projection scalar values. We first project, then subtract off the *µ1*. The same quantity is obtained by subtracting first, and then projecting:

The in this context is just the data in class 1. It does not refer to all of the data.

The is a random variable representing multiple data points, while and are definite quantities.

|  |  |
| --- | --- |
|  | (7.55) |

Again, as a reminder, the in the two E() phrases are referring to data points in the two different classes, not all of the data.

The quantity

is related to a quantity called the within-class scatter matrix Σw.

The green expression from equation (7.55) is what you would actually use to compute the Σw matrix. The following discussion is for theoretical values only.

For a random vector , the covariance is defined as:

, where is the mean.

More generally, for data split into *m* classes:

where P(wm) is the probability of class *m*. As before, the in refers to only a subset of the total data.

When the probabilities for the two classes are equal, we say that

|  |  |
| --- | --- |
|  | (7.56) |

**Maximizing the FDR**

Combining (7.54) and (7.56), we have

The quantity is called the Generalized Rayleigh Quotient. Maximizing this quotient maximizes the FDR.

Let and we would like to maximize *r*.

The following argument might not be mathematically correct.

Assume there is a solution that will maximize *r*.

The whole purpose of adopting instead of letting it remain a variable is that we can drop the from both sides of the equal sign:

This is then an eigenvalue problem, where *r* is the maximum eigenvalue of the matrix.

However, there is no need to compute the eigenvalue, as there is a short cut. Using

and the definition of Σb:

we have

For optimization purposes we only care about the direction of . The terms *r* and are constants that can be dropped out. The optimal is

|  |  |
| --- | --- |
|  | (7.58) |

## Classification Trees

**Entropy**

A tree splits a set *Xt* into two sets *XtY* and *XtY*. The new sets should be purer than the original set.

Say the data has four classes w1, w2, w3, and w4. We want *XtY* to contain say mostly w1 and w2, while *XtN* to contain mostly w3 and w4.

One impurity measure is entropy:

where the probability is approximated as

is the number of points from class *m* in Xt.

is the total number of points in *Xt*.

The minimum entropy = 0, when everything is concentrated in one class:

The decrease in entropy, after splitting data *t* into *tY* and *tN*, is

It's the original entropy I(t) minus the weighted average of the new entropies.

**Gini**

This is another impurity measure:

The advantage here is that it's faster to compute due to lack of the log2. But entropy is said to create more balanced trees.

**Guarding against overfitting**

If left to classify the training set, trees can keep splitting things up until the training error is zero, leading to overfitting.

Stop-splitting rules --- this approach tries to restrict the size of the tree. For example, restrict the depth of the tree, or put a lower bound on the number of elements at the leaf nodes, or limit the maximum number of leaf nodes.

Pruning a tree --- this approach grows the tree to a large size, and then get rid of some of the branches.

**Trees are understandable** --- major advantage of the trees is that its output can be interpreted by humans.

**Trees are unstable** --- a small change in the data can lead to a drastically different tree. If the decision at a node near the root changes, this change propagates to all its children and grandchildren.

The tree training algorithm itself might make some random decisions instead of exhaustively searching for the best entropy split.

**PCA benefits trees** --- Trees like to make perpendicular splits. Any kind of pre-treatment that orthogonalizes the data is helpful.

**How to train a tree? (my thoughts)**

Trying out all features, and computing all impurity values can be too time consuming.

I am thinking first sort the data according to the label, so that we get:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x1 | x2 | x3 | x4 | y |
| ... | ... | ... | ... | 0 |
| ... | ... | ... | ... | 0 |
| ... | ... | ... | ... | 0 |
| ... | ... | ... | ... | 1 |
| ... | ... | ... | ... | 1 |
| ... | ... | ... | ... | 1 |

Then analyze each column for overlap between the blue and orange numbers.

**Bagging** (Bootstrap Aggregating) --- Instead of training one tree on the original data set, train multiple trees on a bootstrap dataset. Each bootstrap sample is created by sampling from the real data *X* with replacement. These multiple trees vote on the classification problem.

**Random Forest** --- in addition to bagging, the approach chooses *F* randomly chosen features, and then uses the best of those features for splitting the data. So it's another way to make each tree in the forest be different.

When multiple trees are used, the model is no longer understandable by the human.

## Combining Classifiers

**Majority vote** --- this is used if the classifiers do not produce a posterior probability

**Arithmetic averaging**

Suppose there are three classifiers and they output probabilities P1, P2, and P3.

The probability of data belonging to class *wk* is

**Geometric averaging**

**Stacking**

This is a weighted average of the probabilities:

The weights need to be determined through an optimization process. During this optimization process, the above expression is not used. This is because using the training data as a metric of how good a classifier is would lead to overfitting.

Instead, the optimization process uses

where means the classifier *f1* has been trained with the *n-th* data point withheld. Then the optimization process uses *f1* on that withheld data point, as a measurement of how good that classifier is.

## AdaBoost

**Sum of classifiers**

The AdaBoost predictor is a sum of weak classifiers .

|  |  |
| --- | --- |
|  | (7.74) |

Each outputs .

Individually these weak perform poorly, but the sum has much better performance.

Strictly speaking, the should be written where each classifier has its own internal parameters .

The prediction process is .

**The exponential loss function**

Each is added one at a time and optimized one at a time.

The loss function being used is the exponential loss function

|  |  |
| --- | --- |
|  | (7.77) |

0

When the prediction is correct, the , and the loss function is small.

**Minimizing the loss function and optimizing**

We are optimizing one at a time, so the current can be expressed recursively as

Dropping the subscripts to simplify the notation:

This loss is totaled over the various data points in the training set:

|  |  |
| --- | --- |
|  | (7.80) |

The term has two outcomes, . So the has two outcomes, .

For the optimization of at this stage, the exponential loss function might as well be a two-level loss function, similar to the 0-1 loss function.

Minimizing the above loss is approximately the same thing as minimizing

|  |  |
| --- | --- |
|  | (7.82) |

The loss01() means that correct predictions produce zero loss, while bad predictions produce a loss of one. This should track the losses sufficiently close, and be easier to compute.

**Optimizing "a"**

In the loss function

the is either if the prediction is correct, or if the prediction is incorrect. Therefore, the loss function can be split into

The weights should be normalized to one. Then we can define

Minimize the loss by setting the derivative to zero

|  |  |
| --- | --- |
|  | (7.86) |

Note that P is defined as

With normalized, the *P* is the probability of being incorrect (with regards to the weighted data). If the prediction is just 50% correct, then . Over 50% correct will produce a positive *a*, while less than 50% correct will produces a negative *a*, which tries to make it correct by inverting the sign.

Note the log odds ratio will tend to infinity for the perfectly correct or perfectly incorrect case. Be sure to check for the perfectly correct case so it doesn't lead to .

**Updating the weights iteratively**

Let be the new weights for the next iteration, and be the variables and classifier from the current iteration.

|  |  |
| --- | --- |
|  | (7.87) |

After computing the *wn\_new* , be sure to normalize it.

**Why AdaBoost works**

The data is scaled according to the weights and the loss function, and the is a sum of functions. Points that are repeatedly classified wrong by multiple will be given exponentially more weight --- but this approach makes AdaBoost vulnerable to misclassified outliers.

Each has the coefficient . As mentioned before, the more correct the latest classifier in fixing the weighted data, the larger the *a*.

# #9. Sparsity

**, and norms**

number of nonzero components in

The norm is not a true norm because it does not obey the property for norms. Amplifying a vector by a factor *α* does not increase its norm.

In 2D world, maps out a diamond, while maps out the *x* and *y* axes.

**Underdetermined system solution is an affine set**

An underdetermined system

has infinitely many solutions.

An affine set is a linear subspace translated by a vector.

The solutions to an underdetermined system is the null space, denoted null(X), translated by a particular solution .

Suppose one particular solution to is

The dimension of the solution is , which is the number of free variables in .

**Optimizing an underdetermined system - geometric interpretation**

Problem statement:

such that

The *X* is a fat matrix, so has infinitely many solutions. The problem is to find a sparse solution, by minimizing the .

The solution to is a translation of the null space, so it's hyperplane that does not go through the origin.

The solution is found by expanding the diamond until it touches the hyperplane.

**The norm minimizer**

Problem statement:

such that

The solution is

|  |  |
| --- | --- |
|  | (9.19) |

This problem has a close form solution, but it does not produce sparse solutions.

**The norm minimizer**

This time it's .

The function is nonconvex and this problem cannot be solved efficiently.

**The norm minimizer**

Sometimes, will produce the same solution as .

Problem statement:

such that

The solution is to bring this into standard linear programming form, which is

such that .

Let

where means keeping the positive components of the vector while setting the negative components to zero.

The can then be expressed as

The can be expressed as

# #11. Support Vector Machine

## Kernels

**The Kernel Trick**

where a transform to the Hilbert space --- it's a place where a dot product exists.

So the kernel function is equivalent to doing transforms followed by a dot product.

**Example - Derive a kernel from a transformation (Theodoridis Example 11.1)**

So the kernel for is

**Example - Polynomial kernel - Show that is a kernel (from Caltech video #15)**

The general inhomogeneous polynomial kernel is .

Suppose the vectors are 2D.

Separate the variables to reveal the dot product. Use square root to split the coefficients equally.

**Example - RBF kernel - Show that is a kernel (from Caltech video #15)**

The general RBF kernel is . This example works on the simplest form to given an idea of what an infinite H space is.

The series expansion for

As before, split the terms in a symmetric manner.

So the transform corresponding to is

Due to the infinite series expansion involved, this is an example of a transform that has infinite dimensions.

## Regression

**Linear ϵ-insensitive loss function**

So losses below the ϵ limit receive no penalty. Larger losses grow linearly.

Compared with the least square metric, this metric would be less sensitive to outliers.

**Loss function statement**

For the linear model , we would like to minimize the loss function.

The loss function has absolute value in it, and is re-written to remove the absolute values.

The positive side error is:

The is a slack variable. It serves as an upper bound for the error.

When the error is low, we have slack = 0, as in .

The negative side error is:

Only one of these two slacks is allowed to be active at a time:

The total loss is

**Optimization problem statement**

Where the is regularization, and C is a regularization parameter.

Subject to:

This form, constraint > 0, is the standard form for optimization problems.

**Lagrangian Function**

L = objective function - (multiplier) \* Σ (constraints)

Lagrange multipliers are non-zero only when constraints are active.

|  |  |
| --- | --- |
|  | (11.40) |

**conditions**

|  |  |
| --- | --- |
|  | (11.44) |

|  |  |
| --- | --- |
|  | (11.45) |

|  |  |
| --- | --- |
|  | (11.46) |
|  | (11.47) |

**Dual Form**

The Lagrangian function is converted to the dual form because in the dual form, all references to appear as , which enables the use of the kernel trick.

The conversion from the Lagrangian function to the dual problem is done in color coded stages

Use equation (11.44):

The other terms are zeroes.

This is zero due to equation (11.45): .

This is due to equation (11.40):

Equation (11.40):

Equation (11.47):

When the slack sn > 0, , and

The dual problem:

where .

The bias term θ0 is obtained last, from the set of equations

So there's one parameter, θ0, and multiple equations. The θ0 is the average from these equations.

**Prediction Equation**

Equation (11.44) says

The prediction is , which expands to

## Classification

**The geometric meaning behind**

The is a hyperplane. The is normal of the plane, determining it's orientation, while θ0 determines the location of the plane.

is the normal because:

Suppose two points and are both on the plane.

The is a vector in the hyperplane, and .

The θ0 is related to the intercept of the plane.

Suppose the and .

The x1 axis intercept is at:

In general, the xn axis intercept is at .

**Distance between a data point and a plane**

The data point is not part of the plane:

z

α

The absolute value is necessary since the dot product produces negative values for obtuse angles.

Due to ,

**Ambiguity in the optimization problem**

The two definitions so far

has an ambiguity. If a works, then any multiple of works as well. For example, and 10θ0 will work, as in

This ambiguity is resolved by letting z take the form

for the point that is closest to the hyperplane.

The numerator limitation means

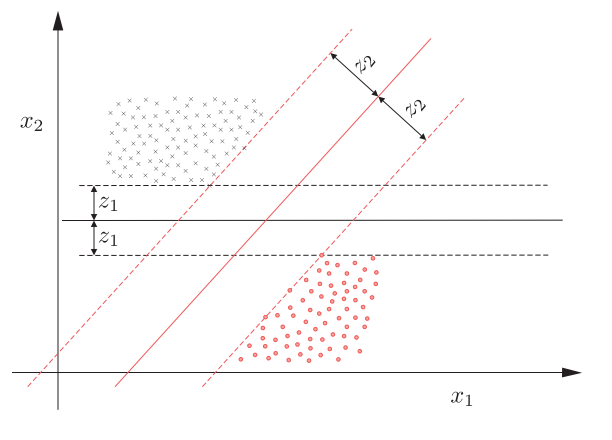
with the closest point satisfying

or

The goal is to force a unique in the optimization problem.

**Support Vector Machine Classifier - the linearly separable scenario**

In this case, there is a gap between two set of data.

  
Fig 11.16

The optimization goal is to find the plane that separates the two sets of points with the greatest margin. In the figure, z2 is better than z1.

The optimization procedure decides first, before θ0. So the procedure choose the plane orientation first by choosing , and then chooses θ0 to position the hyperplane in between two classes of points.

The margin width is

The classification procedure is

**Linearly separable case solution format**

The solution format is

Of the N Lagrange multipliers (λ), most of these will be zero. Only the points nearest to the classifier will have a non-zero Lagrange multiplier. These points satisfy . They sit on the margin that is exactly away from the hyperplane. These points are called support vectors.

The λn of the support vectors are the real parameters. The smaller the number of non-zero λ­n, the better the generalization is expected to be.

Prediction:

Bias term θ0:

The above is done for all constraints where λn ≠ 0. The final θ0 is the average of these θ0 estimates.

**Support Vector Machine Classifier - the linearly non-separable scenario**

The penalty model:

error

1

The green center is the prediction made by the equation.

Correctly label points that lie beyond the margin have no (margin) error. Starting at the margin, the error grows linearly. So points within the margin will be classified correctly - but they contribute error.

This error model is called the **hinge loss function**.

**Error (slack) variables sn**

The error is

In the optimization problem statement, this is modeled using slack variables, similar to the regression case.

The difference is that there is just one slack variable, just , no . That's because the yn is ±1. So for data points that are correctly predicted, the quantity is always positive.

**Optimization Problem Statement**

The objective function to minimize is

The is the objective, while the is the regularization. It's custom to put the regularization constant *C* with the term.

The constraints are

Note how the definition is actually , but we are using the inequality, upper bounded by . The transition to the dual form of the problem require the use of inequalities.

**Lagrangian Function**

|  |  |
| --- | --- |
|  | (11.84) |

**Dual problem formulation**

The minimum of the Lagrangian function is at:

|  |  |
| --- | --- |
|  | (11.80) |
|  | | (11.81) |
|  | | (11.82) |

The transition from the Lagrangian function to its dual form is color coded:

Use equation (11.80):

The other terms are zeroes

Use equation (11.82), , to get a zero

Use equation (11.81), , to get a zero

Restriction:

is a restriction for Lagrange multipliers.

is due to equation (11.82), , and .

The dual problem is

subject to:

**Nature of λn**

These points are correctly classified and lies beyond the margin. They do not affect the operation of the support vector machine.

These points lie to the left of the hinge. They include points that are incorrectly classified, as well as points that are correctly classified, but do not satisfy the hard margin.

For these points, the slack error . This means we are not on the border of the constraint, causing to be zero via equation (11.84) .

Equation (11.82), , and , means .

These points lie right at the turning point of the hinge, at the margin and therefore shape the .

The error at this point is zero, so , and so we are at the border of the , making that an active constraint. The is possible, via equation (11.84) .

Equation (11.82), , and , means λn has a range 0 to *C*.

**RBF Kernel and the Nearest Neighbor Algorithm**

The prediction equation is:

When using the RBF kernel:

For any date point , the output is mainly influenced by the points that are close to , such that the term is large. Due to the exponential function, this distance effect decays rapidly as we move away from existing data points . The support vector machine classifier, when paired with the RBF kernel, is a mathematical refinement of the nearest neighbor algorithm.

# #14. Monte Carlo Methods

## Central Limit Theorem for Integration

The goal is to evaluate the integral

The is treated as a random variable. It takes on the distribution .

This expected value is evaluated by

where is generated to follow .

The central limit theorem says the will approach a bell shape curve, with mean:

and variance:

The key is that the variance is decreasing by , regardless of the dimensionality of the . This makes the approach scale well with increasing dimensions.

## Random Number Generation

**Generating uniformly distributed random numbers using the modulus function**

The simple scheme uses

where α = 75 and *M* = 231 - 1.

The *M* is a large prime number and the *zi* series should then be "stuck" between 1 through *M*-1.

**Function Inversion**

Let be the CDF of random variable *X*.

The *u* is uniformly distributed, so the *x* can be generated via

**Function Transformation**

This procedure generates random variables R and . Then use a function transform

to generate random variables X and Y.

The idea is that X and Y are hard to generate, but R and are easy to generate.

For this scheme to work, the pdf of various random variables, and , are related by the change of variable equations:

where

**Rejection Sampling**

The is the target distribution.

The proposal distribution is something that is easy to generate, and everywhere.

Draw . This is the green bar in the picture above.

Retain the sample if . This is the blue region.

## Importance Sampling

The goal is to evaluate

but we are not able to draw from . In addition, we only know the form of , not its normalization constant.

So we know , but not the value of *Z*.

Let be a distribution that we can draw from

where

The constant *Z* can be determined as:

where .

Eliminating the Z:

|  |  |
| --- | --- |
|  | (14.28) |

## Markov Chains

A Markov chain is a sequence of random variables

**Transitional Probabilities Matrix**

A matrix encodes the transitional probabilities:

Content of the matrix:

means going from state #3 to state #2.

means the probability of ending at state #3 is:

P(starting at #1 and ending at #3)   
+ P(starting at #2 and ending at #3)   
+ P(starting at #3 and ending at #3)

because if you start at state #3, you must end at state #1, #2, or #3.

Notation for going from state to state :

where

The *P* matrix is a **stochastic matrix** with following properties:

All entries are nonnegative.

Each column adds up to one. This means the left eigenvector is all ones, with λ = 1.

**Ergodic Markov Chains**

These chains have a unique **invariant distribution** (steady state) where

We arrive at this steady state by multiplying by *P* for infinitely many times:

The transition matrix P has one eigenvalue that is one, and all other eigenvalues are smaller than one:

The matrix *P* can be diagonalized to look like:

When raised to the ∞ power, the terms all go to zero because they are less than one.

The P(large number) converges to a rank 1 matrix due to all the diagonal terms going to zero.

All the columns of P∞ must be the same, since it is a rank one matrix. Furthermore,

We have , so .

**Markov chain for continuous state spaces**

The probability transition, from to , is described by an integral instead of a matrix

## The Metropolis Method

**Metropolis Algorithm**

Let the desired distribution be

Choose a proposal distribution

Choose

Compute the acceptance ratio α

Use a random number generator to decide whether to accept the new value , or stick with the old value . If the new value is more probable, then α = 1, and it's always accepted. The chance of accepting the new value is lower if the old is more probable.

**Metropolis Algorithm Basic Example**

The book's argument for the convergence of the method is incomplete. A complete justification seems quite complicated and will involve the use of continuous time Markov chains - as the Metropolis algorithm does work on continuous probability distributions.

So I came up with this simple example, which enables simple convergence justification.

Suppose we want to generate the following random variable:

|  |  |
| --- | --- |
| X | p(x) |
| 1 | 1/6 |
| 2 | 2/6 |
| 3 | 3/6 |

using the Metropolis method, with a uniform proposal distribution.

Probability transition matrix:

This matrix is generated one column at a time.

The first column is easiest: P(1|1), P(2|1), P(3|1) --- we are current at x = 1 and there is a 1/3 chance of choosing states {1, 2, 3}. Moving to other numbers is always accepted, because they are more probable.

The second column: P(1|2), P(2|2), P(3|2)

P(3|2) --- it's 1/3 since moving from 2 to 3 is always accepted due to higher probability of 3.

P(1|2) --- it's 1/3 chance of picking #1 out of {1, 2, 3}, and there is a 1/2 chance of accepting this pick.

P(2|2) --- This is the key. There is a 1/3 chance of picking #2 out of {1, 2, 3}. Another way of going from #2 to #2 is that #1 is picked and then rejected. The probability of this is (1/3)(1/2).

The third column: P(1|3), P(2|3), P(3|3)

P(3|3) --- The self-transition is always the key. There are now three ways to go from #3 to #3:

Choose #1 and reject: (1/3)(2/3)

Choose #2 and reject: (1/3)(1/3)

Choose #3: 1/3

Convergence check:

**Continuous state-space setup**

The following describes the transition from to :

The is the rejection probability.

Notice the rejection probability is only triggered for self-transitions --- the δ() function only triggers when .

The rejection probability collects all attempted to transitions that got rejected.

# #18. Neural Networks

## The Perceptron

**Linear predictor**

The perceptron is a linear predictor:

**Learning algorithm**

For any incorrectly classified point

where *µ* is a step size that needs to be gradually decreased.

**How the learning algorithm works**

When is set correctly, things would look like

The orange points are +1 and form an acute angle with the normal vector .

When a +1 point is misclassified:

The correction

is intended to pull the normal in the direction of , so to tilt the plane such that its left edge will dip to include .

Conversely, if a -1 point is misclassified

the correction pushes the normal away from , so the plane will tilt such that its left edge will rise above the misclassified point.

The learning algorithm only converges if the points are indeed linearly separable. Else the plane will just keep moving, satisfying some misclassified points while misclassifying other points in the process.

## Backpropagation

**Activation function *f(x)***

This is a differentiable replacement for .

Logistic sigmoid --- output range is 0 to +1

tanh --- output range is -1 to +1

**Feed-forward neural network**

*f*

*f*

*f*

*f*

*f*

The superscript is used to denote the layer number.

The subscript denotes the different neural nets within a layer.

There are lots of terms that need to be optimized --- 5 in this picture.

There is a single set of input that is applied to all neural nets on the first layer. In contrast, there can be more than one output --- *y1* and *y2* in this picture.

Feed-forward means the data only flows one way, from left to right.

**Optimization overview**

Initialize the variables to something random, and something small. The gradient descent works best in the non-saturated regions of the sigmoid and tanh functions. See remarks 18.2 for more details on data preparation.

Put in a SINGLE data point . The output will be different from the real output . Update the variables based on this output difference.

Note there are many variables. This update has to be done for all of them. For example:

Using the book's notation:

To evaluate the above derivative, the variable *z* is introduced

*f*

*f*

*f*

*f*

*f*

Note the z is following the same superscript and subscript convention as the variables.

The next section deals with how to compute δ.

As for :

For the very first layer, , so

As a reminder the has a bias term, so the input vector has a 1 appended in front.

For the second layer:

In general:

|  |  |
| --- | --- |
|  | (18.18), (18.19) |

The book refers to this vector as . It's the intermediate output, from the previous layer, which is (r-1).

**δ of the final layer**

The final layer is denoted r = L, meaning Last.

The only cares about the k=j term.

|  |  |
| --- | --- |
|  | (18.24) |

**δ of the inner layers**

*f*

*f*

*f*

*f*

*f*

Suppose we want to calculate :

Note that J depends on both and .

Now express layer 2 in terms of layer 1:

Since we are taking partial derivative with respect to , terms like doesn't matter.

Generalization:

|  |  |
| --- | --- |
|  | (18.30) |

The important thing to note is that δr-1 requires δr. The δ is computed at the final layer first, and hence the name of the algorithm: backpropagation.

**Multi data point update**

The equations above assume a single data input is being used to update the variables.

If *N* data points are used to update the variables, use equation (18.21):

where L is the total number of layers.

Note that each data point will get its own δ variable.

Updating using a single data input actually works pretty well in this case. The problem is non-convex with lots of local minimums. So the accuracy of the gradient is not as important as in convex problems.

# Dimensionality Reduction

## Principle Component Analysis

Notes about principle component analysis are scattered across different documentations.

**Linear Algebra (Lay) - CH 7, last section, "Applications to Image Processing"**

The explanation here is that the principle component analysis transform will make the covariance matrix XXT a diagonal matrix. That means after the transform, the various axes are not related to one another.

Be sure to review the singular value decomposition section if necessary.

**Numerical Algorithms (Solomon) - CH 6, section 1, "Applications", subsection: "Principle component of a data set"**

The approach here is to view the principle component as an optimization problem. The principle component is chosen such that if data is projected along this vector, the loss of information will be minimum.

**Numerical Linear Algebra (Trefethen)**

No direct mentioning of principle component analysis, but material in Part I, "Fundamentals", helps with linear algebra manipulations.

Section 5, "More on the SVD", subsection: "low-rank approximations", shows how the SVD is used to create a series that approximates a matrix A. This approximation is used as a way to interpret the principle component analysis.

**Relationship between principle component analysis, SVD, and low-rank approximation**

The singular value decomposition

where each column in *X* is a single piece of data.

The *U* can be viewed as a projection basis, and the *VT* is the "compressed" data. For example, let

Now just look at the very first column of

Therefore is being described by the basis set .

Assuming *σ1* is the largest singular value, and *σ2* is the second largest singular value, we can get a reasonable approximation by keeping the first few largest singular values.

***XXT* and *XTX* have the same eigenvalues and their eigenvectors differ by a factor *X***

SVD requires finding the eigenvalue and eigenvectors of *XTX*

The covariance matrix approach diagonalizes *XXT*.

Both approaches involve finding eigenvalue sand eigenvectors. Depending on the dimension of X, one approach would involve less computation. For example, if *X* is a tall matrix, then *XTX* would result in a smaller matrix.

The relationship between *XTX* and *XXT* eigenvectors:

Apply the associative property on the left side.

# Other Algorithms

## K-Means Clustering

Assume the center of clusters are . Assign the data points to those clusters, resulting in sets of points .

Assume the sets of points are . Compute cluster centers .

Repeat and the process will converge to a local minimum.