The maximum margin classifier

If you completed the exercises for AI I then you’ll know that linear classifiers have a very simple geometry. For

\[ f(x) = w^T x + b \]

For \( x' \) on one side of the line \( f(x) = 0 \) we have \( f(x') > 0 \) and on the other side \( f(x') < 0 \).

Draw the boundary as far away from the examples as possible. The distance \( \gamma \) is the margin, and this is the maximum margin classifier.

Problems:

- Given the usual training data \( s \), can we now find a training algorithm for obtaining the weights?
- What happens when the data is not linearly separable?

To derive the necessary training algorithm we need to know something about constrained optimization.

We can address the second issue with a simple modification. This leads to the Support Vector Machine (SVM).

Despite being decidedly “non-Bayesian” the SVM is currently a gold-standard:

Constrained optimization

You are familiar with maximizing and minimizing a function $f(x)$. This is unconstrained optimization.

We want to extend this:

1. Minimize a function $f(x)$ with the constraint that $g(x) = 0$.
2. Minimize a function $f(x)$ with the constraints that $g(x) = 0$ and $h(x) \geq 0$.

Ultimately we will need to be able to solve problems of the form: find $x_{\text{opt}}$ such that

$$x_{\text{opt}} = \arg\min_x f(x)$$

under the constraints

$$g_i(x) = 0 \text{ for } i = 1, 2, \ldots, n$$

and

$$h_j(x) \geq 0 \text{ for } j = 1, 2, \ldots, m.$$**

Step 1: introduce the Lagrange multiplier $\lambda$ and form the Langrangian

$$L(x, y, \lambda) = f(x, y) - \lambda g(x, y)$$

Necessary condition: it can be shown that if $(x', y')$ is a solution then $\exists \lambda'$ such that

$$\frac{\partial L(x', y', \lambda')}{\partial x} = 0 \quad \frac{\partial L(x', y', \lambda')}{\partial y} = 0$$

So for our example we need

$$2 + y + \lambda = 0$$
$$2y + x + 2\lambda = 0$$
$$x + 2y - 1 = 0$$

where the last is just the constraint.

Step 2: solving these equations tells us that the solution is at:

$$(x, y) = (4, -\frac{3}{2})$$

With multiple constraints we follow the same approach, with a Lagrange multiplier for each constraint.

For example:

Minimize the function

$$f(x, y) = -\left(2x + y^2 + xy\right)$$

subject to the constraint

$$g(x, y) = x + 2y - 1 = 0.$$
Constrained optimization

How about the full problem? Find
\[ x_{opt} = \arg\min_x f(x) \text{ such that } g_i(x) = 0 \text{ for } i = 1, 2, \ldots, n \]
\[ h_j(x) \geq 0 \text{ for } j = 1, 2, \ldots, m \]

The Lagrangian is now
\[ L(x, \lambda, \alpha) = f(x) - \sum_{i=1}^{n} \lambda_i g_i(x) - \sum_{j=1}^{m} \alpha_j h_j(x) \]
and the relevant necessary conditions are more numerous.

The necessary conditions now require that when \( x' \) is a solution \( \exists \lambda', \alpha' \) such that
\[ 1. \quad \frac{\partial L(x', \lambda', \alpha')}{\partial x} = 0. \]
\[ 2. \text{ The equality and inequality constraints are satisfied at } x'. \]
\[ 3. \quad \alpha' \geq 0. \]
\[ 4. \quad \alpha' h_j(x') = 0 \text{ for } j = 1, \ldots, m. \]

These are called the Karush-Kuhn-Tucker (KKT) conditions.
The KKT conditions tell us some important things about the solution.
We will only need to address this problem when the constraints are all inequalities.

Constrained optimization

What we’ve seem so far is called the primal problem.

There is also a dual version of the problem. Simplifying a little by dropping the equality constraints.

1. The dual objective function is
\[ \hat{L}(\alpha) = \inf_x L(x, \alpha). \]
2. The dual optimization problem is
\[ \max_{\alpha} \hat{L}(\alpha) \text{ such that } \alpha \geq 0. \]

Sometimes it is easier to work by solving the dual problem and this allows us to obtain actual learning algorithms.

We won’t be looking in detail at methods for solving such problems, only the minimum needed to see how SVMs work.


The maximum margin classifier

It turns out that with SVMs we get particular benefits when using the kernel trick.
So we work, as before, in the extended space, but now with:
\[ f_{w, w_0}(x) = w_0 + w^T \Phi(x) \]
\[ h_{w, w_0}(x) = \text{sgn} (f_{w, w_0}(x)) \]
where
\[ \text{sgn}(z) = \begin{cases} 
+1 & \text{if } z > 0 \\
-1 & \text{otherwise.} 
\end{cases} \]

Note the following:

1. Things are easier for SVMs if we use labels \( \{+1, -1\} \) for the two classes. (Previously we used \( \{0, 1\} \).)
2. It also turns out to be easier if we keep \( w_0 \) separate rather than rolling it into \( w \).
3. We now classify using a “hard” threshold \( \text{sgn} \), rather than the “soft” threshold \( \sigma \).
The maximum margin classifier

Consider the geometry again. **Step 1:**

1. We're classifying using the sign of the function
   \[ f_{w,w_0}(x) = w_0 + w^T \Phi(x). \]
2. The distance from any point \( \Phi(x') \) in the extended space to the line is
   \[ \frac{|f_{w,w_0}(x')|}{||w||}. \]

**Solution, version 1:** convert to a constrained optimization. For any \( c \in \mathbb{R} \)
   \[ f_{w,w_0}(x) = 0 \iff w_0 + w^T \Phi(x) = 0 \]
   \[ \iff cw_0 + cw^T \Phi(x) = 0. \]

That means you can fix \( ||w|| \) to be anything you like! (Actually, fix \( ||w||^2 \) to avoid a square root.)

---

**Step 2:**

- But we also want the examples to fall on the correct side of the line according to their label.
- Noting that for any labelled example \((x_i, y_i)\) the quantity \( y_i f_{w,w_0}(x_i) \) will be positive if the resulting classification is correct . . .
- . . . the aim is to solve:

\[
(w, w_0) = \arg \max_{w, w_0} \min_i \frac{y_i f_{w,w_0}(x_i)}{||w||}.
\]

That means you can fix \( ||w|| \) to be anything you like! (Actually, fix \( ||w||^2 \) to avoid a square root.)
The maximum margin classifier

**Solution, version 2:** still, convert to a constrained optimization, but instead of fixing $||w||$:

Fix $\min \{y_i f_{w, w_0}(x_i)\}$ to be anything you like!

![Diagram](image)

**Version 2:**

$$(w, w_0) = \underset{w, w_0}{\text{argmin}} \frac{1}{2}||w||^2$$
subject to the constraints

$$y_i f_{w, w_0}(x_i) \geq 1, i = 1, 2, \ldots, m.$$ 

(This works because maximizing $\gamma$ now corresponds to minimizing $||w||$.)

The maximum margin classifier

Working these out is easy:

$$\frac{\partial L(w, w_0, \alpha)}{\partial w} = \frac{\partial}{\partial w} \left( \frac{1}{2} ||w||^2 - \sum_{i=1}^{m} \alpha_i (y_i f_{w, w_0}(x_i) - 1) \right)$$

$$= w - \sum_{i=1}^{m} \alpha_i y_i \frac{\partial}{\partial w} (w^T \Phi(x_i) + w_0)$$

$$= w - \sum_{i=1}^{m} \alpha_i y_i \Phi(x_i)$$

and

$$\frac{\partial L(w, w_0, \alpha)}{\partial w_0} = -\frac{\partial}{\partial w_0} \left( \sum_{i=1}^{m} \alpha_i y_i f_{w, w_0}(x_i) \right)$$

$$= -\frac{\partial}{\partial w_0} \left( \sum_{i=1}^{m} \alpha_i y_i (w^T \Phi(x_i) + w_0) \right)$$

$$= -\sum_{i=1}^{m} \alpha_i y_i.$$ 

The maximum margin classifier

We’ll use the second formulation. (You can work through the first as an exercise.)

The constrained optimization problem is:

Minimize $\frac{1}{2} ||w||^2$

such that

$$y_i f_{w, w_0}(x_i) \geq 1 \text{ for } i = 1, \ldots, m.$$ 

Referring back, this means the Lagrangian is

$$L(w, w_0, \alpha) = \frac{1}{2} ||w||^2 - \sum_{i=1}^{m} \alpha_i (y_i f_{w, w_0}(x_i) - 1)$$

and a necessary condition for a solution is that

$$\frac{\partial L(w, w_0, \alpha)}{\partial w} = 0 \quad \frac{\partial L(w, w_0, \alpha)}{\partial w_0} = 0.$$ 

The maximum margin classifier

Equating those to 0 and adding the KKT conditions tells us several things:

1. The weight vector can be expressed as

$$w = \sum_{i=1}^{m} \alpha_i y_i \Phi(x_i)$$

with $\alpha \geq 0$. This is important: we’ll return to it in a moment.

2. There is a constraint that

$$\sum_{i=1}^{m} \alpha_i y_i = 0.$$ 

This will be needed for working out the dual Lagrangian.

3. For each example

$$\alpha_i [y_i f_{w, w_0}(x_i) - 1] = 0.$$
The maximum margin classifier

The fact that for each example
\[ \alpha_i[y_if_{w,w_0}(x_i) - 1] = 0 \]
means that:

Either \( y_if_{w,w_0}(x_i) = 1 \) or \( \alpha_i = 0 \).

This means that examples fall into two groups.

1. Those for which \( y_if_{w,w_0}(x_i) = 1 \).
   As the constraint used to maximize the margin was \( y_if_{w,w_0}(x_i) \geq 1 \) these are the examples that are closest to the boundary. They are called support vectors and they can have non-zero weights.
2. Those for which \( y_if_{w,w_0}(x_i) \neq 1 \).
   These are non-support vectors and in this case it must be that \( \alpha_i = 0 \).

Remember that
\[ w = \sum_{i=1}^{m} \alpha_i y_i \Phi(x_i) \]
so the weight vector \( w \) only depends on the support vectors.

ALSO: the dual parameters \( \alpha \) can be used as an alternative set of weights. The overall classifier is
\[ h_{w,w_0}(x) = \text{sgn} \left( w_0 + w^{T} \Phi(x) \right) \]
\[ = \text{sgn} \left( w_0 + \sum_{i=1}^{m} \alpha_i y_i \Phi^T(x_i) \Phi(x) \right) \]
\[ = \text{sgn} \left( w_0 + \sum_{i=1}^{m} \alpha_i y_i K(x_i, x) \right) \]
where \( K(x_i, x) = \Phi^T(x_i) \Phi(x) \) is called the kernel.

The kernel is computing
\[ K(x, x') = \Phi^T(x) \Phi(x') \]
\[ = \sum_{i=1}^{k} \phi_i(x) \phi_i(x') \]
This is generally called an inner product.
The maximum margin classifier

If it’s a hard problem then you’ll probably want lots of basis functions so \( k \) is BIG:

\[
h_{w,w_0}(x) = \text{sgn}\left( w_0 + w^T \Phi(x) \right) = \text{sgn}\left( w_0 + \sum_{i=1}^{k} w_i \phi_i(x) \right) = \text{sgn}\left( w_0 + \sum_{i=1}^{m} \alpha_i y_i \Phi(x_i) \Phi(x) \right) = \text{sgn}\left( w_0 + \sum_{i=1}^{m} \alpha_i y_i \Phi(x_i) \Phi(x) \right)
\]

What if \( K(x,x') \) is easy to compute even if \( k \) is HUGE? (In particular \( k \gg m \).)

1. We get a definite computational advantage by using the dual version with weights \( \alpha \).
2. *Mercer’s theorem* tells us exactly when a function \( K \) has a corresponding set of basis functions \( \{ \phi_i \} \).

Maximum margin classifier: the dual version

Collecting together some of the results up to now:

1. The Lagrangian is
   \[
   L(w, w_0, \alpha) = \frac{1}{2}||w||^2 - \sum_i \alpha_i (y_i f_{w,w_0}(x_i) - 1).
   \]
2. The weight vector is
   \[
   w = \sum_i \alpha_i y_i \Phi(x_i).
   \]
3. The KKT conditions require
   \[
   \sum_i \alpha_i y_i = 0.
   \]

It’s easy to show (this is an exercise) that the dual optimization problem is to maximize

\[
\hat{L}(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(x_i, x_j)
\]

such that \( \alpha \geq 0 \).

Support Vector Machines

There is one thing still missing:

*Problem:* so far we’ve only covered the linearly separable case.

Even though that means linearly separable in the extended space it’s still not enough.

By dealing with this we get the Support Vector Machine (SVM).
Support Vector Machines

Fortunately a small modification allows us to let some examples be misclassified.

\[ x^2 \]

\[ x^1 \]

\[ f(x') \geq 1 - \xi_i \]

We introduce the slack variables \( \xi_i \), one for each example.

Although \( f_{w_0}(x') < 0 \) we have \( f_{w_0}(x') \geq 1 - \xi_i \) and we try to force \( \xi_i \) to be small.

There is a further new parameter \( C \) that controls the trade-off between maximizing the margin and controlling misclassification.

Once again, the theory of constrained optimization can be employed:

1. We get the same insights into the solution of the problem, and the same conclusions.
2. The development is exactly analogous to what we’ve just seen.

However as is often the case it is not straightforward to move all the way to having a functioning training algorithm.

For this some attention to good numerical computing is required. See:

Supervised learning in practice

We now look at several issues that need to be considered when applying machine learning algorithms in practice:

- We often have more examples from some classes than from others.
- The obvious measure of performance is not always the best.
- Much as we’d love to have an optimal method for finding hyperparameters, we don’t have one, and it’s unlikely that we ever will.
- We need to exercise care if we want to claim that one approach is superior to another.

This part of the course has an unusually large number of Commandments. That’s because so many people get so much of it wrong!

Supervised learning

As usual, we want to design a classifier.

\[ h_{\theta}(x) \]

It should take an attribute vector

\[ x^T = [x_1 \ x_2 \ \cdots \ x_n] \]

and label it.

We now denote a classifier by \( h_{\theta}(x) \) where \( \theta^T = (w \ p) \) denotes any weights \( w \) and (hyper)parameters \( p \).

To keep the discussion and notation simple we assume a classification problem with two classes labelled +1 (positive examples) and −1 (negative examples).

Machine Learning Commandments

We’ve already come across the Commandment:

- Thou shalt try a simple method. Preferably many simple methods.

Now we will add:

- Thou shalt use an appropriate measure of performance.

Unfortunately that turns out not to be enough, so a new box has been added.
Measuring performance

How do you assess the performance of your classifier?

1. That is, after training, how do you know how well you’ve done?
2. In general, the only way to do this is to divide your examples into a smaller training set $s$ of $m$ examples and a test set $s'$ of $m'$ examples.

The GOLDEN RULE: data used to assess performance must NEVER have been seen during training.

This might seem obvious, but it was a major flaw in a lot of early work.

How do we choose $m$ and $m'$? Trial and error!

Assume the training is complete, and we have a classifier $h_\theta$ obtained using only $s$. How do we use $s'$ to assess our method’s performance?

The obvious way is to see how many examples in $s'$ the classifier classifies correctly:

$$\hat{\text{err}}(h_\theta) = \frac{1}{m'} \sum_{i=1}^{m'} \mathbb{I}[h_\theta(x'_i) \neq y'_i]$$

where

$$s' = [(x'_1, y'_1), (x'_2, y'_2), \ldots, (x'_{m'}, y'_{m'})]^T$$

and

$$\mathbb{I}[z] = \begin{cases} 1 & \text{if } z = \text{true} \\ 0 & \text{if } z = \text{false} \end{cases}$$

This is just an estimate of the probability of error and is often called the accuracy.

Unfortunately it is often the case that we have unbalanced data and this can make such a measure misleading. For example:

If the data is naturally such that almost all examples are negative (medical diagnosis for instance) then simply classifying everything as negative gives a high performance using this measure.

We need more subtle measures.

For a classifier $h$ and any set $s$ of size $m$ containing $m^+$ positive examples and $m^-$ negative examples...

Define

1. The true positives $P^+ = \{(x, +1) \in s | h(x) = +1\}$, and $p^+ = |P^+|
2. The false positives $P^- = \{(x, -1) \in s | h(x) = +1\}$, and $p^- = |P^-|
3. The true negatives $N^+ = \{(x, -1) \in s | h(x) = -1\}$, and $n^+ = |N^+|
4. The false negatives $N^- = \{(x, +1) \in s | h(x) = -1\}$, and $n^- = |N^-|

Thus $\hat{\text{err}}_s(h) = (p^+ + n^+)/m$.

This allows us to define more discriminating measures of performance.
Performance measures

Some standard performance measures:

1. Precision $\frac{p^+}{p^+ + p^-}$.
2. Recall $\frac{p^+}{p^+ + n^-}$.
3. Sensitivity $\frac{p^+}{n^+ + p^-}$.
4. Specificity $\frac{n^-}{n^- + p^+}$.
5. False positive rate $\frac{p^-}{p^- + n^+}$.
6. Positive predictive value $\frac{p^+}{p^+ + n^+}$.
7. Negative predictive value $\frac{n^-}{n^- + p^+}$.
8. False discovery rate $\frac{p^-}{p^- + p^+}$.

In addition, plotting sensitivity (true positive rate) against the false positive rate while a parameter is varied gives the receiver operating characteristic (ROC) curve.

Performance measures

The following specifically take account of unbalanced data:

1. Matthews Correlation Coefficient (MCC)
   \[ MCC = \frac{p^+n^+ - p^-n^-}{\sqrt{(p^+ + p^-)(n^+ + n^-)(p^+ + n^-)(n^+ + p^-)}} \]
2. F1 score
   \[ F1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \]

When data is unbalanced these are preferred over the accuracy.

Machine Learning Commandments

Thou shalt not use default parameters.
Thou shalt not use parameters chosen by an unprincipled formula.
Thou shalt not avoid this issue by clicking on 'Learn' and hoping it works.
Thou shalt either choose them carefully or integrate them out.

Bad hyperparameters give bad performance

![Diagrams showing the impact of different hyperparameters on model performance.](image)
Bad hyperparameters give bad performance

Answer: try different values and see which values give the best (estimated) performance.

There is however a problem:

If I use my test set $s'$ to find good hyperparameters, then I can’t use it to get a final measure of performance. (See the Golden Rule above.)

Solution 1: make a further division of the complete set of examples to obtain a third, validation set:

Now, to choose the value of a hyperparameter $p$:

For some range of values $p_1, p_2, \ldots, p_n$:

1. Run the training algorithm using training data $s$ and with the hyperparameter set to $p_i$.
2. Assess the resulting $h_{\theta}$ by computing a suitable measure (for example accuracy, MCC or F1) using $v$.

Finally, select the $h_{\theta}$ with maximum estimated performance and assess its actual performance using $s'$.

This was originally used in a similar way when deciding the best point at which to stop training a neural network.

The figure shows the typical scenario.
Crossvalidation

The method of crossvalidation takes this a step further.

We our complete set into training set $s$ and testing set $s'$ as before.

But now instead of further subdividing $s$ just once we divide it into $n$ folds $s^{(i)}$ each having $m/n$ examples.

Typically $n = 10$ although other values are also used, for example if $n = m$ we have leave-one-out cross-validation.

Crossvalidation

Two further points:

1. What if the data are unbalanced? Stratified crossvalidation chooses folds such that the proportion of positive examples in each fold matches that in $s$.
2. Hyperparameter choice can be done just as above, using a basic search.

What happens however if we have multiple hyperparameters?

1. We can search over all combinations of values for specified ranges of each parameter.
2. This is the standard method in choosing parameters for support vector machines (SVMs).
3. With SVMs it is generally limited to the case of only two hyperparameters.
4. Larger numbers quickly become infeasible.
Machine Learning Commandments

Thou shalt provide evidence before claiming that thy method is the best. The shalt take extra notice of this Commandment if thou considers thyself a True And Pure Bayesian.

Comparing classifiers

Imagine I have compared the Bloggs Classifier 2000 and the CleverCorp Discriminotron and found that:

1. Bloggs Classifier 2000 has estimated accuracy 0.981 on the test set.
2. CleverCorp Discriminotron has estimated accuracy 0.982 on the test set.

Can I claim that the CleverCorp Discriminotron is the better classifier?
Answer: NO! NO! NO! NO! NO! NO! NO! NO! NO! NO!!

Comparing classifiers

NO!!!!!!!

Note for next year: include photo of grumpy-looking cat.

Assessing a single classifier

From Mathematical Methods for Computer Science:
The Central Limit Theorem: If we have independent identically distributed (iid) random variables $X_1, X_2, \ldots, X_n$ with mean $E[X] = \mu$ and standard deviation $E[(X - \mu)^2] = \sigma^2$ then as $n \to \infty$

\[
\hat{X}_n = \frac{\mu - \mu}{\sigma/\sqrt{n}} \to N(0, 1)
\]

where

\[
\hat{X}_n = \frac{1}{n} \sum_{i=1}^{n} X_i.
\]
Assessing a single classifier

We have tables of values $z_p$ such that if $x \sim N(0, 1)$ then
$$\Pr(-z_p \leq x \leq z_p) > p.$$ Rearranging this using the equation from the previous slide we have that with probability $p$
$$
\mu \in \left[ \hat{X}_n \pm z_p \sqrt{\frac{\sigma^2}{n}} \right].
$$

We don’t know $\sigma^2$ but it can be estimated using
$$\sigma^2 \simeq \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \hat{X}_n)^2.$$ Alternatively, when $X$ takes only values 0 or 1
$$\sigma^2 = \mathbb{E}[(X - \mu)^2] = \mathbb{E}[X^2] - \mu^2 = \mu(1 - \mu) \simeq \hat{X}_n(1 - \hat{X}_n).$$

Assessing a single classifier

Typically we are interested in a 95% confidence interval, for which $z_p = 1.96$.
Thus, when $m > 30$ (so that the central limit theorem applies) we know that, with probability 0.95
$$\text{er}(h) = \hat{\text{er}}_s(h) \pm 1.96 \sqrt{\frac{\hat{\text{er}}_s(h)(1 - \hat{\text{er}}_s(h))}{m}}.$$ Example: I have 100 test examples and my classifier makes 18 errors. With probability 0.95 I know that
$$\text{er}(h) = 0.18 \pm 1.96 \sqrt{\frac{0.18(1 - 0.18)}{100}} = 0.18 \pm 0.075.$$ This should perhaps raise an alarm regarding our suggested comparison of classifiers above.

Assessing a single classifier

The actual probability of error for a classifier $h$ is
$$\text{er}(h) = \mathbb{E}[I[h(x) \neq y]]$$
and we are estimating $\text{er}(h)$ using the accuracy
$$\hat{\text{er}}_s(h) = \frac{1}{m} \sum_{i=1}^{m} I[h(x_i) \neq y_i]$$
for a test set $s$.
We can find a confidence interval for this estimate using precisely the derivation above, simply by noting that the $X_i$ are the random variables
$$X_i = I[h(x_i) \neq y_i].$$

There is an important distinction to be made here:

1. The mean of $X$ is $\mu$ and the variance of $X$ is $\sigma^2$.
2. We can also ask about the mean and variance of $\hat{X}_n$.
3. The mean of $\hat{X}_n$ is
$$\mathbb{E}[\hat{X}_n] = \mathbb{E}\left[\frac{1}{n} \sum_{i=1}^{n} X_i\right] = \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}[X_i] = \mu.$$ 4. It is left as an exercise to show that the variance of $\hat{X}_n$ is
$$\sigma_{\hat{X}_n}^2 = \frac{\sigma^2}{n}.$$
Comparing classifiers

We are using the values $z_p$ such that if $x \sim N(0,1)$ then
$$\Pr(-z_p \leq x \leq z_p) > p.$$ 

There is an alternative way to think about this.

1. Say we have a random variable $Y$ with variance $\sigma_Y^2$ and mean $\mu_Y$.
2. The random variable $Y - \mu_Y$ has variance $\sigma_Y^2$ and mean 0.
3. It is a straightforward exercise to show that dividing a random variable having variance $\sigma^2$ by $\sigma$ gives us a new random variable with variance 1.
4. Thus the random variable $\frac{Y - \mu_Y}{\sigma_Y}$ has mean 0 and variance 1.

So: with probability $p$

$$Y = \mu_Y \pm z_p \sigma_Y$$
$$\mu_Y = Y \pm z_p \sigma_Y.$$ 

Compare this with what we saw earlier. You need to be careful to keep track of whether you are considering the mean and variance of a single RV or a sum of RVs.

Also notice:

1. The two parts of the estimate $\hat{e}_{r_1}(h_1)$ and $\hat{e}_{r_2}(h_2)$ are each sums of random variables and we can apply the central limit theorem to each.
2. The variance of the estimate is the sum of the variances of $\hat{e}_{r_1}(h_1)$ and $\hat{e}_{r_2}(h_2)$.
3. Adding Gaussians gives another Gaussian.
4. We can calculate a confidence interval for our estimate.

With probability 0.95

$$d = \hat{d} \pm 1.96 \sqrt{\frac{\hat{e}_{r_1}(h_1)(1 - \hat{e}_{r_1}(h_1))}{m_1} + \frac{\hat{e}_{r_2}(h_2)(1 - \hat{e}_{r_2}(h_2))}{m_2}}.$$ 

In fact, if we are using a split into training set $s$ and test set $s'$ we can generally obtain $h_1$ and $h_2$ using $s$ and use the estimate

$$\hat{d} = \hat{e}_{r_1}(h_1) - \hat{e}_{r_2}(h_2).$$

Comparing classifiers—hypothesis testing

Now say I have classifiers $h_1$ (Bloggs Classificator 2000) and $h_2$ (CleverCorp Discriminotron) and I want to know something about the quantity

$$d = e(r(h_1)) - e(r(h_2)).$$

I estimate $d$ using

$$\hat{d} = \hat{e}_{r_1}(h_1) - \hat{e}_{r_2}(h_2)$$

where $s_1$ and $s_2$ are two independent test sets.

Notice:

1. The estimate of $d$ is a sum of random variables, and we can apply the central limit theorem.
2. The estimate is unbiased

$$E[\hat{e}_{r_1}(h_1) - \hat{e}_{r_2}(h_2)] = d.$$ 

This still doesn’t tell us directly about whether one classifier is better than another—whether $h_1$ is better than $h_2$.

What we actually want to know is whether

$$d = e(r(h_1)) - e(r(h_2)) > 0.$$ 

Say we’ve measured $\hat{D} = \hat{d}$. Then:

- Imagine the actual value of $\hat{d}$ is 0.
- Recall that the mean of $\hat{D}$ is $\hat{d}$.
- So larger measured values $\hat{d}$ are less likely, even though some random variation is inevitable.
- If it is highly unlikely that when $d = 0$ a measured value of $\hat{d}$ would be observed, then we can be confident that $d > 0$.
- Thus we are interested in

$$Pr(\hat{D} > \hat{d} + d).$$ 

This is known as a one-sided bound.
One-sided bounds

Given the two-sided bound
\[ \Pr(-z \leq x \leq z) = 1 - \epsilon \]
we actually need to know the one-sided bound
\[ \Pr(x \leq z_i) = 1 - \epsilon / 2 \]
Clearly, if our random variable is Gaussian then \( \Pr(x \leq z) = 1 - \epsilon / 2 \).

Comparing algorithms: paired t-tests

Recall, we subdivide s into \( n \) folds \( s^{(i)} \) each having \( m/n \) examples

\[
\begin{array}{c|c|c|c}
\text{w}^{(0)} & \text{w}^{(1)} & \text{w}^{(n)} \\
\hline
\end{array}
\]
and denote by \( s_{-i} \) the set obtained from \( s \) by removing \( s^{(i)} \). Then
\[
\frac{1}{n} \sum_{i=1}^{n} \hat{e}_{r^{(i)}}(L(s_{-i}))
\]
is the \( n \)-fold crossvalidation error estimate. Now we estimate \( d \) using
\[
\hat{d} = \frac{1}{n} \sum_{i=1}^{n} [\hat{e}_{r^{(i)}}(L_1(s_{-i})) - \hat{e}_{r^{(i)}}(L_2(s_{-i}))].
\]