Introduction to Neural Networks
U. Minn. Psy 5038

Linear discriminant, Bias/Variance & Model Selection, Pointers to support vector approaches

Initialize

■ **Read in Statistical Add-in packages:**

In[165]:=
```
Off[General::spell1];
<< Statistics`DescriptiveStatistics`
<< Statistics`DataManipulation`
<< Statistics`NormalDistribution`
<< Statistics`MultiDescriptiveStatistics`
<< Statistics`MultinormalDistribution`
```

Review Discriminant functions

Let's review earlier material on discriminant functions. Perceptron learning is an example of nonparametric statistical learning, because it doesn't require knowledge of the underlying probability distributions generating the data (such distributions are characterized by a relatively small number of "parameters", such as the mean and variance of a Gaussian distribution). Of course, how well it does will depend on the generative structure of the data. Much of the material below is covered in Duda and Hart (1978).

Linear discriminant functions: Two category case

A discriminant function, g(x) divides input space into two category regions depending on whether g(x)>0 or g(x)<0. (We've switched notation, x=**f**). The linear case corresponds to the simple perceptron unit we studied earlier:

$$g(\mathbf{x}) = \mathbf{w}.\mathbf{x} + w_0 \qquad (1)$$

where w is the weight vector and w_0 is the threshold (sometimes called bias, although this "bias" has nothing to do with statistical "bias").

Discriminant functions can be generalized, for example to quadratic decision surfaces:

$$g(\mathbf{x}) = w_0 + \sum_{i=1} w_i x_i + \sum_{i=1}\sum_{j=1} w_{ij} x_i x_j \qquad (2)$$

We've seen how g(x)=0 defines a decision surface which in the linear case is a hyperplane. Suppose x_1 and x_2 are points sitting on the hyperplane, then their difference is a vector lying in the hyperplane

$$\begin{aligned} \mathbf{w}.\mathbf{x}_1 + w_0 &= \mathbf{w}.\mathbf{x}_2 + w_0 \\ \mathbf{w}.(\mathbf{x}_1 - \mathbf{x}_2) &= 0 \end{aligned} \qquad (3)$$

so the weight vector w is normal to any vector lying in the hyperplane. Thus w determines how the plane is oriented. The normal vector w points into the region for which g(x)>0, and -w points into the region for which g(x)<0.

Let x be a point on the hyperplane. If we project x onto the normalized weight vector x.w/|w|, we have the normal distance of the hyperplane from the origin equal to:

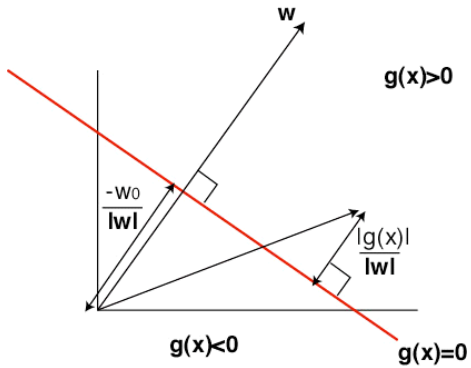
$$\mathbf{w}.\mathbf{x} / |\mathbf{w}| = -w_0 / |\mathbf{w}| \qquad (4)$$

Thus, the threshold determines the position of the hyperplane.

One can also show that the normal distance of x to the hyperplane is given by:

$$g(\mathbf{x}) / |\mathbf{w}| \qquad (5)$$

So we've seen that: 1) disriminant function divides the input space by a hyperplane decision surface; 2) The orientation of the surface is determined by the weight vector w; 3) the location is determined by the threshold w_0 ; 4) the discriminant function gives a measure of how far in input vector is from the hyperplane.

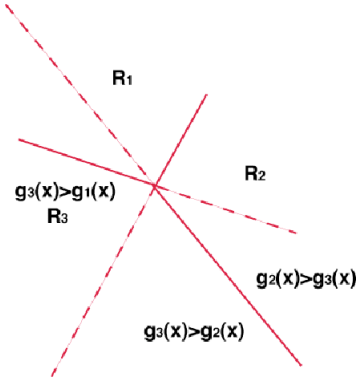


Multiple classes

Suppose there are c classes. There are a number of ways to define multiple class discriminant rules. One way that avoids undefined regions is:

$$g_i(x) = w_i.x + w_{i0},\ i = 1, \ldots, c \qquad (6)$$

$$\text{Assign x to the ith class if : } g_i(x) > g_j(x) \text{ for all } j \neq i. \qquad (7)$$



It can be shown that this classifier partitions the input space into simply connected convex regions. This means that if you connect any two feature vectors belonging to the same class by a line, all points on the line are in the same class. Thus this linear classifier won't be able to handle problems for which there are disconnected clusters of features that all belong to the same class. Also, from a probabilistic perspective, if the underlying generative probability model for a given class has multiple modes, this linear classifier won't do a good job either.

Task-dependent Dimensionality reduction

Fisher's linear "discriminant"

The idea is that the original input space may be impractically huge, but if we can find a subspace (hyperplane) that preserves the distinctions between categories as well as possible, we can make our decisions in smaller space. We will derive the Fisher linear "discriminant".

This is closely related to the psychology idea of finding "distinctive" features. E.g. consider bird identification. If I want to discriminate cardinals from other birds in my backyard, I can make use of the fact that (male) cardinals may be the only birds that are red. So even tho' the image of a bird can have lots of dimensions, if I project the image on to the "red" axis, I can do fairly well with just one number. How about male vs. female human faces?

■ Generative model: two nearby gaussian classes

Define two bivariate base distributions

In[572]:=
```
(ar = {{1, 0.99}, {0.99, 1}};
ndista = MultinormalDistribution[{0, -1}, ar];)
(br = {{1, .9}, {.9, 2}};
ndistb = MultinormalDistribution[{0, 1}, br];)
```

Find the expression for the probability distribution function of ndista

Use Mean[ ] and CovarianceMatrix[ndista] to verify the population mean and the covariance matrix of ndistb

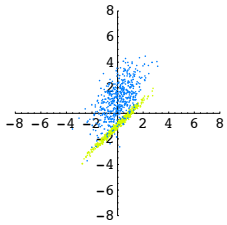
In[574]:=
```
CovarianceMatrix[ndista]
```

Out[574]=
$$\begin{pmatrix} 1 & 0.99 \\ 0.99 & 1 \end{pmatrix}$$

Try different covariant matrices. Should they be symmetric? Constraints on the determinant of ar, br?

Make a contour plot of the PDF ndista

In[611]:=
```
nsamples = 500;
a = Table[Random[ndista], {nsamples}];
ga = ListPlot[a, PlotRange → {{-8, 8}, {-8, 8}},
   AspectRatio → 1, PlotStyle → Hue[0.2], DisplayFunction → Identity];
b = Table[Random[ndistb], {nsamples}];
gb = ListPlot[b, PlotRange → {{-8, 8}, {-8, 8}},
   AspectRatio → 1, PlotStyle → Hue[0.6], DisplayFunction → Identity];
Show[ga, gb, DisplayFunction → $DisplayFunction];
```



Use Mean[ ] to find the *sample* mean of b. Whats is the sample *covariance* of b?

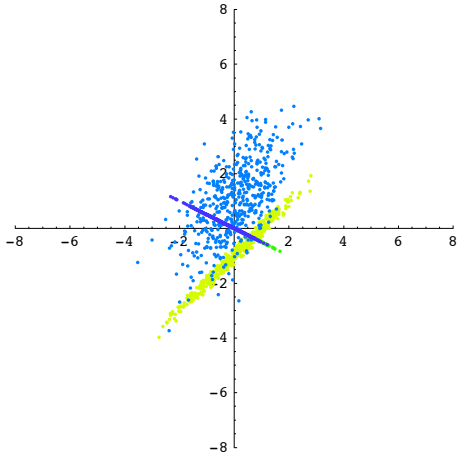
■ **Try out different projections of the data by varying the slope (m) of the discriminant line**

In[617]:=
```
m = -1 / 2;
wnvec = {1, m} / Sqrt[1 + m^2];
```



```
{{x, y}}.{n1, n2}
Map[#1 * {n1, n2} &, {{x, y}}.{n1, n2}]
```

In[646]:=
```
aproj = Map[#1 * wnvec &, a.wnvec];
gaproj =
  ListPlot[aproj, AspectRatio → 1, PlotStyle → Hue[0.3], DisplayFunction → Identity];

bproj = Map[#1 * wnvec &, b.wnvec];
gbproj =
  ListPlot[bproj, AspectRatio → 1, PlotStyle → Hue[0.7], DisplayFunction → Identity];

Show[ga, gb, gaproj, gbproj,
  DisplayFunction → $DisplayFunction, AspectRatio → Automatic];
```



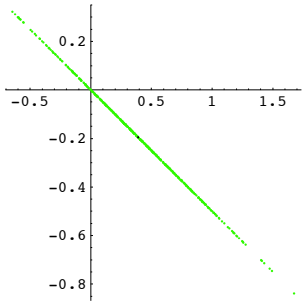
By trial and error, find a value of m that separates the classes well along the projection line

Calculate the "signal-to-noise" ratio along the projection line: difference between the means divided by the square root of the product of the standard deviations along the line

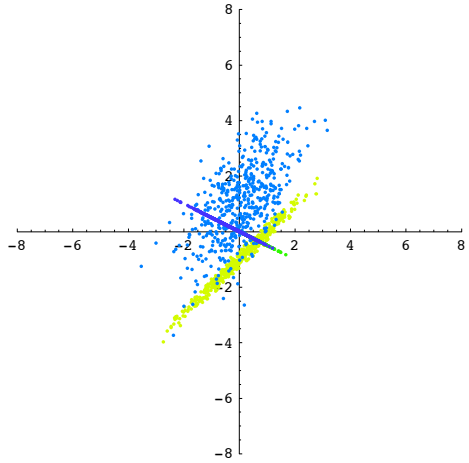
In[652]:=
```
Mean[aproj]
```

Out[652]=
```
{0.391426, -0.195713}
```

In[662]:=
```
Show[gaproj, gm1, DisplayFunction → $DisplayFunction]
```



In[642]:=
```
Show[ga, gb, gaproj, gbproj,
  DisplayFunction → $DisplayFunction, AspectRatio → Automatic];
```



■ Theory for simple 2-class case

(see Duda and Hart for general case)

A measure of the separation between the projections is the difference between the means:

$$|\, w \cdot (m_a - m_b) \,|$$

and

$$m_a = \frac{1}{N} \sum_{i=1}^{N} x, \quad \text{summed over the N x's from class a} \qquad (8)$$

$$m_b = \frac{1}{M} \sum x, \quad \text{summed over the M x's from class b}$$

where w (**wnvec**) is the unknown unit vector along the discriminant line .

In our case above, the vector difference between the means is:

In[735]:=
```
Mean[a] - Mean[b]
```

Out[735]= {−0.0923584, −2.07925}

and the difference between the means projected onto a discriminant line is:

In[734]:=
```
wnvec.(Mean[a] - Mean[b])
```

Out[734]= 0.847261

To improve separation, we can't just scale w, because the noise scales too.

We'd like the difference between the means to be large relative to the variation for each class. We can define a measure of the scatter for the projected samples in say class a (a==1), by:

$$\sum_{y \in \text{class a}} (y - \tilde{m}_a)^2 \tag{9}$$

where $\tilde{m}_a$ is the sample mean of the points from class a projected onto discriminant line and y=$\mathbf{w.x}$

Or in terms of the Mathematic example:

```
Apply[Plus, aproj - wnvec.Mean[a]];
```

The total scatter S is defined by the sum of the scatters for both classes (a and b).

$$S = \sum_{y \in \text{class a}} (y - \tilde{m}_a)^2 + \sum_{y \in \text{class b}} (y - \tilde{m}_b)^2$$

If we divide the above number by the total number of points, we have an estimate of the variance of the combined data along the projected axis.

We now have the basic ingredients behind basic intuition for the Fisher linear discriminant. We'd like to find that **w** for which J:

$$J(w) = \frac{|\tilde{m}_a - \tilde{m}_b|^2}{S} = \frac{|\mathbf{w}.(m_a - m_b)|^2}{S}$$

is biggest. We want to maximize the difference between the projected class means, while minimizing the dispersion of the data on the projected line.

One can show that S = $\mathbf{w^T.S_W.w}$, where

S_W is measure of within-class variation called the *within-class scatter matrix:*

$$S_W = \sum_{i=1}^{2} \sum_{x \in \text{class } i} (x - m_i)(x - m_i)^T \tag{10}$$

For the numerator, a measure of between class variation is the *between-class scatter matrix*:

$$S_B = (m_1 - m_2).(m_1 - m_2)^T \tag{11}$$

and the difference between the projected means can be show to be:

$|\tilde{m}_a - \tilde{m}_b|^2 = w^T.S_B.w$

Find **w** (corresponding to slope) to maximize the criterion function

$$J(w) = \frac{w^T.S_B.w}{w^T.S_W.w} \tag{12}$$

Answer:

$$w = S_W^{-1}.(m_a - m_b) \tag{13}$$

■ Demo: Finding Fisher's linear discriminant

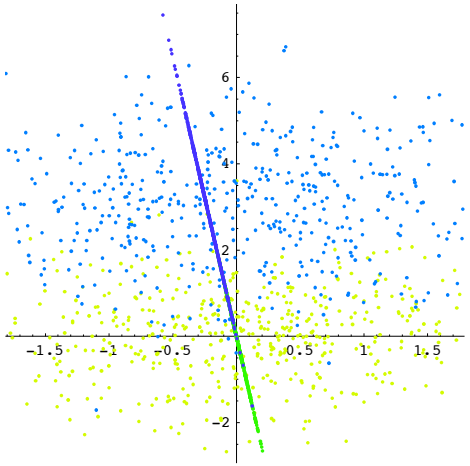
In[189]:=
```
normalize[x_] := x / Sqrt[x.x];
```

In[190]:=
```
ma = Mean[a];
mb = Mean[b];
```

In[192]:=
```
Sa = Sum[Outer[Times, a[[i]] - ma, a[[i]] - ma], {i, 1, nsamples}];
Sb = Sum[Outer[Times, b[[i]] - mb, b[[i]] - mb], {i, 1, nsamples}];
Sw = Sa + Sb;
wldf = normalize[Inverse[Sw].(ma - mb)]
```

Out[195]= {0.0769866, −0.997032}

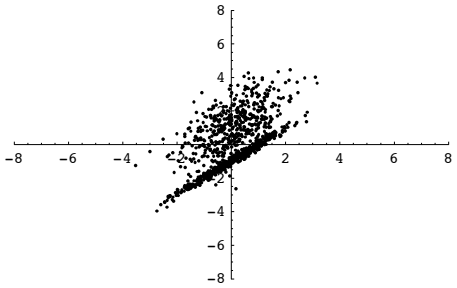
In[196]:=
```
aproj = Map[#1 * wldf &, a.wldf];
gaproj =
  ListPlot[aproj, AspectRatio → 1, PlotStyle → Hue[0.3], DisplayFunction → Identity];

bproj = Map[#1 * wldf &, b.wldf];
gbproj =
  ListPlot[bproj, AspectRatio → 1, PlotStyle → Hue[0.7], DisplayFunction → Identity];

Show[ga, gb, gaproj, gbproj, DisplayFunction → $DisplayFunction];
```



We started off with a 2-dimensional input problem and turned it into a 1-D problem. For the n-dimensional case, see Duda and Hart.

■ Compare with the principal component axes

In[624]:=
```
c = Join[a, b];
ListPlot[c, PlotRange → {{-8, 8}, {-8, 8}}];
```

In[626]:=
```
g1 = ListPlot[c, PlotRange → {{-8, 8}, {-8, 8}},
   AspectRatio → 1, DisplayFunction → Identity];
```

In[664]:=
```
auto = CovarianceMatrix[c]
eigvalues = Eigenvalues[auto]
MatrixForm[eigauto = Eigenvectors[auto]]
```

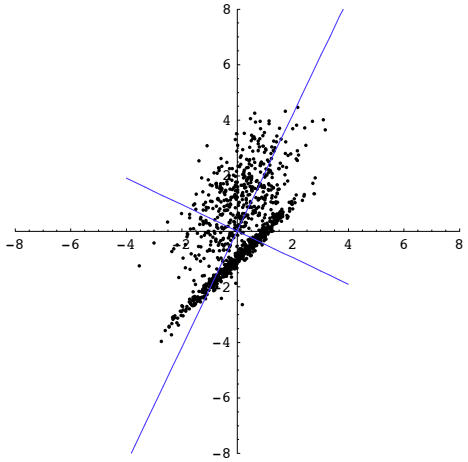
Out[664]=
$$\begin{pmatrix} 0.970786 & 0.928494 \\ 0.928494 & 2.47561 \end{pmatrix}$$

Out[665]=
$$\{2.91828, 0.528115\}$$

Out[666]//MatrixForm=
$$\begin{pmatrix} 0.430355 & 0.90266 \\ 0.90266 & -0.430355 \end{pmatrix}$$

In[633]:=
```
gPCA = Plot[{eigauto[[1,2]]/eigauto[[1,1]] x,
    eigauto[[2,2]]/eigauto[[2,1]] x},
        {x,-4,4}, AspectRatio->1,
        DisplayFunction->Identity,
        PlotStyle->{RGBColor[.2,0,1]}];
```

In[634]:=
```
Show[g1,gPCA,DisplayFunction->$DisplayFunction];
```



How does the principal component (biggest variance) compare with the Fisher discriminant line?

Model selection & The bias/variance dilemma

Above we assumed linear separability. If the data are not linearly separable, then we resort to more complicated decision surfaces.That is what originally led us to non-linear multi-layer networks with hidden units. At first thought, one might think that we could just add lots of hidden units and solve any input-output problem. But there is a problem--the bias/variance dilemma.

Consider the regression problem, fitting data that may be a complex function of the input.

The problem in general is how to choose the function that both remembers the relationship between **x** and **y**, and generalizes with new values of **x**. The function has to be parameterized in some way that allows easy computation. What form should the function take? At first one might think that it should be as general as possible to allow all kinds of maps.

For example, if one is fitting a curve, you might wish to use a very high-order polynomial, or a back-prop network with lots of hidden units. There is a draw back to the flexibility afforded by extra degrees of freedom in fitting the data. We can get drastically different fits for different sets of data that are randomly drawn from the same underlying process. The fact that we get different fit parameters (e.g. slope of a regression line) each time means that although we may exactly fit the data each time, we introduce variation between the average fit (over all data sets) and the fit for a single data set. We could get around this problem with a huge amount of data, but the problem is that the amount of required data can grow exponentially with the order of the fit--an example of the so-called "curse of dimensionality".

On the other hand, if the function is restrictive, (e.g. straight lines through the origin), then we will get similar fits for different data sets, because all we have to adjust is one parameter--the slope. The problem here, is that the fit is only good if the underlying process is in fact a straight line through the origin. If it isn't a straight line for instance, there will be a fixed error or **bias** that will never go away, no matter how much data we collect. Statisticians refer to this problem as the *bias/variance* dilemma.

To sum up, lots of parameter flexibility (or lots of hidden units) has the benefit of fitting anything, but at the cost of sensitivity to variability in the data set--there is *variance* introduced by the fits found over multiple training sets (e.g. of a small fixed size).

A fit with very few parameters is not as sensitive to the inevitable variability in the training set, but can give large constant errors or *bias* if the data do not match the underyling model.

There is no general way of getting around this problem, and neural networks are no exception. We generalized linear regression to non-linear fits using error back-propagation. Because back-propagation models can have lots of hidden layers with many units and weights, they form a class of very flexible approximators and can fit almost any function. But these models can show high variability in their fits from one data set to the next, even when the data comes from the same underlying process. Lots of hidden units can mean low bias, but at a high cost in variance.

Few or no hidden units, such as the linear associator is very restrictive, would have smaller variance, but could show high bias.

One has to study the problem one is trying to model (independently of the neural network parameters if possible) and choose the appropriate network model to do the fit. Because the straight line, or in higher dimensions, a linear model is, in a mathematical , the "simplest", it does make good sense to try it first. Later on, one can complicate matters if it turns out that

the data one is trying to fit is non-linear.

For a formal definition of the bias/variance trade-off, see pdf notes in class outline.

The bias/variance dilemma has led to techniques for model selection. This is the problem of how to pick the "simplest" model that is still consistent with the data. Much research in data modeling and pattern learning is devoted to the problem of how to achieve good generalization after learning from finite example sets. Good models combine some faithfulness to the data seen, while minimizing the complexity of the model used to fit the data. There are Bayesian approaches (e.g. MacKay, 1995) and related minimum description length techniques (cf. Bishop). But another rather different approach is support vector machines.

Beyond linear separation, beyond sigmoidal kernels

Density estimation vs. classification

Map the data (through some non-linear mapping, e.g. polynomial) to a higher-dimensional space to find the optimal hyper-plane separating the data. But what is "optimal". Require good generalization, small VC dimension. Construct the hyper-plane on a small number of support vectors, then the generalization ability will be high.

Main source: Vapnik (1995)

Demo links:

http://svm.dcs.rhbnc.ac.uk/pagesnew/GPat.shtml

References

Duda, R. O., & Hart, P. E. (1973). Pattern classification and scene analysis . New York.: John Wiley & Sons.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification* (2nd ed.). New York: Wiley.

(Amazon.com)

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. Neural Computation, 4(1), 1-58.

MacKay, D. J. C. (1992). Bayesian interpolation. *Neural Computation, 4*(3), 415-447.

Ripley, B. D. (1996). *Pattern Recognition and Neural Networks*. Cambridge, UK: Cambridge University Press.

Vapnik, V. N. (1995). *The nature of statistical learning*. New York: Springer-Verlag.

http://neuron.eng.wayne.edu/software.html