Introduction to Neural Networks

Unifying neural network computations using Bayesian decision theory

Initialize:

```
<< "MultivariateStatistics`"
```

```
Off[General::spell1];
```

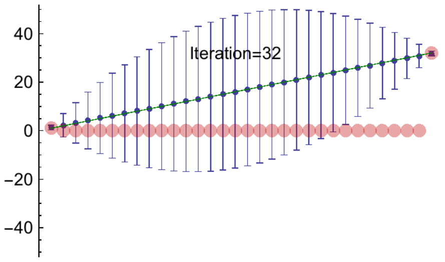
```
SetOptions[ListDensityPlot, ImageSize → Small];
SetOptions[DensityPlot, ImageSize → Small];
SetOptions[ContourPlot, ImageSize → Small];
```

Introduction

Belief propagation

We also computed marginal distributions in the previous lecture's example of belief propagation. Belief propagation is a general technique which can be used to compute marginal distributions and to do MAP estimation. We looked at a very specific example. In particular, our goal was to calculate the marginals:

p(y_i | y_1^*, ...), where y_i represented the depth of an interpolated surface, and $\{y_j^*\}$ represent the measurements of the depth. In our case, the distributions were gaussian. Thus each marginal told us the most probable depth (the mean) and the degree of uncertainty (standard deviation). In our example we had only two data points, one at each end. The rest of the depths were interpolated based on a prior smoothness constraint that encouraged nearby depths to be the same. The result was a "compromise" between what the data "said" and what the prior "thought" answers should usually look like:

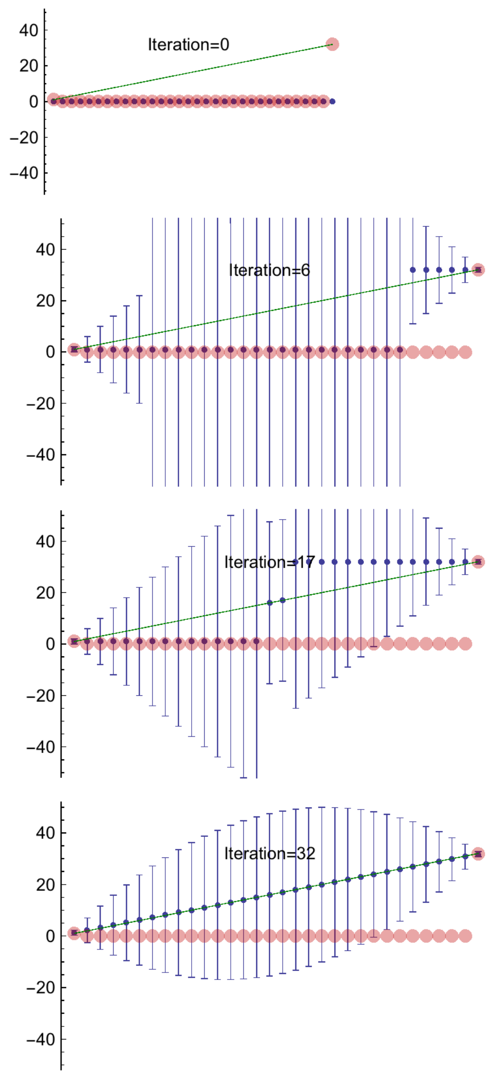


This was the final result. To get there, the challenge was to make our best guess of the surface depth given that:

1) We didn't have data everywhere. The stereogram was "sparse";

2) When we did have depth measurements, they were noisy.

We arrived at the result using belief propagation, which is an iterative method where one updates marginal distributions at each node by receiving and passing messages between neighboring nodes. If there are no loops in the graph, belief propagation provides an exact solution.

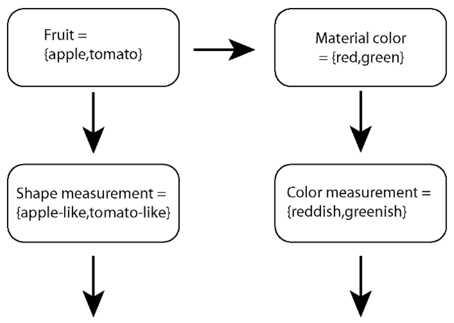


Previously...

...a brief review of what is needed today

Natural patterns are complex, and in general it is difficult and often impractical to build a detailed quantitative generative model. But natural inputs, such as sounds and images, do have regularities, and we can get insight into the problem by considering how various factors might produce them.

One way to begin simplifying the problem is to note that not all variables have a direct influence on each other. So draw a graph in which lines only connect variables that influence each other. We use directed graphs to represent conditional probabilities.


Fruit = {apple,tomato}
Material color = {red,green}
Shape measurement = {apple-like,tomato-like}
Color measurement = {reddish,greenish}

The the graph specifies how to decompose the joint probability:
p[F, C, Is, Ic] = p[Ic | C] p[C | F] p[Is | F] p[F]

Basic rules: Condition on what is known, integrate out what you don't care about.

Condition on what is known:

Given a state of the world S, and inputs I, the "universe" of possibilities is:

$p(S, I)$

If we know I (i.e. the visual system has measured some image feature I), the joint can be turned into a conditional (posterior):

$p(S \mid I) = p(S, I) / p(I)$

Integrate out what we don't care about

Let $S_{primary}$ be the variables we care about, $S_{secondary}$ the ones we don't (also called "noise" or confounding variables or nuisance variables or generic variables).

We don't care to estimate the noise (or other generic, nuisance, or secondary variables):

$$p(S_{primary} \mid I) = \sum_{S_{noise}} p(S_{primary}, S_{secondary} \mid I),$$

$$\text{or if continuous} = \int_{S_{secondary}} p(S_{primary}, S_{secondary} \mid I)\, dS_{secondary}$$

This is the "integrating out" or "marginalization" step. Decisions are then based on the marginals.

Summary of the fruit & color classification example

Pick most probable fruit AND color--

We want to maximize the probability of getting the right fruit and the right color. Both fruit and color are primary variables.

--Answer "red tomato"

Pick most probable color

We want to maximize the probability of getting the right color, and don't care about which fruit it is. Color is primary, fruit type is secondary.

So sum over (marginalize) the fruit type variable to get the marginal p(color | measurements)

--Answer "red"

Pick most probable fruit

We want to maximize the probability of getting the right fruit, and don’t care about what color it is. Fruit type is primary, and color is secondary.

So sum over (marginalize) the material color variable to get the marginal p(fruit | measurements)

--Answer "apple"

Today

Generalize the notion of what it means to “care about” a variable. Leads us to Bayesian decision theory.

Showing how Bayesian decision theory relates to neural network and machine learning

Set neural network supervised learning in the context of various statistical/machine learning methods

Bayesian Decision Theory: Utility

How to generalize optimal inference to include different degrees of importance in task requirements?

Bayes Decision theory, loss, and risk

We'd now like to generalize the idea of "integrating out" unwanted variables to allow us to put weights on how important a variable is for a task.

Consider a simple discrete decision task in which there is a measurement and one has to decide whether to accept or reject an hypothesis.This is the classic problem of signal detection.

Imagine a noisy image and you have to decide whether it is a human face or not. There are two ways of being correct and two ways of being wrong.

If there really is a face there, and you decide “yes”--that is called a “hit” (or true positive).

If there really is a face there, and you decide “no”, that is a miss (also called a false negative).

If there really is not a face there, and you decide “yes there is a face”--that is called a “false alarm”

(also called a false positive).

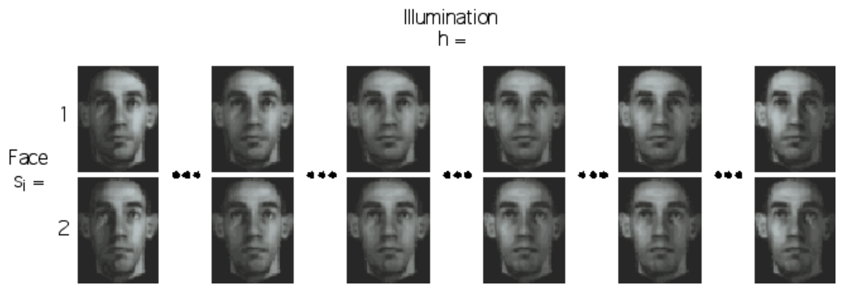
If there really is not a face there, and you decide “no”, that is a correct rejection (or true negative).

But sometimes getting the benefits of getting the right answer has to be considered in light of the costs of the different types of errors. The costs of certain kinds of errors (e.g. a high cost to false alarms) can affect the decision criterion.

For example, a health professional might say that since stress EKG's have about a 30% false alarm rate, follow-up tests aren’t worth doing. The cost of a false alarm is high in dollars, with the resulting follow-ups, angiograms, etc.. And there is some increased risk to the patient of extra unnecessary tests. But, of course, false alarm rate isn't the whole story, and one should ask what the hit rate (or alternatively the miss rate) is. If the miss rate is about 10%, from the patient's point of view, the cost of a miss is very high, one's life. So a patient's goal would not be to minimize error rate per se (i.e. probability of a mis-diagnosis, which in effect gives equal weights to both kinds of errors), but rather to minimize a measure of subjective cost that puts a very high cost on a miss, and low cost on a false alarm.

Bayes Decision theory, loss, and risk in perception

Although decision theory in vision has traditionally been applied to analogous trade-offs that are more cognitive than perceptual, the concept of utility is relevant in many aspects of perception. Perception has implicit, unconscious trade-offs in the kinds of errors that are made.



For example, image intensities provide the data that can be used to estimate an object's shape and/or estimate the direction of illumination. Accurate object identification often depends crucially on an objec-t's shape, and the illumination is a confounding (secondary or nuisance) variable. This suggests that visual recognition should put a high cost to errors in shape perception, and lower costs on errors in illumination direction estimation.

So the process of perceptual inference depends on task.

The effect of marginalization in the fruit example illustrated task-dependence. Now we show how marginalization can be generalized through decision theory to model other kinds of goals than error minimization (MAP) in task-dependence.

Why do people often report seeing faces in clouds, tree bark, shower curtains, on mars, in pancakes?

Perhaps it is because the social cost of a false alarm is low compared to the cost of a miss, which is high. If you see a face in a tree trunk, you might be alarmed, but as the saying goes, “better safe than sorry”.

Bayes Decision theory provides tools to model performance as a function of utility.

Some terminology. The terms *state*, *hypothesis*, *signal* are essentially the same--to represent the random variable (which could be vector or list) indicating the state of the world--the "state space". We often assume that the decision, d, of the observer maps directly to state space, d->s. E.g. I estimate that an object is 10 feet away.

We now clearly distinguish the decision space from the state or hypothesis space, and introduce the idea of a loss L(d , s), which is a number representing the cost of making the decision d, when the actual state is s. Loss could be between like variables, such as the cost of estimating a distance of 10 feet when it is actually 11 feet away. But it could also represent the cost of a decision or action that relies on the true distance. E.g. the cost of choosing an initial velocity (call it d) of a bean bag intended to hit a target 11 feet away.

As we’ve seen in sensory processing, often we can't directly measure the true value s, and we can only infer it from observations, e.g. a sensory or image measurement x, through a posterior p(s|x).

Thus, given an observation x, we define a risk function that represents the *average loss* over signal states s:

$$R(d;x) = \sum_s L(d,s)\, p(s \mid x)$$

This suggests a decision rule: $\alpha(x)=\underset{d}{\text{argmin}}\, R(d;x)$ for that particular x.

But there could be many x’s and not all x are equally likely. This suggests a decision rule that minimizes the expected risk averaged over all observations:

$$R(\alpha) = \sum_x R(d;x)\, p(x)$$

Loss functions determine standard inference choices

We won't show them all here, but with suitable choices of likelihood, prior, and loss functions, we can derive standard estimation procedures (maximum likelihood, MAP, estimation of the mean) that minimize risk as special cases.

For the MAP estimator,

$$R(d;x) = \sum_s L(d,s)\, p(s \mid x) = \sum_s (1-\delta_{d,s})\, p(s \mid x) = 1 - p(d \mid x)$$

where as we've seen before, $\delta_{d,s}$is the discrete analog to the Dirac delta function--it is zero if d≠s, and one if d=s. (See KroneckerDelta[ ])

Thus minimizing risk with the loss function L = $(1-\delta_{d,s})$ is equivalent to maximizing the posterior, p(d | x). Choose d that maximizes the posterior effectively penalizes all errors equally.

What about marginalization? You can see from the definition of the risk function, that this corresponds to a uniform loss:

$$L = -1.$$

We don't care how bad (or good) the values of the marginalized variables are, so we given all combinations of d and s the same constant negative loss value L(d,s) = -1. So for L(d2,s2) = -1, minimizing risk is quantitatively equivalent to maximizing the marginal p(p(s1 | x)):

$$R(s1;x) = \sum_{s2} L(d2,s2)\, p(s1,s2 \mid x)$$

So for our face recognition example, a really huge error in illumination direction has the same cost as getting it right.

Back again to the fruit color example. With the identical conditional probablistic structure (e.g. graph) and probabilities, can one ideal decision maker decide "red tomato" and another "apple"? Optimal classification of the fruit identity required marginalizing over fruit color--i.e. effectively treating fruit color identification errors as equally costly...even tho', doing MAP after marginalization effectively means we are not explicitly identifying color.

Show that quadratic loss, L(d,s) = (d-s)^2, is equivalent to estimating the mean

---

Note on terminology: Is the glass half full or half empty? Above we described utility theory for pessimists. You may also see terminology favored by optimists, where "utility" or "gain" = - loss, and "expected utility" = - risk.

Slant estimation example

This section takes a toy version of "real life" problem, and derives a quantitative prediction of ideal behavior.

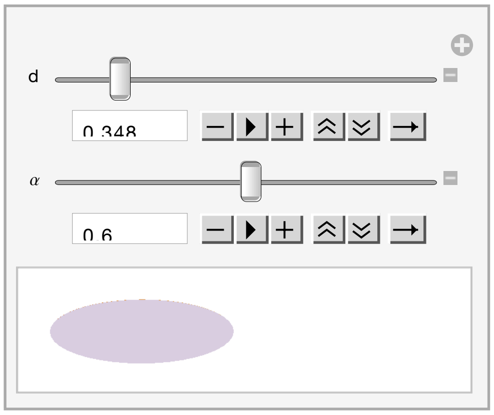
Estimation

Imagine the top of a coffee mug. It typically has a circular cross-section. However, due to projection, the image on your retina is more like an ellipse from most viewpoints. Now imagine it is a "designer coffee mug" which has an elliptical cross-section.

How could you guess the true, i.e. physical 3D shape, from measurements made in the projected image? The "aspect" slider below changes the ratio of the major to minor axes of the coffee mug. The "y" variable changes the slant of your viewpoint. These two causes determine an image measurement x--the height of the projected ellipse in the image (See "Slant" example below).

In[1]:=
```
Manipulate[Graphics3D[
  {EdgeForm[], Scale[Cylinder[{{-.0, -.01, -.0}, {.0, .01, .0}}, 1 / 2], {1, 1, d}]},
  Boxed → False, ImageSize → Tiny, ViewCenter → {0, 0, 0},
  ViewPoint → {0, 10, α}], {{d, 1.0}, .1, 2}, {α, -20, 20}]
```

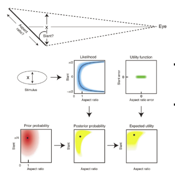
Out[1]=

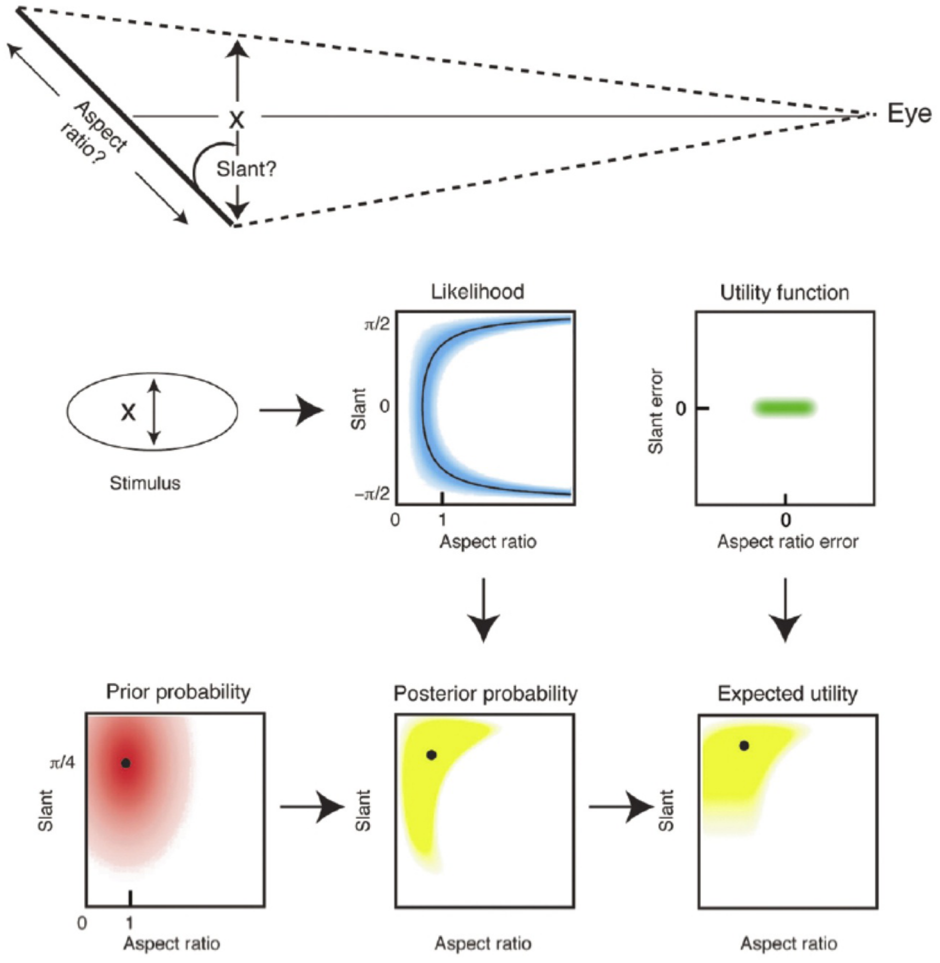


Let x be the data measurement, i.e. the "stimulus". It could be measured by height in the projected image. In our example below, we'll assume that x = 1/2.

We approximate the generative model as: x ≃ d Cos[slant] + noise, where d is the physical height of the disk in 3D, and slant is the inclination relative to the viewer. This approximation is reasonable when the disk size is small relative to the viewing distance. We make the simplifying assumption that the width is 1, so then **d** = **aspect ratio**.

We’d like to estimate the aspect ratio and the slant. But we have one measurement and two unknowns.

Image[]



From: Geisler, W. S., & Kersten, D. (2002). Illusions, perception and Bayes. Nat Neurosci, 5(6), 508-510.

Introduction

Consider the above figure.

Bayesian ideal observers for tasks involving the perception of objects or events that differ along two physical dimensions, such as aspect ratio and slant, size and distance, or speed and direction of motion. When a stimulus is received, the ideal observer computes the likelihood of receiving that stimulus for each possible pair of dimension values (that is, for each possible interpretation). It then multiplies this likelihood distribution by the prior probability distribution for each pair of values to obtain the posterior probability distribution—the probability of each possible pair of values given the stimulus. Finally, the posterior probability distribution is convolved with a utility function, representing the costs and benefits of different levels of perceptual accuracy, to obtain the expected utility associated with each possible interpretation. The ideal observer picks the interpretation that maximizes the expected utility. (Black dots and curves indicate the maxima in each of the plots.)

As a tutorial example, the figure was constructed with a specific task in mind; namely, determining the aspect ratio (**d**) of the physical object, and slant (***α***) of a tilted ellipse. The data is a measurement *(***x**) of the *aspect ratio of the image on the retina*. The black curve in the likelihood plot shows the ridge of maximum likelihood corresponding to the combinations of slant and aspect ratio that are exactly consistent with x=0.5; the other non-zero likelihoods occur because of noise in the image and in the measurement of x. The prior probability distribution corresponds to the assumption that surface patches tend to be slanted away at the top and have aspect ratios closer to 1.0.

The asymmetric utility function corresponds to the assumption that it is more important to have an accurate estimate of slant than aspect ratio.

Bayesian decision theory and learning

Bayesian inference seems completely different from the type of neural network feedforward models required to recognize objects. Let's look at how the Bayesian approach relates to alternative models based on neural networks, radial basis functions, or other machine learning methods? Understanding this relationship will also provide a justification for Bayes rule and the intuitions behind it.

Let a decision rule to estimate an output S be: $S^* = \alpha$(I) and let the loss function (or negative utility) be L(α(I),S). As discussed above, the loss function is the penalty for making the decision α(I) when the true state is S (e.g., a fixed penalty for a misclassification).

Suppose we have a set of examples $\{I_i, S_i$: i =1,...,N}, then the empirical risk (Vapnik 1998) of the rule α(I) is defined to be:

$$R_{emp}(\alpha) = (1/N)\sum_{i=1}^{N} L(\alpha(I_i), S_i) \tag{1}$$

For example, empirical risk could be the proportions of misclassifications.

The best decision rule $\alpha^*(.)$ is selected to minimize $R_{\text{emp}}(\alpha)$. For example, the decision rule is chosen to minimize the number of misclassifications. Neural networks and machine learning models select rules to minimize various forms of $R_{\text{emp}}(\alpha)$.

Now suppose that the samples $\{S_i , I_i\}$ come from a distribution p(S, I) over the set of training pairs. Then, if we have a sufficient number of samples, we can replace the empirical risk by the true risk:

$$R(\alpha) = \sum_I \sum_S L(\alpha(I), S) p(S, I) \tag{2}$$

Note this is just working backwards from what it means to take an average.

Minimizing R(α) leads to a decision rule that depends on the posterior distribution p(S|I) obtained by Bayes rule
p(S|I) = p(I|S)p(S)/p(I). To see this, we rewrite Equation 2 as

$$R(\alpha) = \sum_I p(I)\{\sum_S p(S|I)L(\alpha(I), S)\}$$

where we have expressed p(S, I) = p(S|I)p(I). So the best decision α(I) for a specific image I is given by

$$\alpha^*(I) = \arg\min_\alpha \sum_S p(S|I)L(\alpha(I), S)$$

and depends on the posterior distribution p(S|I). Hence, Bayes arises naturally when you start from the risk function specified by Equation 2.
There are two take-home messages:

1) the Bayes posterior p(S|I) follows logically from trying to minimize the number of misclassifications in the empirical risk (provided there are a sufficient number of samples).

2) it is possible to have an algorithm, or a network, that computes α(.) and minimizes the Bayes risk but that does not explicitly represent the probability distributions p(I|S) and p(S).
(See

Graphical model for decision theory: A summary

This section shows the common structure shared by three types of inference:
detection (“is the signal there or not?”), classification (“which signal is it?”), and estimation (“what is the quantity?”).

- Detection: a = s', s’ ∊ $\{s_1$, not $s'_1\}$
- Classification: a = s' ∊ $\{s_1, s_2, s_3, s_4 ...\}$
- Estimation: a = s', where s' takes on continuous values

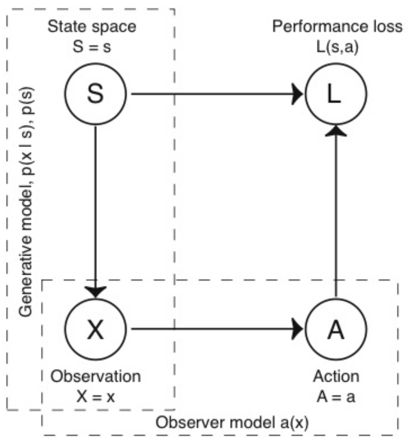
Decisions can be right or wrong regarding a discrete hypothesis (detection, classification), or have some metric distance from an hypothesis along a continuous dimensions (estimation). Each decision or estimation has an associated loss function. There is a common graphical structure to each type of inference.

In the diagram below, we replace the decision variable **d**, by a more general term **a** for "action".

The observer model might refer to the input/output model of a human, animal, neural population, or neuron. Or it can refer to and ideal observer or ideal agent that minimizes the average loss. An ideal agent is called a “normative” model of a process or behavior...i.e. given a description of the problem to be solved, what is best solution? Then one asks how an human, animal, neural population, or neuron compares.

Decisions, tasks, and actions

The variables in decision theory have a graphical structure (see Kersten and Mamassian, 2009) which determines how to factorize the joint distribution over them, and thus determines a common starting point for deriving ideal observers--i.e. decision makers that minimize risk. The graph below formally summarizes an observer or "agent".



How is utility learned and represented by the brain?

Natural loss functions may be "hard-wired", embedded in the architecture. But also adaptive changes through learning. The role of reward.

Supervised learning: neural networks, statistics, & machine learning

Relationship of neural networks to methods of statistics/regression and machine learning.

Rationale: The brain is complicated. Good to understand bottom-up, from neurons to behavior. But also good to understand top-down, from behavior to quantitative models with as few free parameters as possible. But with a view to plausible neural networks. We don’t know enough about neurons to know how a given behavior is implemented with specific neural network.

A good idea to try the simplest model first.

Logistic regression, and the generalized linear model

The single (weight) layer perceptron is a special case of logistic regression, which in turn is a special case of a *generalized linear model*. Logistic regression has been studied by statisticians since the 1950s.

The starting point is wanting to model a binary response Y (think action potential on or off) by the conditional probability:

P(Y=1 | X=x),

where X are the input features. Let's assume that

P(Y=1 | X=x) = p(x;w)

for some parameters w. How to model this function p?

As we've done before, let's assume we want a linear function. But over what function of p?

p(x) = w.x + b has problems because the left side is bounded between 0 and 1, and the right side is unbounded. Log p(x) = w.x + b may seem nice because adding input features multiplies probabilities, but is bounded on only one side. A solution is to model log odds as a linear function:

$$\text{Log}\,\frac{p(x)}{1-p(x)} = w.x + b$$

$$p(x) = \frac{1}{1+e^{-(w.x+b)}}$$

Looking familiar? This says that the probability of Y=1 is determined by taking a linear weighted sum of the inputs plus b (think bias) and deciding "yes" if this is bigger than zero. This is a linear classifier. This is also the probabilistic update rule we used for the Boltzmann machine.

To make things look even more familiar let's calculate the expected value of Y=1 (of a neuron firing). First note that we can write the probability of Y= y_i= 1 or 0 as:

$$p(y_i \mid x) = \frac{e^{y_i\,(w.x+b)}}{1+e^{(w.x+b)}}$$

Then the probability of firing, i.e. y_i = 1 is:

$$p(y_i = 1 \mid x) = \frac{1}{1+e^{-(w.x+b)}} = \sigma(\text{w.x + b})$$

By definition, the average rate is:

$$\sum_{y_i=0}^{1} y_i\, p\,(y_i \mid x) = 0 \times p(y_i = 0 \mid x) + 1 \times p(y_i = 1 \mid x) = \sigma(\text{w.x + b})$$

This is stage 1 and 2 of our original generic neuron model.

The logistic regression model can be extended to multiple class decisions.

These models in turn can be viewed as special cases of generalized linear models.

Note: The general linear model is different! It really is linear. The generalized linear model is in general not. For applications of the generalized linear model to computational neuroscience, see Pillow (2007).

The single layer perceptron seems pretty simple, but...

K nearest neighbor classification

is even simpler. k nearest neighbors or k-NN for short. Has no problem with Xor.
We return to this in a demo below.

Fisher linear discriminant

Find a projection that minimizes the within-class variance, while maximizing the between-class variance.

Support vector machines

Naive Bayes

$$p(C|F_1,\dots,F_n) = \frac{1}{Z}p(C)\prod_{i=1}^{n} p(F_i|C)$$

http://en.wikipedia.org/wiki/Naive_Bayes_classifier

And others, Random Forest, AdaBoost, ...

This link also has a concise description of k-NN.

http://en.wikipedia.org/wiki/Random_forest

Nearest neighbor demonstration: “random Xor”

Define $\mathcal{D}$ to be a mixture distribution of bivariate normals

```
𝒟 = MixtureDistribution[{1, 1, 1, 1},
   {MultinormalDistribution[{0, 0.}, {{1., 0}, {0, 1.}}],
    MultinormalDistribution[{0, 10.}, {{1., 0}, {0, 1.}}],
    MultinormalDistribution[{10, 0.}, {{1., 0}, {0, 1.}}],
    MultinormalDistribution[{10, 10.}, {{1., 0}, {0, 1.}}]}];
```

Here’s a sample:

```
RandomVariate[𝒟]
```

```
{9.53876, 9.56682}
```

We can generate 100 samples:

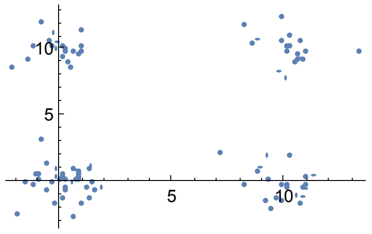
```
somedata = Table[RandomVariate[𝒟], {100}];
```

And then artificially assign labels.

Change below for Xor, Or...And...

```
labels = Which[EuclideanDistance[#, {0, 0}] < 5, 0,
     EuclideanDistance[#, {0, 10}] < 5, 1, EuclideanDistance[#, {10, 0}] < 5,
     1, EuclideanDistance[#, {10, 10}] < 5, 0] &  /@ somedata;
```

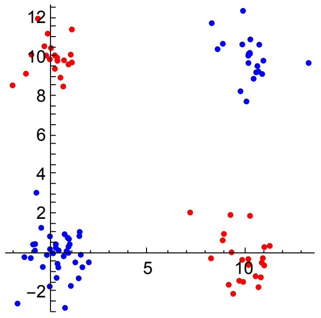
```
ListPlot[somedata]
```



We can color the plots with the labels:

```
colorf = Blend[{{0, Blue}, {1, Red}}, #] &
pl = Graphics[MapThread[{colorf[#1], Point[#2]} &, {labels, somedata}],
  Axes → True, AspectRatio → 1]
```

```
Blend[{{0, Blue}, {1, Red}}, #1] &
```

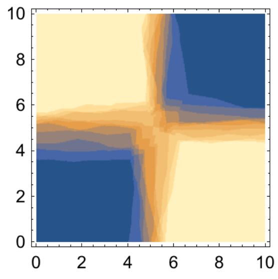


Estimate nearest neighbor boundaries

Now run through lots of points inside {{0,10},{0,10}}. For each point find the k nearest neighbors, and count how many have labels of 1 (i.e. how many dots are “blue”). Plot up the count numbers.

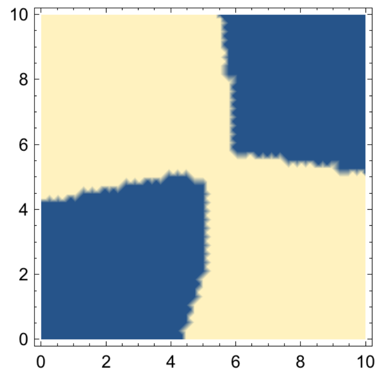
```
nf2[x_, k_] := Total[
    Nearest[Flatten[Table[{somedata[[i]] → labels[[i]]}, {i, 1, 100}], 1], x, k]] / k;
```

```
DensityPlot[nf2[{x, y}, 10], {x, 0, 10}, {y, 0, 10}, PlotPoints → 100]
```



If the number of neighbors with label= +1 is bigger than k/2, output 1, otherwise 0.

```
k = 1;
DensityPlot[If[nf2[{x, y}, k] > 1 / 2, 1, 0], {x, 0, 10}, {y, 0, 10}, PlotPoints → 20]
```



k=1 and Voronoi diagrams

Nearest neighbor regression

“If something is similar to something else in one respect, it is likely to be similar in another respect.”

Fishers linear discriminant demo

Initialize

Read in Statistical Add-in packages:

```
Off[General::"spell1"];
<< "MultivariateStatistics`";
<< "ComputationalGeometry`"
```

Discriminant functions

Let's build our geometric intuitions of what a simple perceptron unit does by viewing it from a more formal point of view. 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:

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

where **w** is the weight vector and $\mathbf{w_0}$ is the (scalar) 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:

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

where $\mathbf{x} = \{x_1, x_2, x_3 ...\}$. We've seen how g(**x**)=0 defines a decision surface which in the linear case is a hyperplane.
Suppose $\boldsymbol{x}_1$and $\boldsymbol{x}_2$ are vectors, with endpoints sitting on the hyperplane, then their difference is a vector lying in the hyperplane

$$\mathbf{w}.\mathbf{x}_1 + \mathbf{w}_0 = \mathbf{w}.\mathbf{x}_2 + \mathbf{w}_0$$
$$\mathbf{w}.(\mathbf{x}_1 - \mathbf{x}_2) = 0$$

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:

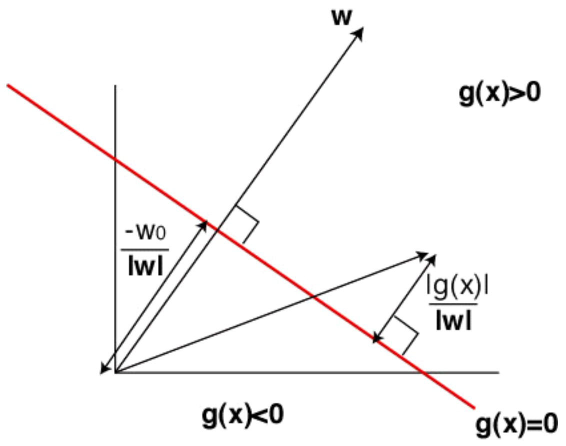
$$\mathbf{w}.\mathbf{x} / |\mathbf{w}| = -\mathbf{w}_0 / |\mathbf{w}|$$

Thus, the threshold determines the position of the hyperplane.

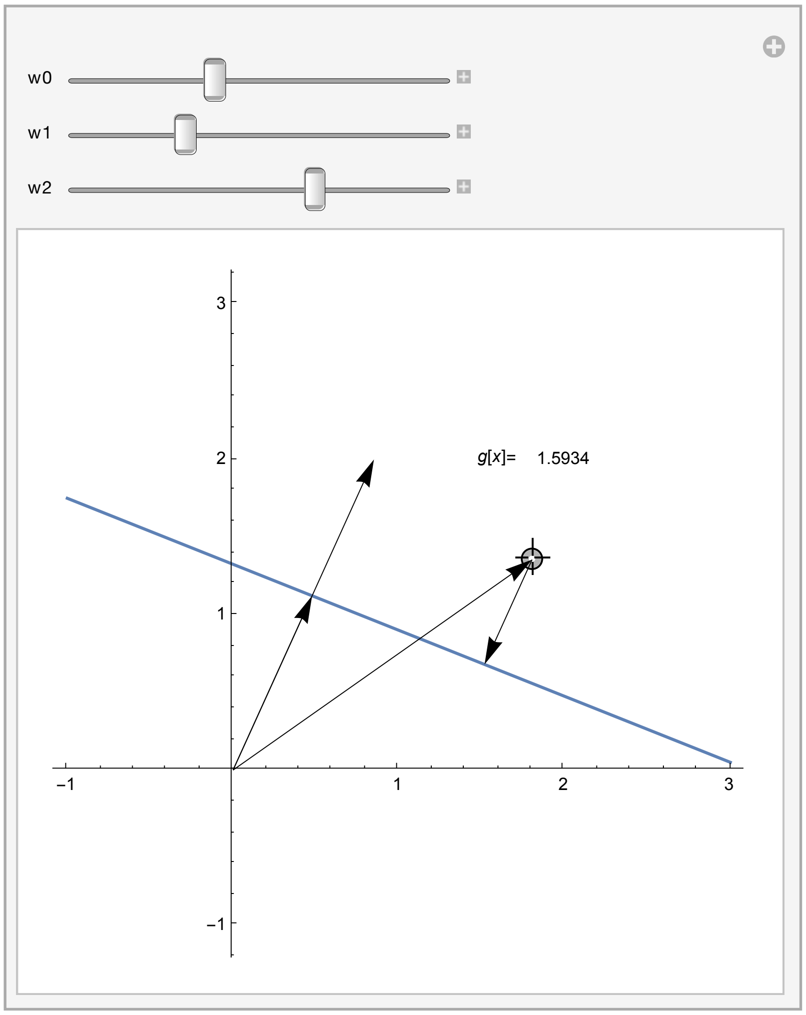
One can also show that the normal distance of x to the hyperplane is given by:

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

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 an input vector is from the hyper-plane.
The figure summarizes the basic properties of the linear discriminant.



```
Clear[w1, w2, w, w0];
Manipulate[
 (* x0={2,1};*)
 w = {w1, w2}; wn = w / Norm[w];
 g[x_] := {w1, w2}.x + w0;
 gg = Plot[Tooltip[
    x2 /. Solve[{w1, w2}.{x1, x2} + w0 == 0, x2], "discriminant"], {x1, -1, 3}];
 ggg = Graphics[g[Dynamic[MousePosition["Graphics"]]]];
 Show[{gg, Graphics[Inset["g[x]=", {1.6, 2}]],
   Graphics[Inset[ToString[g[x0]], {2, 2}]], Graphics[
    {Tooltip[Arrow[{{0, 0}, w}], "w"], Tooltip[Arrow[{{0, 0}, (-w0 / Norm[w]) * wn}],
      "-w0/|w|"], Tooltip[{Arrow[{{0, 0}, x0}]}, "x"],
     Tooltip[{Arrow[{x0, x0 - wn * g[x0] / Norm[w]}]}, "g(x)/|w|"]}]},
  PlotRange → {{-1, 3}, {-1, 3}}, AxesOrigin → {0, 0}, Axes → True,
  AspectRatio → 1, ImageSize → Medium],
 {{w0, -2.5}, -6, 3}, {{w1, 1}, 0, 3}, {{w2, 2}, 0, 3},
 {{x0, {2, 1}}, Locator}]
```



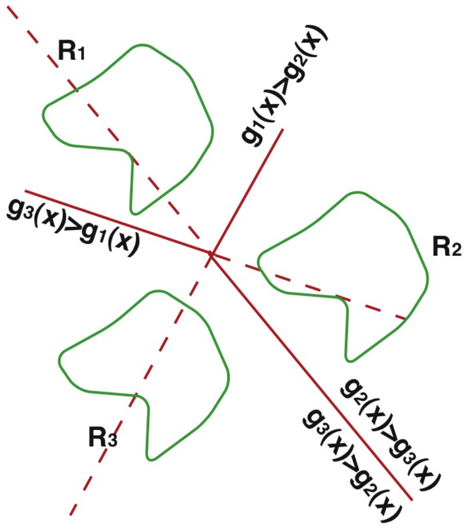
Multiple classes

Suppose there are c classes. There are a number of ways to define multiple class discriminant rules.

One way that avoids undefined regions is:

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

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



(Adapted from Duda & Hart, 1973)

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 peaks, this linear classifier won't do a good job either.

Task-dependent Dimensionality reduction

Motivation

Later, when we consider the problem of dimensionality reduction, we will take another look at hyperplanes. But here the idea will be to find hyperplanes onto which we can project our input data, and from there divide up the hyperplane into decision regions. 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 (males) 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?

Fisher's linear "discriminant"

Generative model: two nearby gaussian classes

Define two bivariate base distributions

```
(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

```
pdf = PDF[ndista, {x1, x2}]
```

$$1.12822\, e^{\frac{1}{2}(-x1\,(50.2513\,x1-49.7487\,(x2+1))-(x2+1)\,(50.2513\,(x2+1)-49.7487\,x1))}$$

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

```
Mean[ndistb]
```

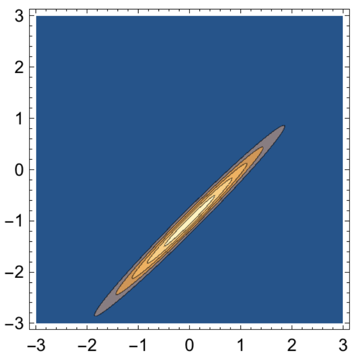
```
{0, 1}
```

```
Covariance[ndista]
```

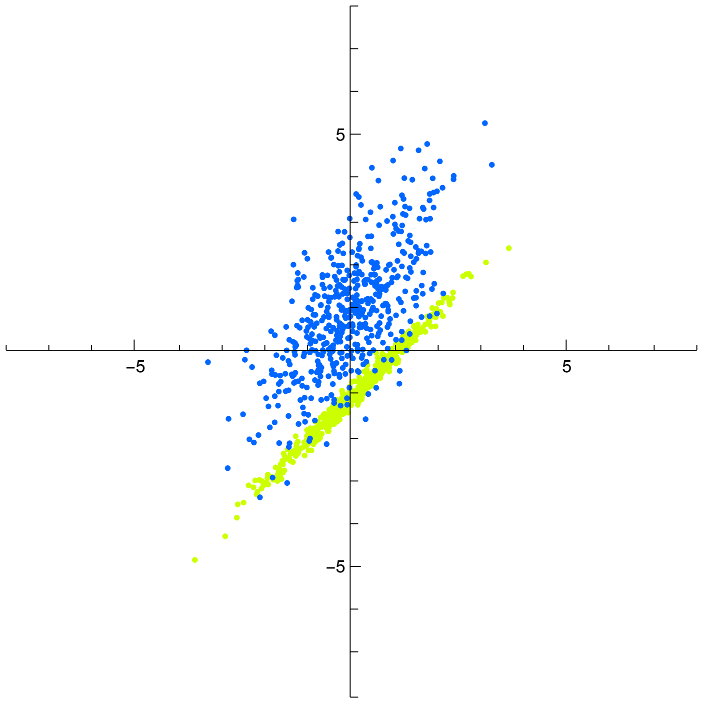
```
{{1, 0.99}, {0.99, 1}}
```

Try different covariant matrices. Should they be symmetric? Constraints on the determinant of ar, br?
Make a contour plot of the PDF ndista

```
pdfa = PDF[ndista, {x1, x2}];
ContourPlot[pdfa, {x1, -3, 3}, {x2, -3, 3}, PlotPoints → 64, PlotRange → All]
```



```
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?

```
Mean[b]
```

```
{-0.0634543, 0.930223}
```

```
Covariance[b]
```

```
{{1.05063, 0.957104}, {0.957104, 1.98301}}
```

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

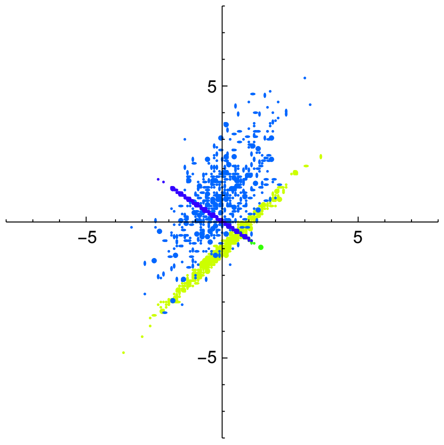
```
m = -2 / 3;
wnvec = {1, m} / Sqrt[1 + m^2];
```

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

```
{n1 x + n2 y}
```

```
{{n1 (n1 x + n2 y), n2 (n1 x + n2 y)}}
```

```
aproj = (#1 wnvec &) /@ (a.wnvec);
gaproj = ListPlot[aproj, AspectRatio → 1,
   PlotStyle → Hue[0.3`], DisplayFunction → Identity];
bproj = (#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. Plot out the marginal distributions relative to this line.

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

```
Mean[aproj]
```

```
{0.473876, -0.315917}
```

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:

$$|\mathbf{w}\cdot(m_a - m_b)|$$

and

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

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

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

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

```
Mean[a] - Mean[b]
```

```
{0.0831499, -1.92741}
```

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

```
wnvec.(Mean[a] - Mean[b])
```

```
1.13832
```

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:

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

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

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

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

$$S_B = (m_1 - m_2).(m_1 - m_2)^T$$

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

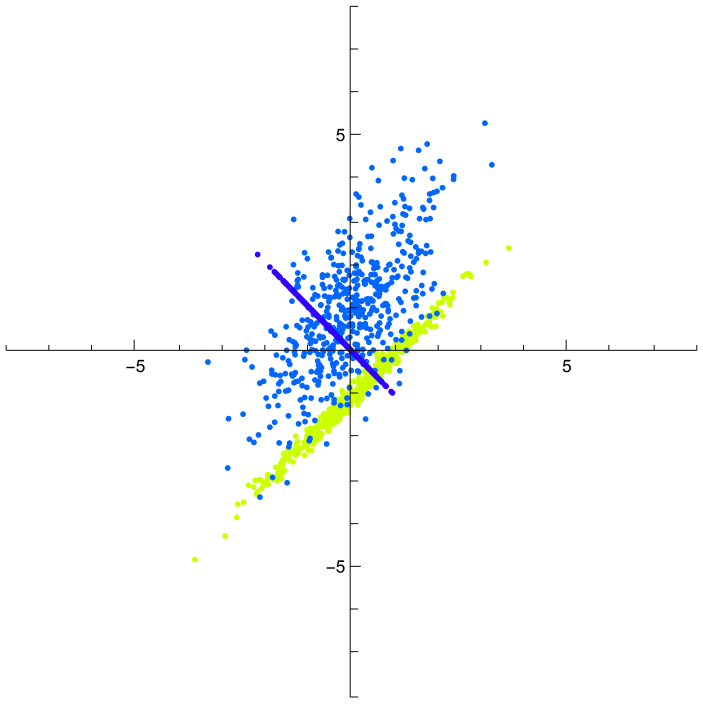
$$J(w) = \frac{w^T.S_B.w}{w^T.S_W.w}$$

Answer:

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

Demo: Finding Fisher's linear discriminant

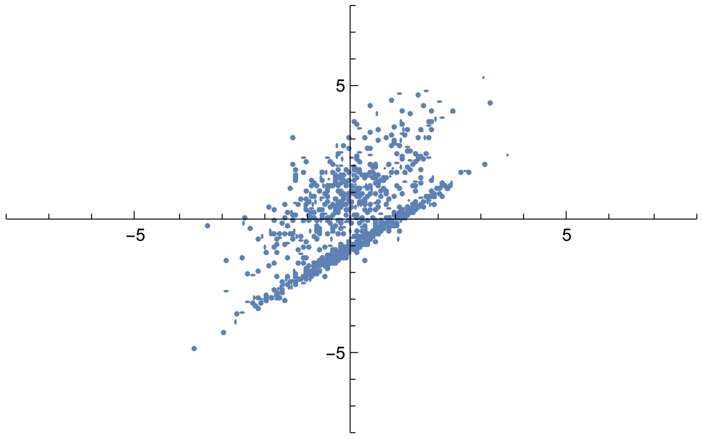
```
normalize[x_] := x / Sqrt[x.x];

ma = Mean[a];
mb = Mean[b];

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)]

{0.697928, -0.716168}

aproj = (#1 wldf &) /@ (a.wldf);
gaproj = ListPlot[aproj, AspectRatio → 1,
   PlotStyle → Hue[0.3`], DisplayFunction → Identity];
bproj = (#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

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



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

```
auto = Covariance[c]
eigvalues = Eigenvalues[auto]
eigauto = Eigenvectors[auto]
```

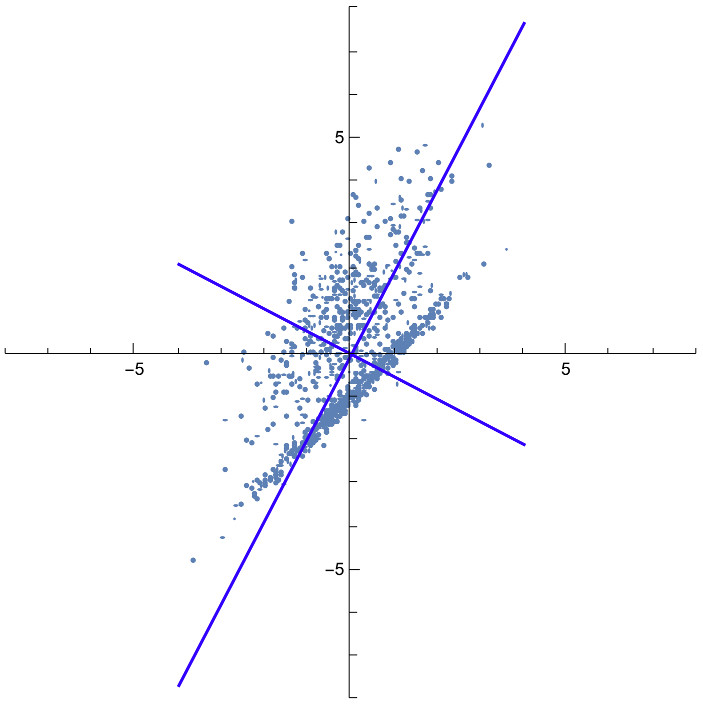
```
{{1.09252, 0.999432}, {0.999432, 2.48674}}
```

```
{3.00816, 0.571092}
```

```
{{0.462554, 0.886591}, {-0.886591, 0.462554}}
```

```
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]}];
```

```
Show[g1, gPCA, DisplayFunction → $DisplayFunction]
```



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

SVMs and Kernel methods

Initialize

Read in Add-in packages:

```
Off[General::"spell1"];
<< "ErrorBarPlots`";
<< "MultivariateStatistics`";
```

```
Ellipsoid::shdw :
  Symbol Ellipsoid appears in multiple contexts {MultivariateStatistics`, System`}; definitions in context
      MultivariateStatistics` may shadow or be shadowed by other definitions. ≫
```

Make sure the SVM package is downloaded in the default directory

```
SetDirectory[NotebookDirectory[]]
```

```
/Users/kersten/Sites/kersten-lab/courses/Psy5038WF2014/Lectures/Lect_15
```

```
<< MathSVMv7`
```

We assume a simple perceptron TLU to classify vector data x into one of two classes depending on the sign of g(x):

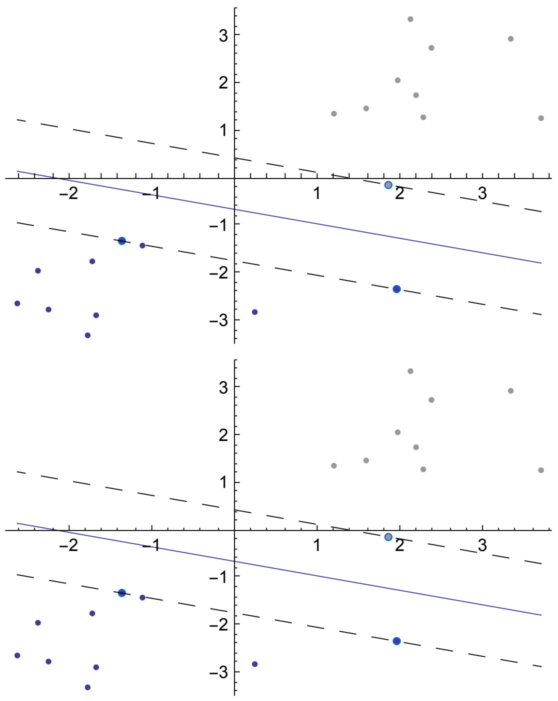
decision(x) = sign($w.x + b$).

Given g (**x**) = **w.x** + b, recall that g(x)/||w|| is the distance of a data point x from a plane defined by g(x) = 0. In support vector machines, the goal is to find the separating plane (i.e. find w and b) that is as far as possible from any of the data points. The intuition is that this will minimize the probability of making wrong classifications when given new data at some future point. Formally, we want to solve"

$$\max_{(w,b)}\left(\min_i d\left(\Pi_{w,b}, x_i\right)\right),$$

where $d(\Pi_{w,b}, x_i) = g(x) / || w ||$, i.e.

$d(\Pi_{w,b}, x_i) = g(x) / || w || = | w.x_i + b | / || w ||$



Two-class data (black and grey dots), their optimal separating hyperplane (continuous line), and support vectors (circled in blue). This is an example output of the `SVMPlot` function in *MathSVM*. The width of the “corridor” defined by the two dotted lines connecting the support vectors is the margin of the optimal separating hyperplane. (From Nilsson et al., 2006)

The Primal Problem

It can be shown that the optimal separating hyperplane solving (1) can be found as the solution to the equivalent optimization problem

$$\min_{w,b} \frac{1}{2} || w ||^2$$
$$\text{subject to } y_i\left(w^T x_i + b\right) \geq 1,$$

Typically, equality will hold for a relatively small number of the data vectors. These data are termed *support vectors.* The solution (w, b) depends only on these specific points, and in effect contain all the information for the decision rule. The "dual problem".

A Simple linear SVM Example (from Nilsson et al. 2006)

Here's a demo of a simple SVM problem. It uses the add on package *MathSVMv7 written by Nilsson et al.*

```
len = 20;
X = Join[
   RandomReal[NormalDistribution[-2, 1], {len / 2, 2}],
   RandomReal[NormalDistribution[2, 1], {len / 2, 2}]];
y = Join[Table[1, {len / 2}], Table[-1, {len / 2}]];
```

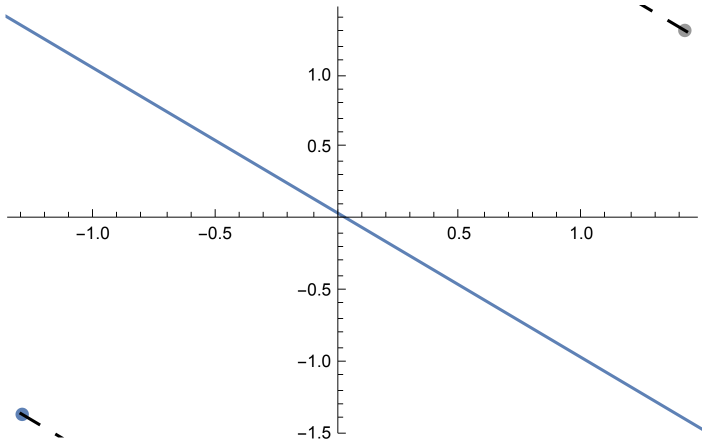
We use the simple SVM formulation provided in *MathSVM* by the `SeparableSVM` function.

```
τ = 0.01;
α = SeparableSVM[X, y, τ]
```

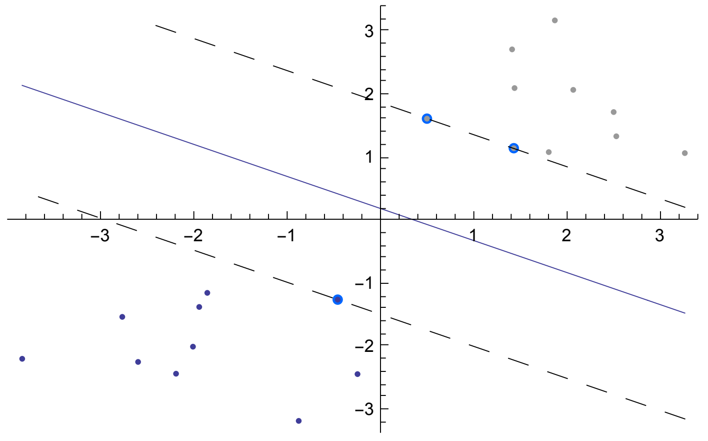
```
{0, 0, 0, 0.137456, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.137456, 0, 0, 0, 0, 0}

{0, 0, 0.190548, 0, 0, 0, 0, 0, 0, 0, 0.00155191, 0, 0, 0, 0, 0, 0, 0, 0, 0.188996}

SeparableSVM[{{-0.270935, -2.65218}, {-1.95276, -2.52008}, {-1.89739, -1.07845},
   {-0.763816, -2.32397}, {-0.88129, -0.999857}, {-2.13166, -0.312347},
   {-0.675034, -3.00938}, {-3.05368, -2.05241}, {-4.04021, -2.05777},
   {-1.08124, -2.69893}, {2.10368, 0.889012}, {1.99685, 2.99536},
   {2.99501, 2.22825}, {0.724624, 2.59584}, {3.77872, 2.63434}, {0.657296, 1.73201},
   {0.841787, 1.65132}, {1.93301, 1.91092}, {3.3824, 2.3385}, {1.41159, 1.72466}},
  {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 0.01]

{0, 0.222806, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.126342, 0.0964638, 0, 0, 0, 0, 0, 0, 0}
```

In the output figure below, the solid line marks the optimal hyperplane, and dotted lines mark the width of the corridor that joins support vectors (highlighted in blue).

```
SVMPlot[α, X, y]
```



```
SVMPlot[
 SeparableSVM[{{-0.270935, -2.65218}, {-1.95276, -2.52008}, {-1.89739, -1.07845},
   {-0.763816, -2.32397}, {-0.88129, -0.999857}, {-2.13166, -0.312347},
   {-0.675034, -3.00938}, {-3.05368, -2.05241}, {-4.04021, -2.05777},
   {-1.08124, -2.69893}, {2.10368, 0.889012}, {1.99685, 2.99536},
   {2.99501, 2.22825}, {0.724624, 2.59584}, {3.77872, 2.63434}, {0.657296, 1.73201},
   {0.841787, 1.65132}, {1.93301, 1.91092}, {3.3824, 2.3385}, {1.41159, 1.72466}},
  {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 0.01],
 {{-0.270935, -2.65218}, {-1.95276, -2.52008}, {-1.89739, -1.07845},
  {-0.763816, -2.32397}, {-0.88129, -0.999857}, {-2.13166, -0.312347},
  {-0.675034, -3.00938}, {-3.05368, -2.05241}, {-4.04021, -2.05777},
  {-1.08124, -2.69893}, {2.10368, 0.889012}, {1.99685, 2.99536},
  {2.99501, 2.22825}, {0.724624, 2.59584}, {3.77872, 2.63434}, {0.657296, 1.73201},
  {0.841787, 1.65132}, {1.93301, 1.91092}, {3.3824, 2.3385}, {1.41159, 1.72466}},
 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}]
```



A Nonlinear Example: Using Kernels (from Nilsson et al. 2006)

What if the data are not linearly separable? The essential idea is to map the data (through some non-linear mapping, e.g. polynomial) to a higher-dimensional "feature" space to find the optimal hyperplane separating the data. The dot product gets replaced by a non-linear kernel function. For example, the polynomial kernel is given by:

$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j})^d$

$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j})^d$

If d = 1, we have the standard dot product, but for d = 2, 3, etc.. we have polynomial functions of the elements of the vectors x. See Nilsson et al, and paper by Jäkel (2009) for more information on kernels.

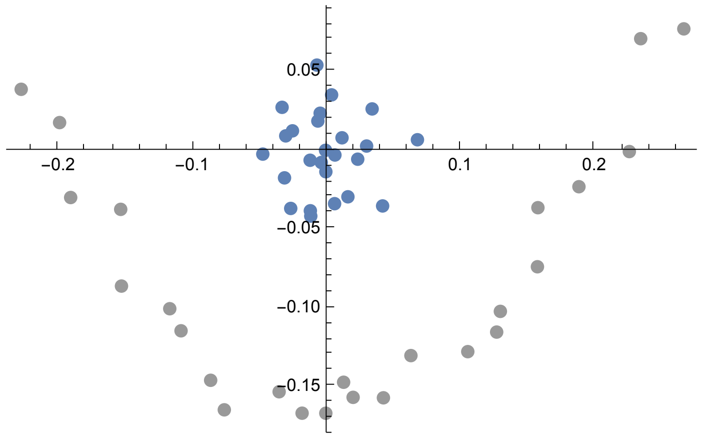
Here is a demo for an application for nonlinear classification . We'll use second-degree kernel:

```
PolynomialKernel[x, y, 2]
```

```
(1 + x.{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1})^2
```

Some synthetic data which is not linearly separable.

```
len = 50;
X = Join[
   RandomReal[NormalDistribution[0, 0.03], {len / 2, 2}],
   Table[
    {RandomReal[NormalDistribution[i / len - 1 / 4, 0.01]],
     Random[NormalDistribution[(2 i / len - 1 / 2)^2 - 1 / 6, 0.01]]},
    {i, len / 2}]];
y = Join[Table[1, {len / 2}], Table[-1, {len / 2}]];
SVMDataPlot[X, y, PlotRange → All]
```



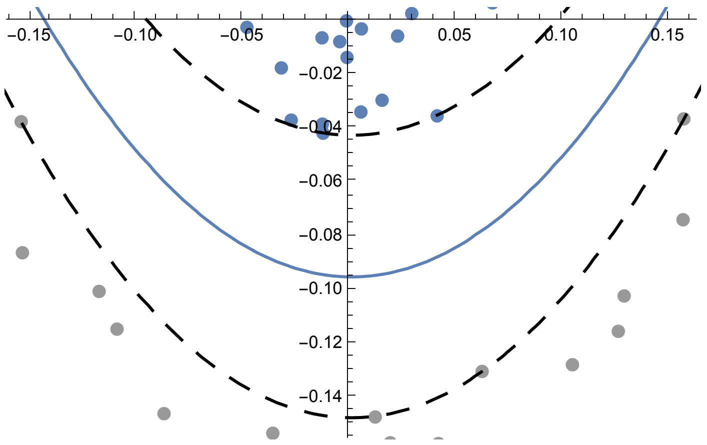
We use the `KernelFunction` to specify the kernel type and run **SeparableSVM[]**.

```
τ = 0.01;
pk = PolynomialKernel[#1, #2, 2] &;
α = SeparableSVM[X, y, τ, KernelFunction → pk]
```

```
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3865.71, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 1424.3, 0, 0, 0, 0, 0, 0, 0, 0, 29.2739, 0, 0, 4.32638, 0, 0, 0, 0, 2407.8, 0, 0, 0, 0}
```

When visualizing the results, `SVMPlot` can use the kernel functions to draw any nonlinear decision curves.

```
SVMPlot[α, X, y, KernelFunction → pk]
```



```
Clear[len, X, y, α, pk]
```

More information

The wiki entry for SVMs has a fairly good introduction (As of fall 2014).

To go to the source see Vapnik (1995)

http : // svm.first.gmd.de/

Demo links:

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

And for applications of kernel methods more generally, to cognitive and neuroscience see reviews by Jäkel et al. (2006; 2009).

Mathematica’s Classify[ ] function demo

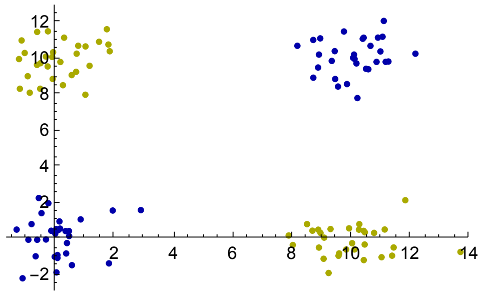
Here we adapt code from one of *Mathematica*’s examples to apply it to the noisy Xor problem.

Define clusters sampled from normal distributions:

```
sampledata[center_] :=
  RandomVariate[MultinormalDistribution[center, IdentityMatrix[2]], 30];
```

Generate clusters, assign labels to each and plot;

```
clusters = sampledata /@ {{0, 0}, {0, 10}, {10, 0}, {10, 10}};
colors = {Blue, Yellow, Yellow, Blue};
ListPlot[clusters, PlotStyle → Darker@colors]
```



```
Dimensions[clusters]
```

```
{4, 30, 2}
```

The finagling in the next line is to assign all of the 2D points to their corresponding colors, and then running the associations through the Classifier.

```
Flatten[Thread[Transpose[clusters][[#]] → colors] & /@ Range[30], 1]
```

```
c3 = Classify[Flatten[Thread[Transpose[clusters][[#]] → colors] & /@
    Range[Dimensions[clusters][[2]]], 1]]
```

```
ClassifierFunction[ Method: NearestNeighbors
                    Number of classes: 2 ]
```

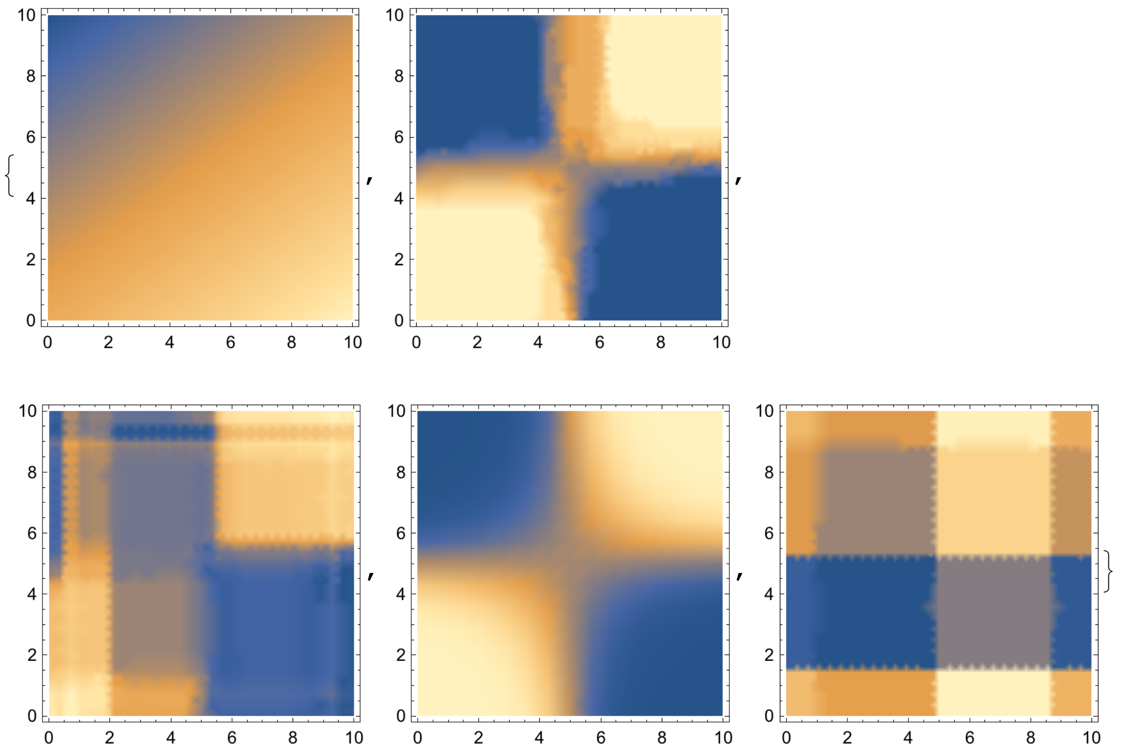
How do probabilities get calculated?

```
Normal@c3[{2, 7}, "Probabilities"][[2]]
```

```
0.97619
```

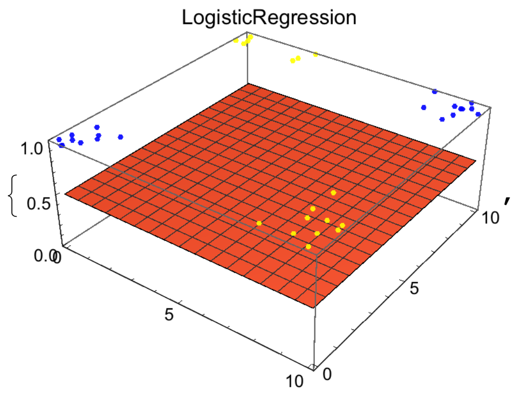
```
plotprob[method_] := Module[{},
   c3 =
    Classify[Flatten[Thread[Transpose[clusters][[#]] → colors] & /@ Range[30], 1],
     Method → method];
   DensityPlot[Normal@c3[{x, y}, "Probabilities"][[1]], {x, 0, 10}, {y, 0, 10}]
  ];
```

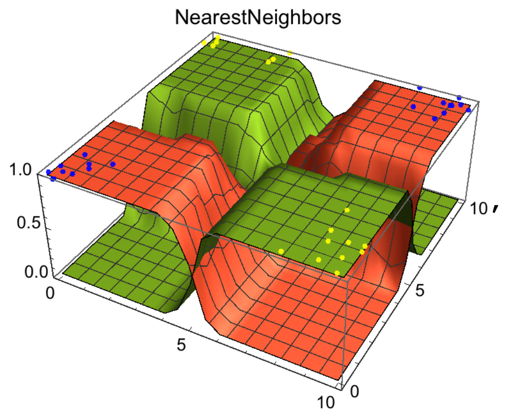
```
plotprob /@ {"LogisticRegression", "NearestNeighbors",
  "RandomForest", "SupportVectorMachine", "NaiveBayes"}
```

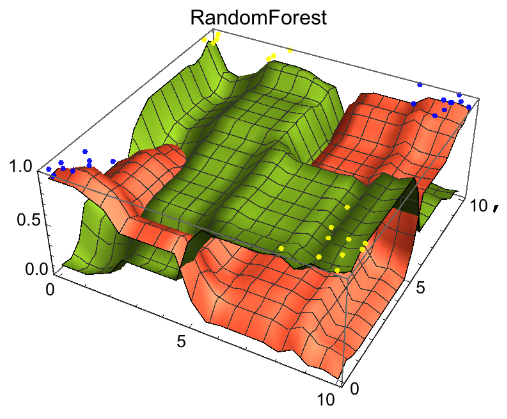


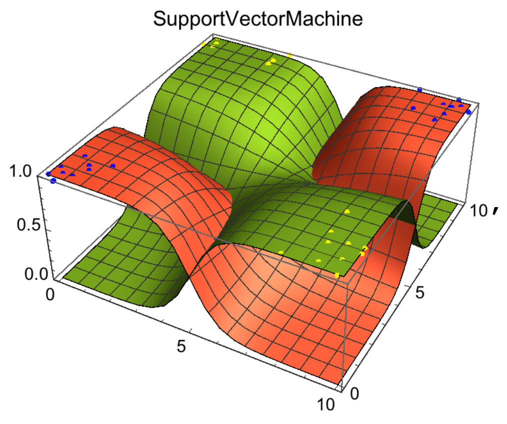
```
plotprobabilities[method_] := Module[{c, proba},
   c3 = Classify[Flatten[
      Thread[Transpose[clusters][[#]] → colors] & /@ Range[30], 1], Method → method];
   proba = Table[Append[{x, y}, #] & /@ Lookup[c3[{x, y}, "Probabilities"], colors],
     {x, 0, 10, .5}, {y, 0, 10, .5}];
   proba = Flatten[#, 1] & /@ Transpose[proba, {3, 2, 1, 4}];
   Show[
    ListPlot3D[proba, PlotRange → {0, 1}, PlotLabel → method],
    ListPointPlot3D[Map[Append[#, 1] &, clusters, {2}],
     PlotStyle → ({Opacity[.9], #} & /@ colors)], ImageSize → 250
   ]
];
```

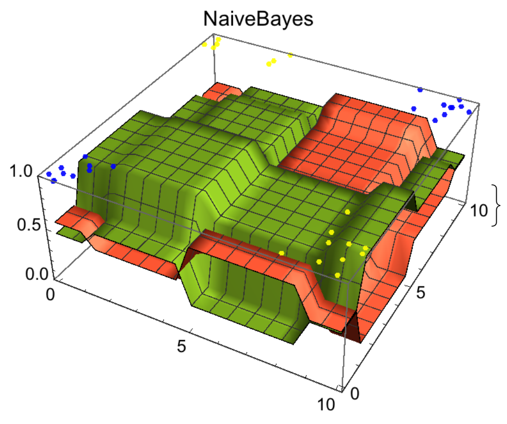
```
plotprobabilities /@ {"LogisticRegression", "NearestNeighbors",
   "RandomForest", "SupportVectorMachine", "NaiveBayes"}
```











Appendix

Mathematica code to illustrate Bayesian estimation of surface slant and aspect ratio

This code was used to produce the figure in a Nature Neuroscience News & Views article by Geisler and Kersten (2002). (See paper by Weiss, Simoncelli and Adelson.)

Initialization

```
npoints = 128;
loaspect = 0;
hiaspect = 5;
$TextStyle = {FontFamily → "Helvetica", FontSize → 14}
Fswitch = True;
```

```
{FontFamily → Helvetica, FontSize → 14}
```

```
PadMatrix[mat_, gray_, n_] := Module[{d},
   d = Dimensions[mat];
   Return[PadRight[PadLeft[mat, {d[[1]] + n, d[[2]] + n}, gray],
     {d[[1]] + 2 * n, d[[2]] + 2 * n}, gray]];
  ];
```

Init delta

```
gdelta[x_, w_] := 1 - (UnitStep[x + w / 2] - UnitStep[x - w / 2]);
(*Plot[gdelta[x,1],{x,-10,10},PlotRange→{0,2}];*)
```

Calculate Likelihood function and its maxima

$$p(I \mid S_{prim}, S_{\sec})$$

$$p(x \mid \alpha, d) = p(x - \phi(\alpha, d))$$

$$x = \phi(\alpha, d) + noise$$

(We've used the notion "prim" and "sec" for primary and secondary variables. But below rather than integrating out the secondary variable, we use a loss function to soften the notion of what is important and what is not. We"ll require more precision of the slant estimate than of the aspect ratio.)

Image model determines the constraint, x = d Cos[alpha] + noise, determines the likelihood

Assume noise has a Gaussian distribution with standard deviation = 1/5;
Assume an image measurement (x=1/2)

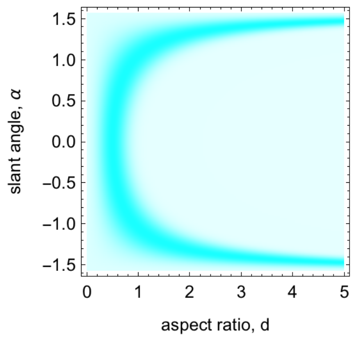
```
likeli[alpha_, x_, d_, s_] :=
 Exp[-((x - d Cos[alpha])^2) / (2 s^2)] (1 / Sqrt[2 Pi s^2])
likeli[α, x, d, s]
x = 1 / 2;  s = 1 / 5;
like = likeli[α, x, d, s]
```

$$\frac{e^{-\frac{(x-d\,\mathrm{Cos}[\alpha])^2}{2 s^2}}}{\sqrt{2\pi}\sqrt{s^2}}$$

$$\frac{5\, e^{-\frac{25}{2}\left(\frac{1}{2}-d\,\mathrm{Cos}[\alpha]\right)^2}}{\sqrt{2\pi}}$$

Plot likelihood

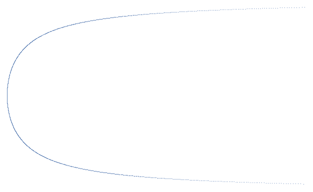
```
gdlike = DensityPlot[like, {d, loaspect, hiaspect}, {α, -π/2, π/2}, PlotPoints → npoints,
   Mesh → False, ColorFunction → (RGBColor[1 - (0.1+ 0.8 #1), 1, 1] &),
   FrameLabel → {"aspect ratio, d", "slant angle, α"}]
```



Plot likelihood maxima

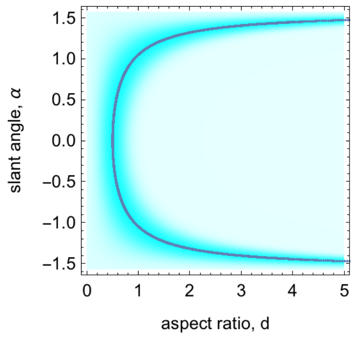
There is no unique maximum. The likelihood function has a ridge

```
gtemp29 = ListPlot[Table[{N[x] / Cos[alpha], alpha}, {alpha, -π/2., π/2., 0.001}],
   ImageSize → Small, Axes → False]
```



Plot likelihood together with maximum along the ridge

```
gdlike = DensityPlot[like, {d, loaspect, hiaspect}, {α, -π/2, π/2}, PlotPoints → npoints,
   Mesh → False, ColorFunction → (RGBColor[1 - (0.1` + 0.8` #1), 1, 1] &),
   FrameLabel → {"aspect ratio, d", "slant angle, α"}, Frame → Fswitch];
glikemax = Show[gdlike, gtemp29]
```



Calculate the prior, and find its maximum

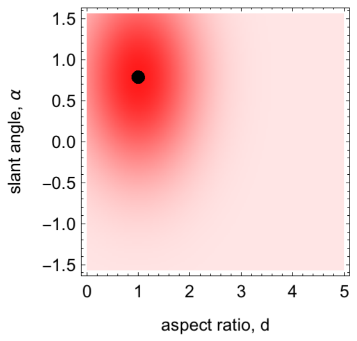
$$p(S_{prim}, S_{\sec})$$

$$p(\alpha, d)$$

The prior probability distribution corresponds to the assumption that surface patches tend to be slanted away at the top and have aspect ratios closer to 1.0. We model the prior by a bivariate gaussian:

```
PDF[MultinormalDistribution[{μα, μd}, R], {α, d}];

R1 = {{.25, 0}, {0, .25}};
ndist3 = MultinormalDistribution[{Pi / 4., 1}, R1];
pdf3 = PDF[ndist3, {α, d}];
FindMinimum[-pdf3, {{d, 0}, {α, 1}}]
gdprior = DensityPlot[pdf3 ^ .4, {d, loaspect, hiaspect},
   {α, -Pi / 2, Pi / 2}, PlotPoints → npoints, Mesh → False,
   ColorFunction -> (RGBColor[1, 1 - (0.1 + 0.8 #), 1 - (0.1 + 0.8 #)] &),
   FrameLabel → {"aspect ratio, d", "slant angle, α"}];
{-0.63662, {d → 1., α → 0.785398}}
```

```
Show[gdprior, Graphics[{PointSize[0.05`], Point[{1, 0.785`}]}]]
```



Calculate the posterior, and find its maximum

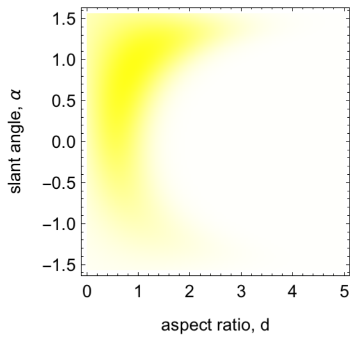
$$p(S_{prim}, S_{sec} \mid I) \propto p(I \mid S_{prim}, S_{sec})p(S_{prim}, S_{sec})$$

$$p(\alpha, d \mid x) = \frac{p(x \mid \alpha, d)p(\alpha, d)}{p(x)}$$

$$p(\alpha, d \mid x) \propto p(x \mid \alpha, d)p(\alpha, d)$$

More precisely, we'll calculate a quantity proportional to the posterior. The posterior is equal to the product of the likelihood and the prior, divided by the probability of the image measurement, x. Because the image measurement is fixed, we only need to calculate the product of the likelihood and the prior:

```
Clear[α, x, d, s];
likeli[α, x, d, s] * PDF[MultinormalDistribution[{μα, μd}, R], {α, d}];

gdpost = DensityPlot[(pdf3 * like) ^.2, {d, loaspect, hiaspect}, {α, -Pi / 2, Pi / 2},
  ColorFunction -> (RGBColor[1, 1, 1 - (0.01 + 0.9 #)] &), PlotPoints → npoints,
  Mesh → False, FrameLabel → {"aspect ratio, d", "slant angle, α"}, Frame → Fswitch]
```

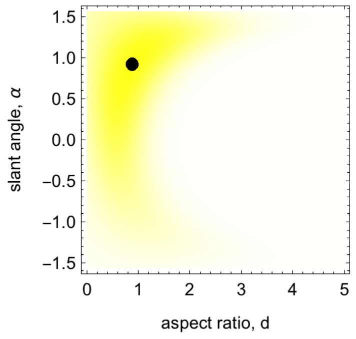


```
R1 = {{.25, 0}, {0, .25}};
ndist3 = MultinormalDistribution[{Pi / 4., 1}, R1];
pdf3 = PDF[ndist3, {α, d}]
FindMinimum[-pdf3 * like, {{d, 1}, {α, 1}}]
```

$$0.63662\, e^{\frac{1}{2}(-(0.+4.\,(-1+d))\,(-1+d)-(0.+4.\,(-0.785398+\alpha))\,(-0.785398+\alpha))}$$

```
{-1.17378, {d → 0.881475, α → 0.923647}}
```

```
Show[gdpost, Graphics[{PointSize[0.05], Point[{0.88, 0.92}]}]]
```



Compute expected loss--i.e. risk, and find its minimum

The expected loss is given by the convolution of the loss with the posterior:

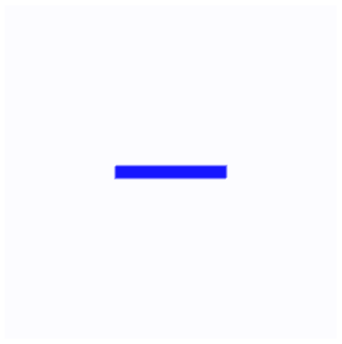
risk=posterior*loss, where * means convolve; utility= - risk

Loss function

$$l(\Delta\alpha, \Delta d) = l(\alpha' - \alpha, d' - d)$$

The asymmetric utility function corresponds to the assumption that it is more important to have an accurate estimate of slant than aspect ratio. The loss function reflects the task. Accurate estimates of slant may be more important for an action such as stepping, whereas an accurate estimation of aspect ratio may be more important for determining object shape (circular coffee mug top or not?). If one were to grasp an object that is elliptical in shape with say, the thumb on the bottom and finger on top, this task could require accurate estimates of both slant and aspect ratio.

```
maploss = Table[(1 - gdelta[x1d, 0.25`]) (1 - gdelta[x2d, 2]),
   {x1d, -3, 3, 6/npoints}, {x2d, -3, 3, 6/npoints}];
gdloss = ListDensityPlot[maploss, Mesh → False,
  ColorFunction → (RGBColor[1 - (0.01+ 0.9 #1), 1 - (0.01+ 0.9 #1), 1] &), Frame → False]
```



Convolve posterior with loss function

$$utility(\alpha', d') = -\sum_{\alpha, d} p(x \mid \alpha, d) p(\alpha, d) l(\alpha' - \alpha, d' - d)$$

Convert function description to numerical arrays for convolving

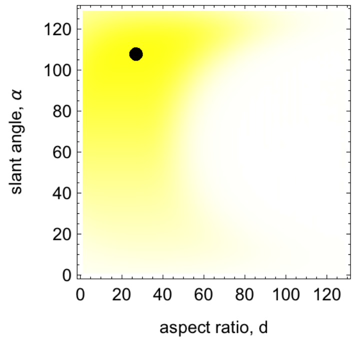
```
post = Transpose[Table[like * pdf3, {d, loaspect, hiaspect,
      (hiaspect - loaspect) / npoints}, {α, -Pi / 2, Pi / 2, Pi / npoints}]];
post2 = PadMatrix[post, 0, 16];
maploss2 = PadMatrix[maploss, 0, 16];
offset = Floor[Dimensions[maploss2][[1]] / 2];
tempcon = ListConvolve[maploss2, post2, {-1, -1}];
risk2 = RotateLeft[tempcon, {offset, offset}];
risk =
  Take[risk2, {17, Dimensions[risk2][[1]] - 16}, {17, Dimensions[risk2][[1]] - 16}];
```

```
grbrisk = ListDensityPlot[Map[#^1. &, risk]^.2,
   Mesh → False, ColorFunction -> (RGBColor[1, 1, 1 - (0.01 + 0.9 #)] &),
   FrameLabel → {"aspect ratio, d", "slant angle, α"}, Frame → Fswitch];
```

```
Position[(risk), Max[(risk)]]
```

```
{{108, 27}}
```

```
Show[grbrisk, Graphics[{PointSize[0.05], Point[{27, 108}]}]]
```



References

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification* (2nd ed.). New York: Wiley. (Amazon.com)
Geisler Wilson S. and Daniel Kersten (2002) Illusions, perception and Bayes. Nature Neuroscience, 5 (6), 508-510. Or (pdf).
http://gandalf.psych.umn.edu/~kersten/kersten-lab/papers/GeislerKerstennn0602-508.pdf
http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&list_uids=12037517
Green, D. M., & Swets, J. A. (1974). Signal Detection Theory and Psychophysics . Huntington, New York: Robert E. Krieger Publishing Company.
Hsu, M., Bhatt, M., Adolphs, R., Tranel, D., & Camerer, C. F. (2005). Neural systems responding to degrees of uncertainty in human decision-making. Science, 310(5754), 1680-1683.

Kersten, D & Mamassian, P (2009) Ideal Observer Theory. In: Squire LR (ed.) Encyclopedia of Neuroscience, volume 5, pp. 89-95. Oxford: Academic Press.
Kersten, Masmassian, P., & Yuille, A. (2004). Object perception as Bayesian inference. Annual Review of Psychology, 55, 271–304.
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.
Pillow, J. (2007). Likelihood-based approaches to modeling the neural code. In K. Doya, S. Ishii, A. Pouget, & R. Rao, Bayesian Brain: Probablistic Approaches to Neural Coding (pp. 53–70). MIT Press Cambridge, MA.
Vapnik, V. N. (1995). *The nature of statistical learning*. New York: Springer-Verlag.
http://neuron.eng.wayne.edu/software.html
Yuille, A. L., & Bülthoff, H. H. (1996). Bayesian decision theory and psychophysics. In K. D.C. & R. W. (Eds.), Perception as Bayesian Inference. Cambridge, U.K.: Cambridge University Press.
Yuille, A., Coughlan J., Kersten D. (1998) (pdf)