# Psych 253
## Advanced Statistical Modeling

### Classification Part 2: SVMs and Logistic Regression

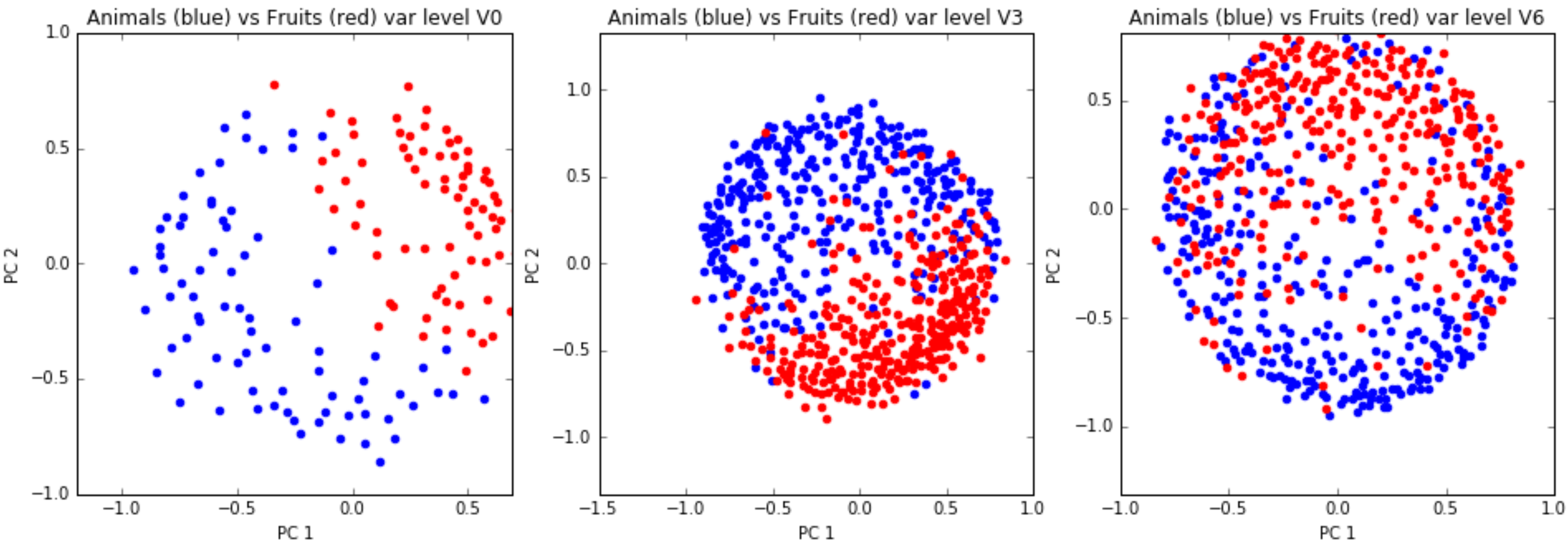
Daniel Yamins
Wu Tsai Neurosciences Institute
Departments of Psychology and Computer Science
Stanford Artificial Intelligence Laboratory
Stanford University

Russ Poldrack
Department of Psychology
Stanford University

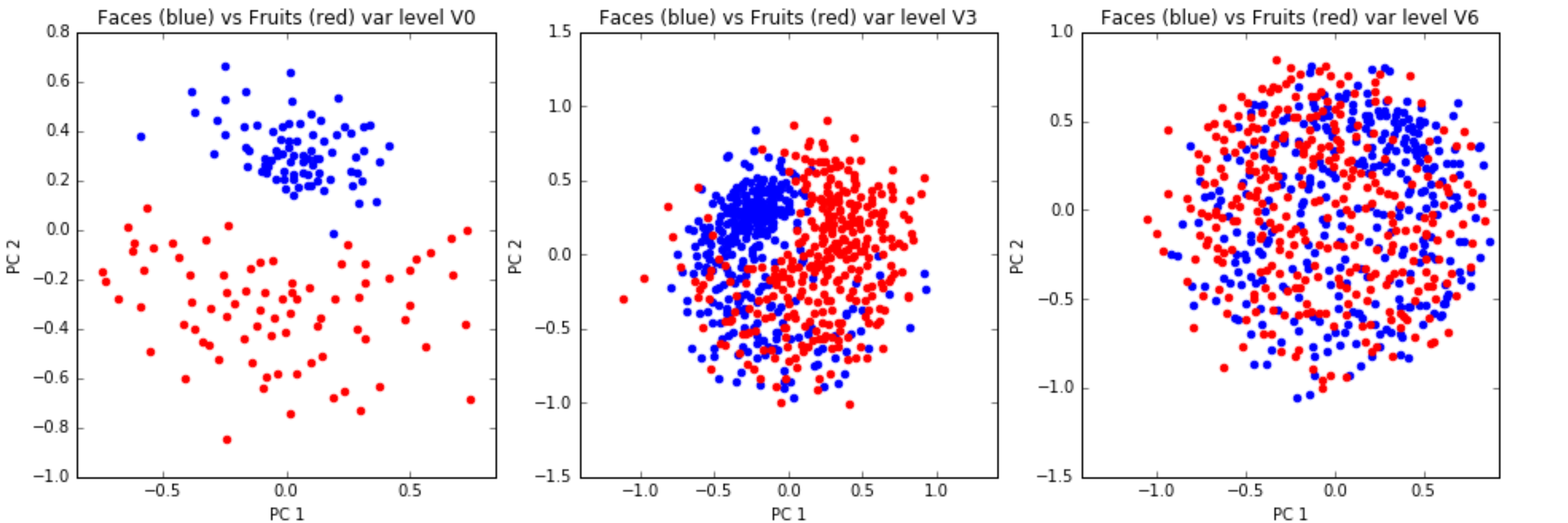
# The Problem with Distance-Based Classifiers

*[IPYNB: Two-D Projection of Binary Class]*

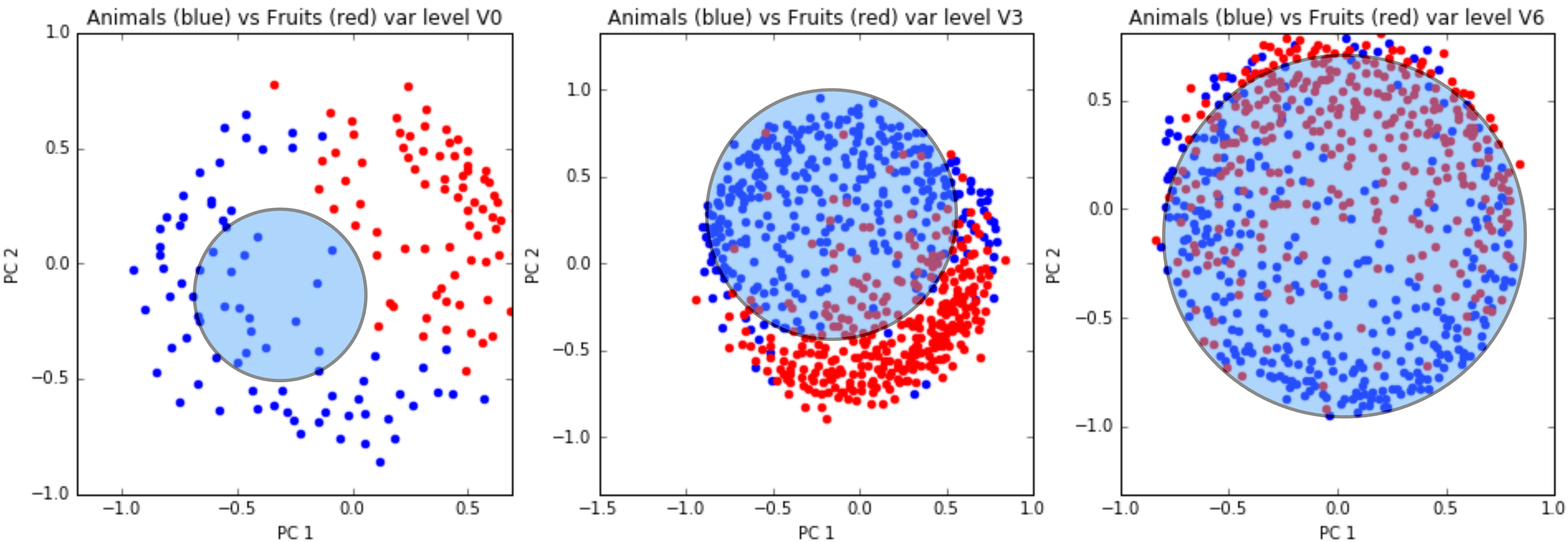
# The Problem with Distance-Based Classifiers



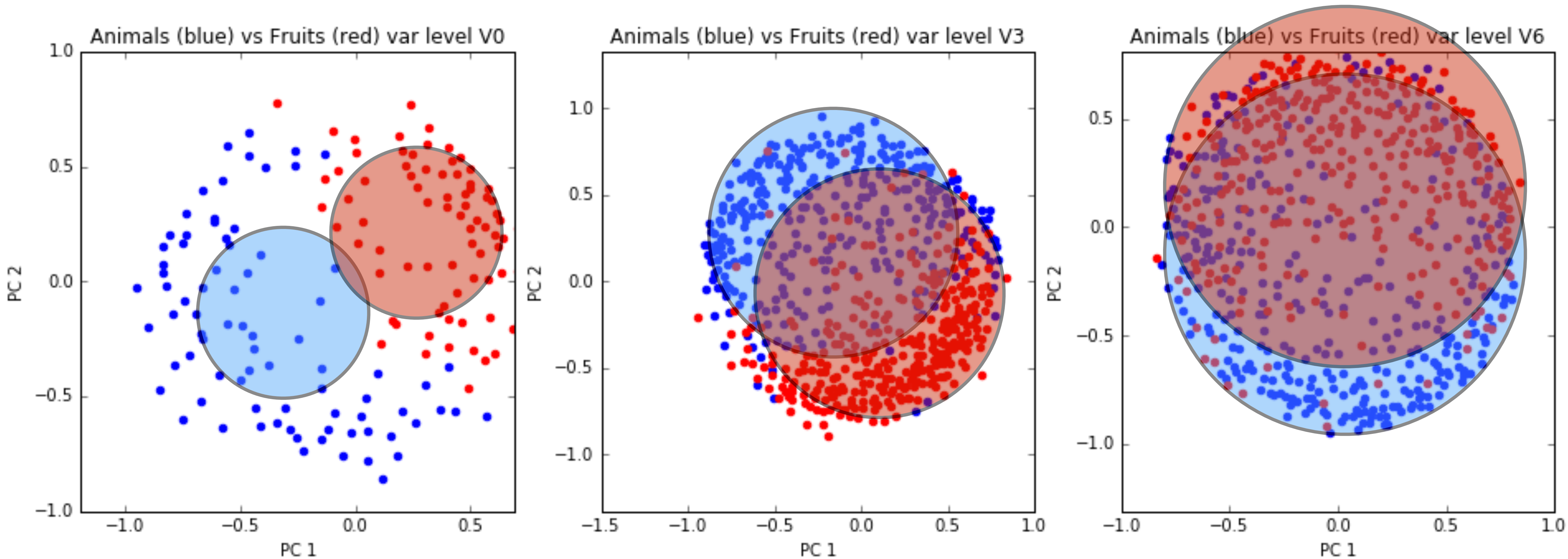
# The Problem with Distance-Based Classifiers



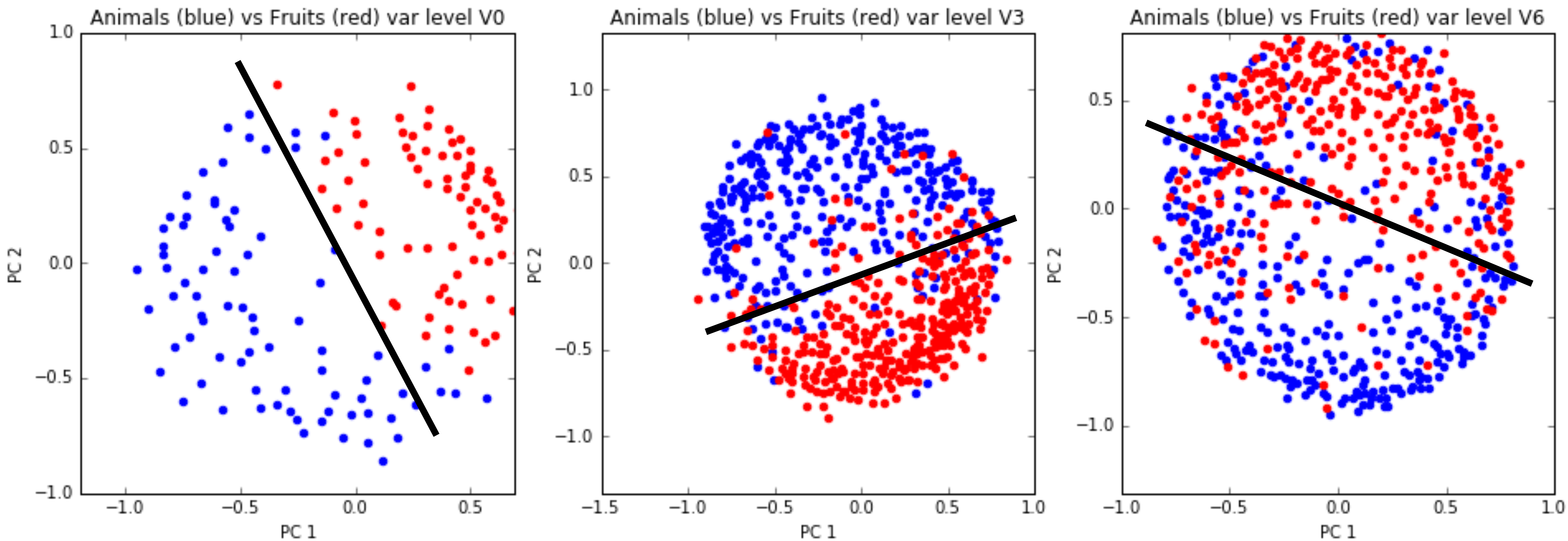
# The Problem with Distance-Based Classifiers



# The Problem with Distance-Based Classifiers

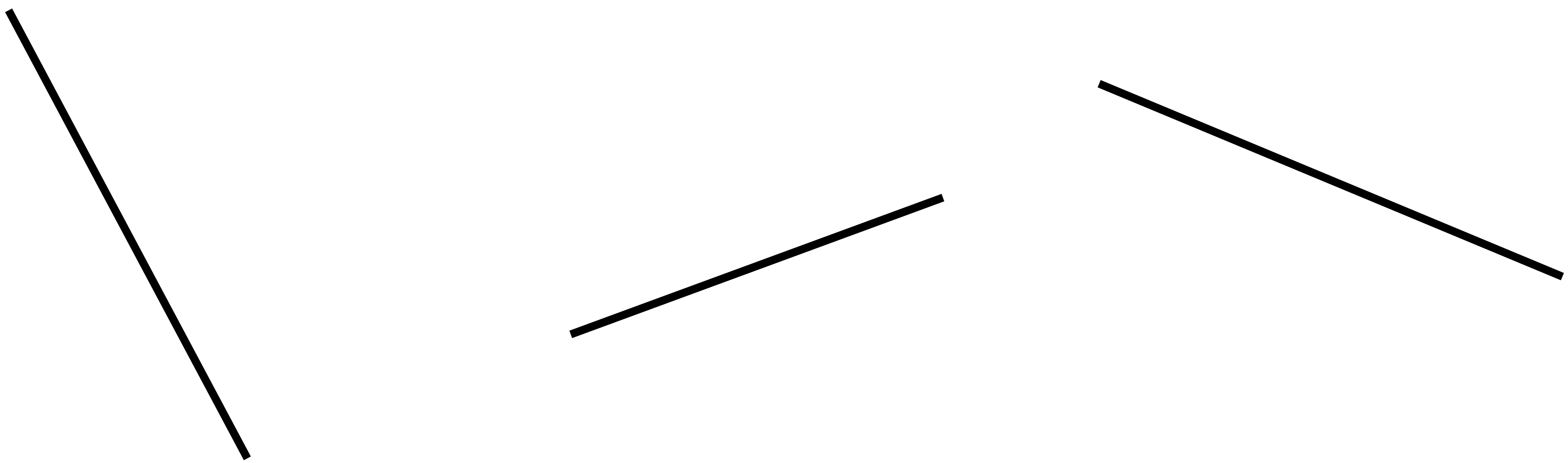


Support Vector Machines (SVMs): let's draw lines the space instead



# Support Vector Machines

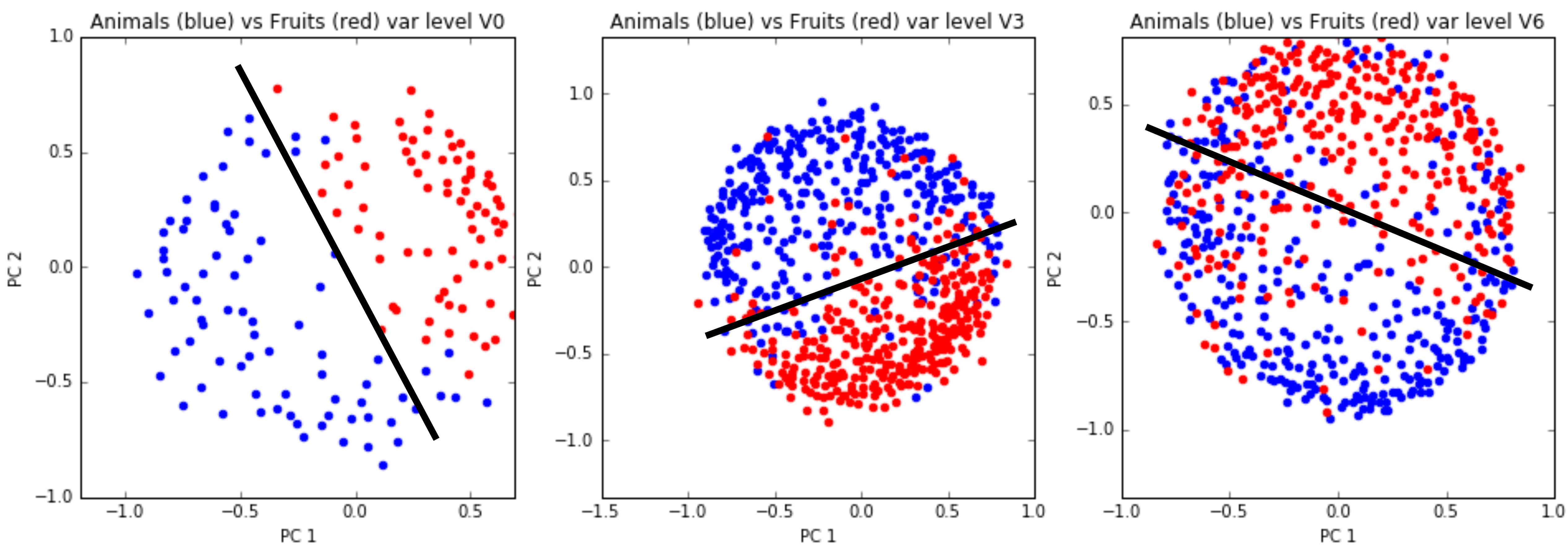
What is a line in high dimensional space?   A "hyperplane".



$$\left(\sum_i w_i \mathrm{neuron}_i\right) + b$$

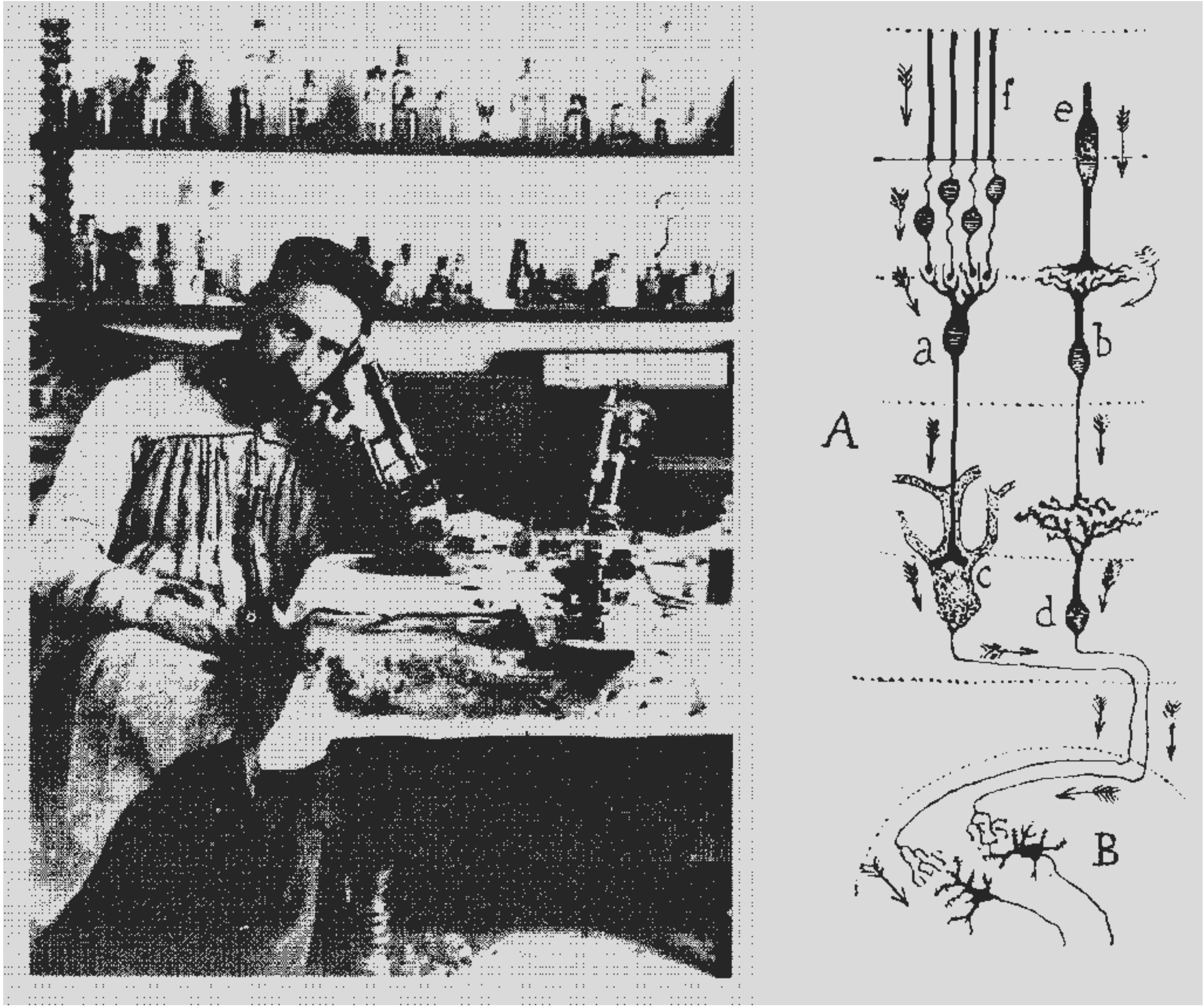
# Support Vector Machines

What is a line in high dimensional space?   A "hyperplane".




$$\text{red} \sim \text{sign}\left[\left(\sum_i w_i \text{neuron}_i\right) + b\right]$$

# Aside: The Neuron Doctrine




***Ramon y Cajal*** *from Rodieck (1973)*

# Aside: The Neuron Doctrine

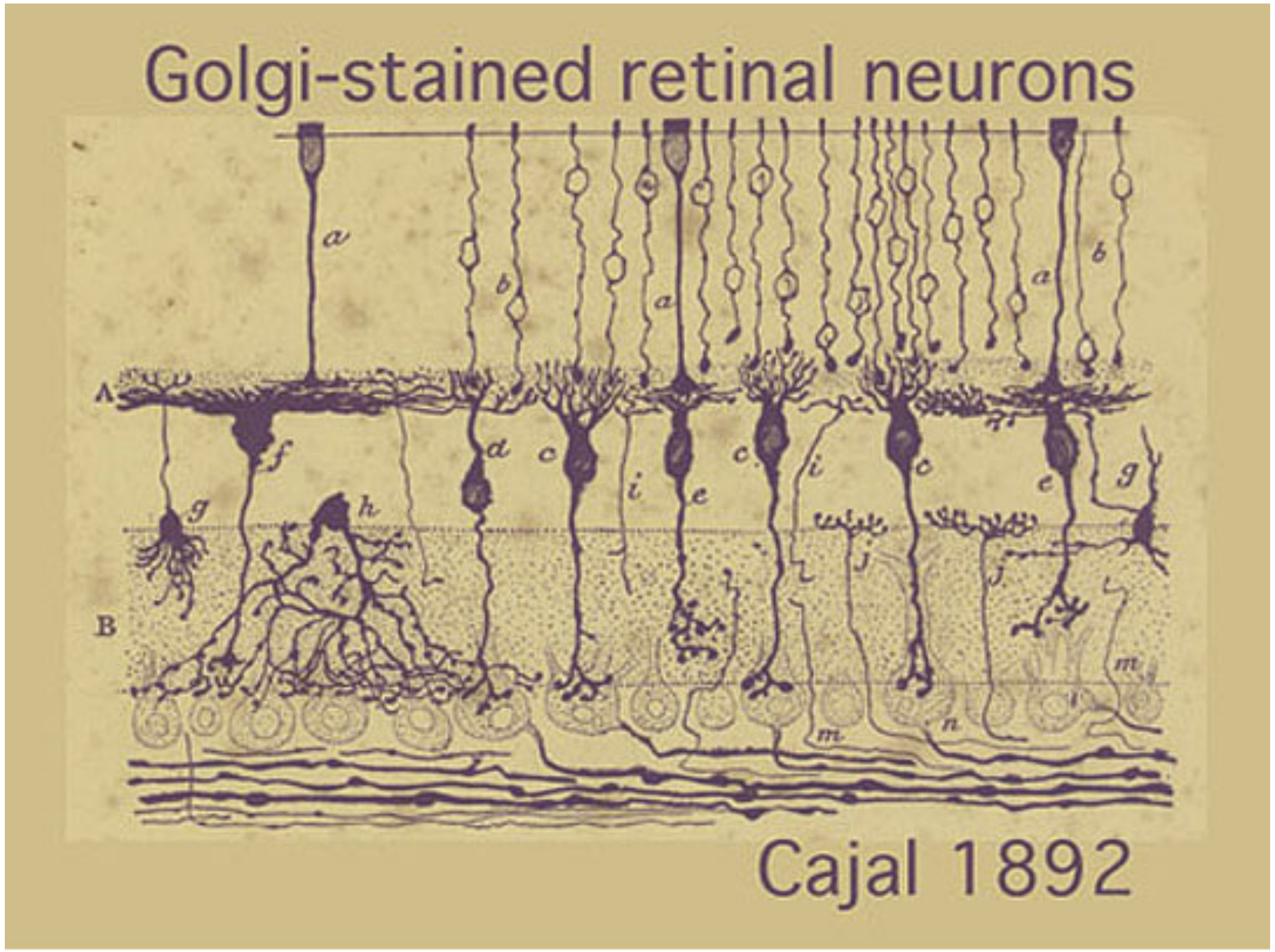
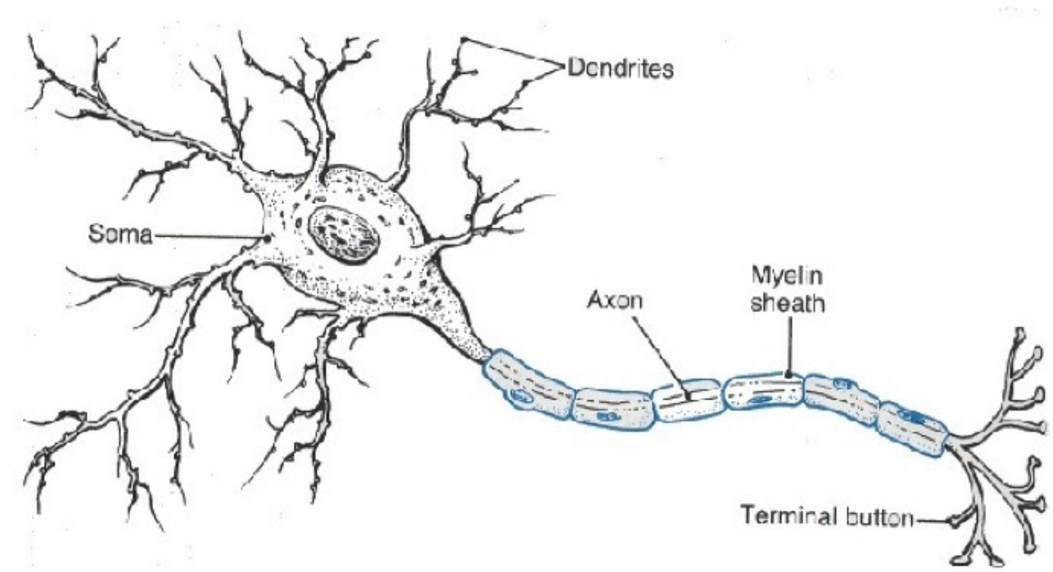


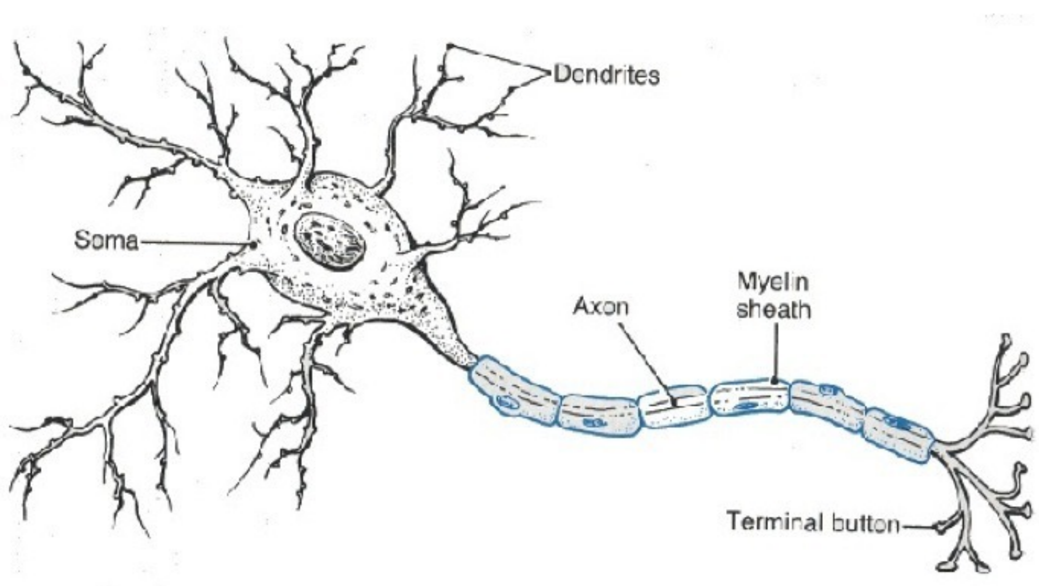

Fig. 2. A drawing done by Cajal to show some of the neurons of the retina in vertical section.

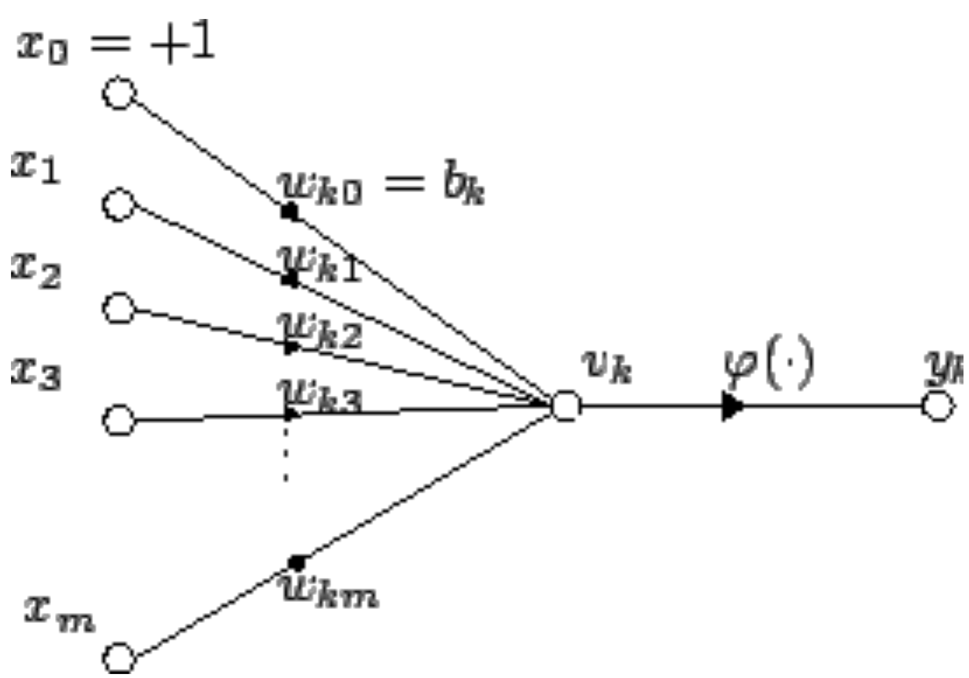



neurons are the nodes of a directed graph;
information propagates along the connections between networks

# Artificial Neural Networks (ANNs)

## McCulloch and Pitts (1943)




$$y_k = \phi\left(\sum_{j=0}^{m} w_{kj} x_j + b_j\right)$$

$$\phi : \mathbb{R} \longmapsto \mathbb{R}$$

some (nonlinear) activation function

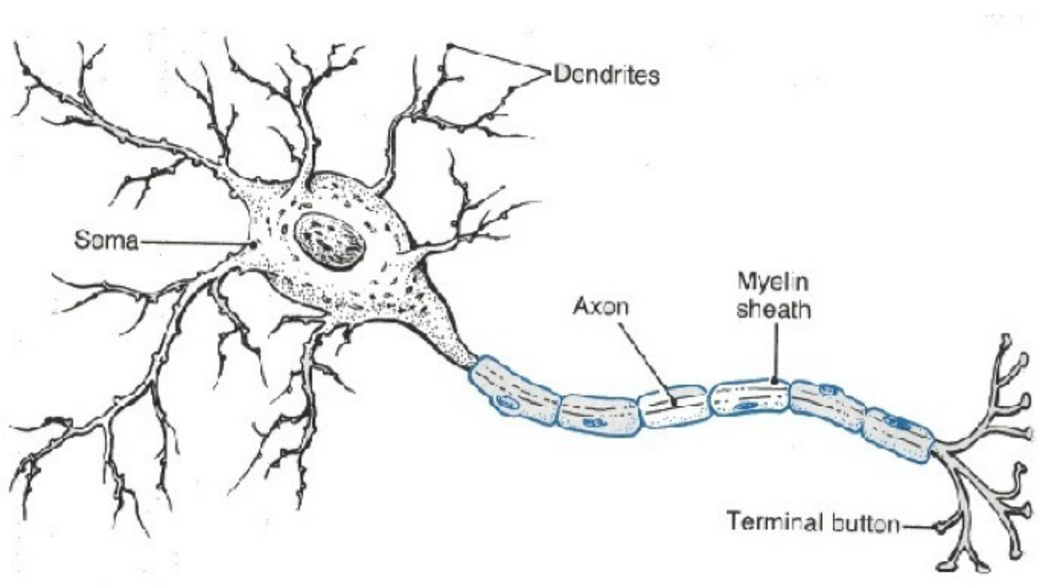
$$w_{kj} \in \mathbb{R}^{m+1}$$

"synaptic strengths"

$$b_j \in \mathbb{R}$$

"biases"

$$\text{red} \sim \text{sign}\left[\left(\sum_i w_i \text{neuron}_i\right) + b\right]$$ exactly of this form

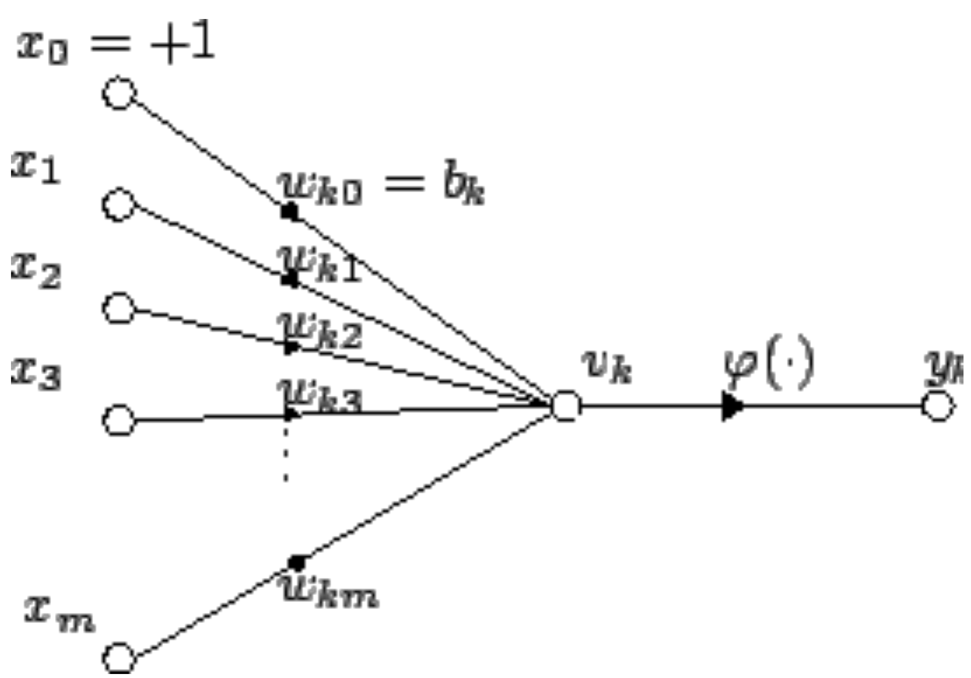



$$y_k = \phi\left(\sum_{j=0}^{m} w_{kj} x_j + b_j\right)$$

with $\phi = \text{sign}(\cdot)$

$$\phi : \mathbb{R} \longmapsto \mathbb{R}$$

some (nonlinear) activation function




$$w_{kj} \in \mathbb{R}^{m+1}$$

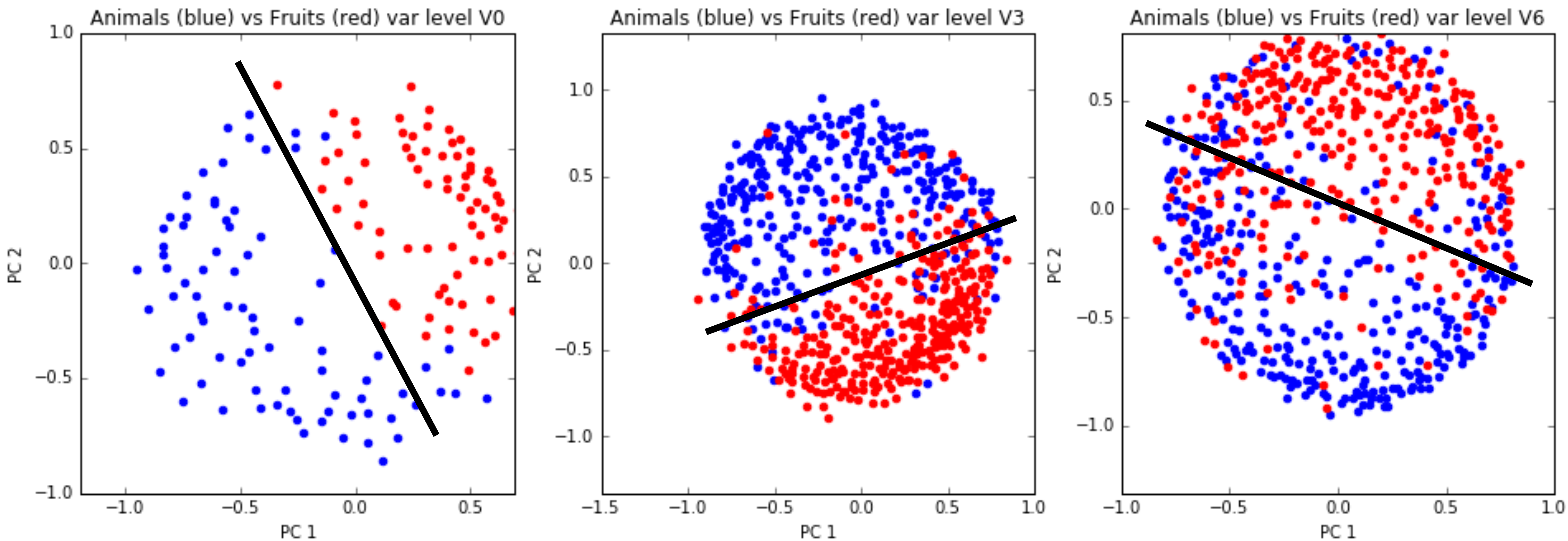
"synaptic strengths"

$$b_j \in \mathbb{R}$$

"biases"

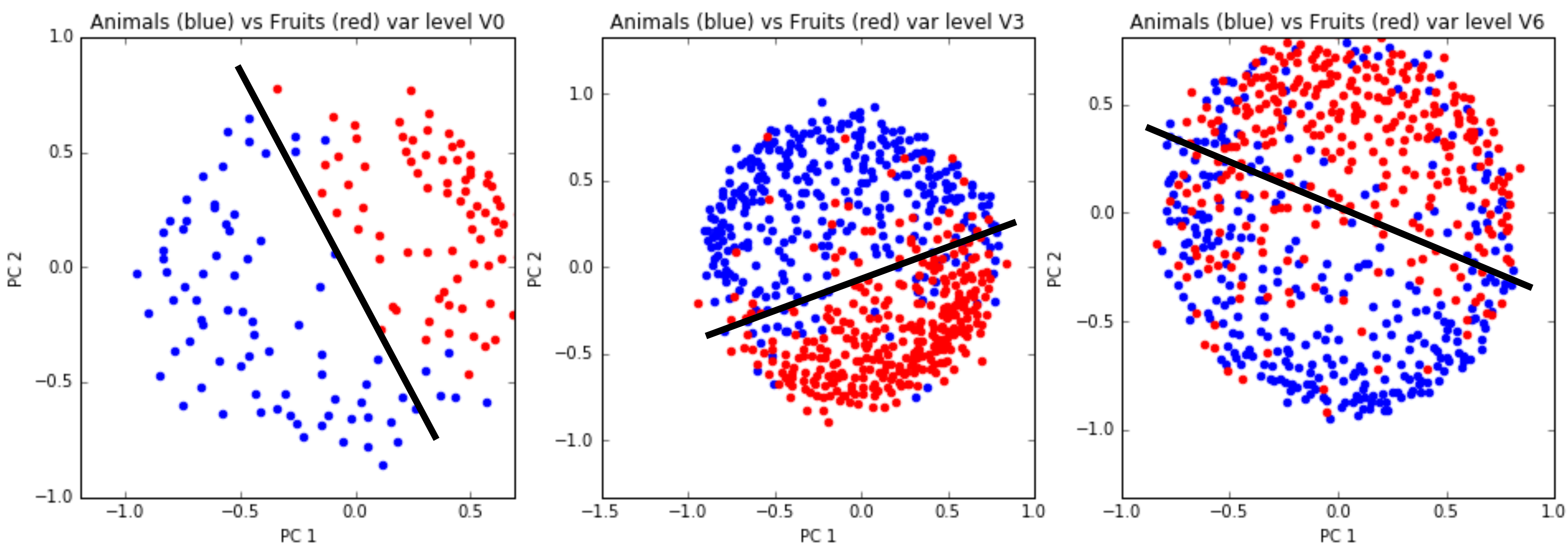
# Support Vector Machines

Which hyperplane do we want?



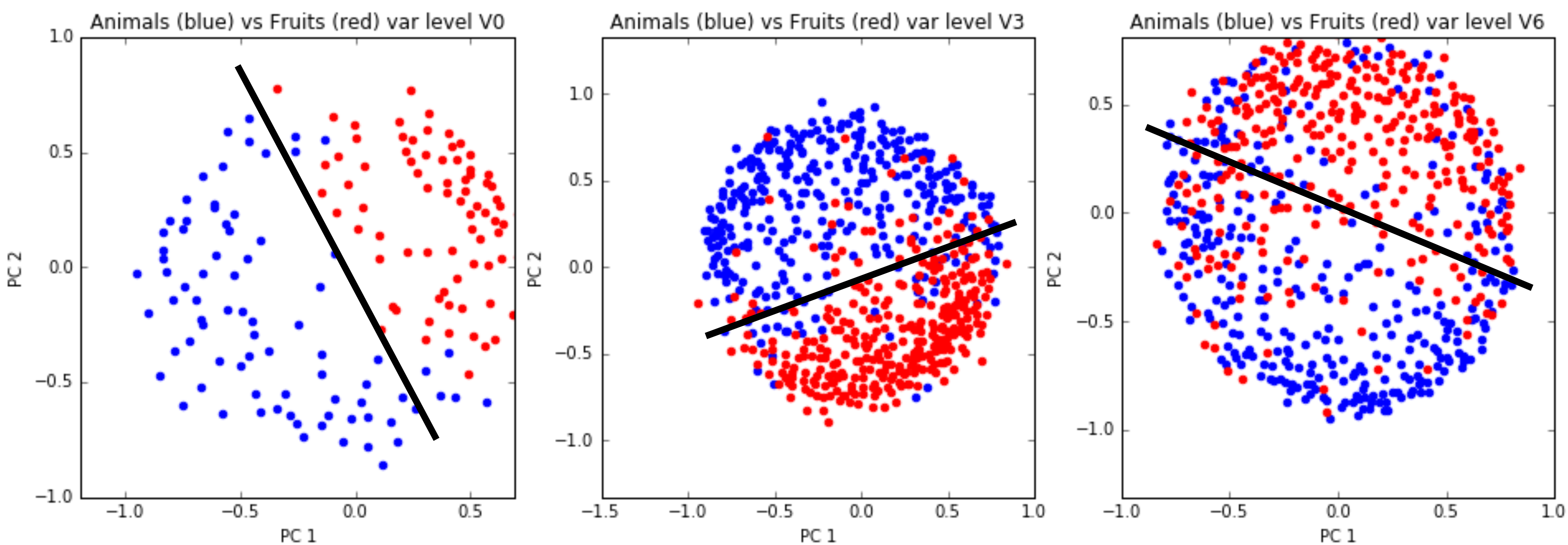
# Support Vector Machines

Which hyperplane do we want? One that separates the points! (Duh)



# Support Vector Machines

Which hyperplane do we want? One that separates the points! (Duh)



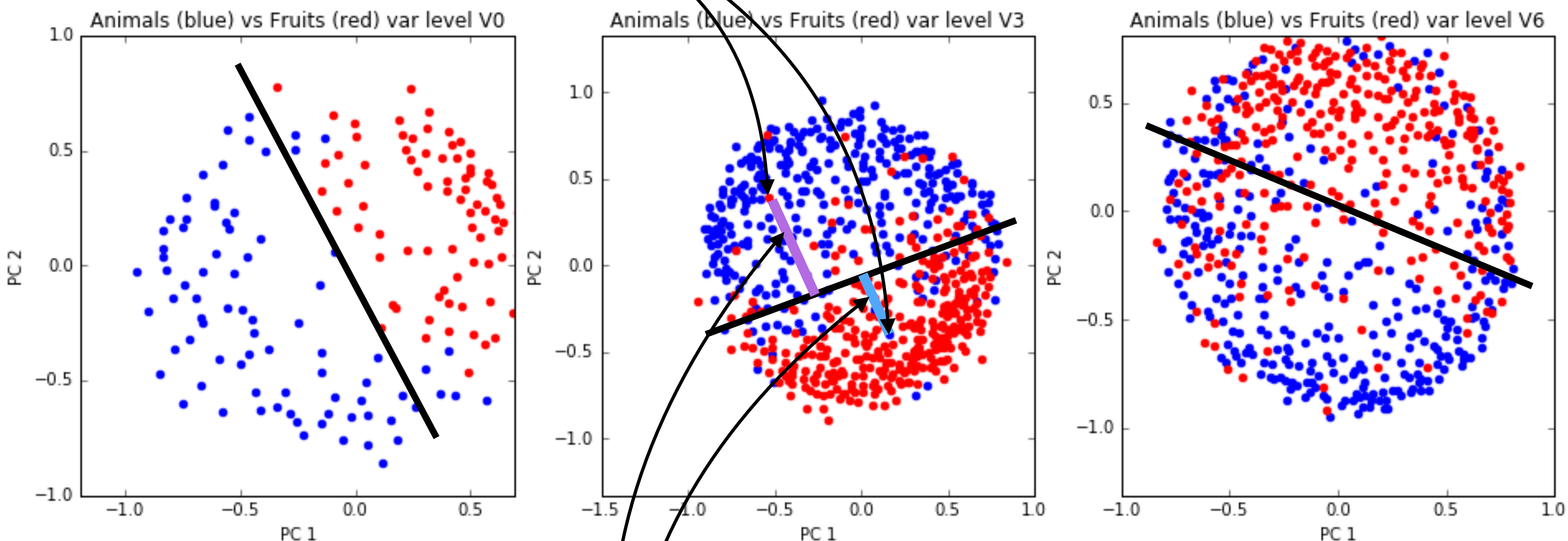

Find w's and b that minimize:

$$\max\left(0, 1 - \mathrm{red}_j * \left[\sum_i w_i \cdot neuron_i(s_j) + b\right]\right)$$

averaged over training stimuli s_j

# Support Vector Machines

the formula below penalizes incorrect answers by amount proportional to distance from the boundary

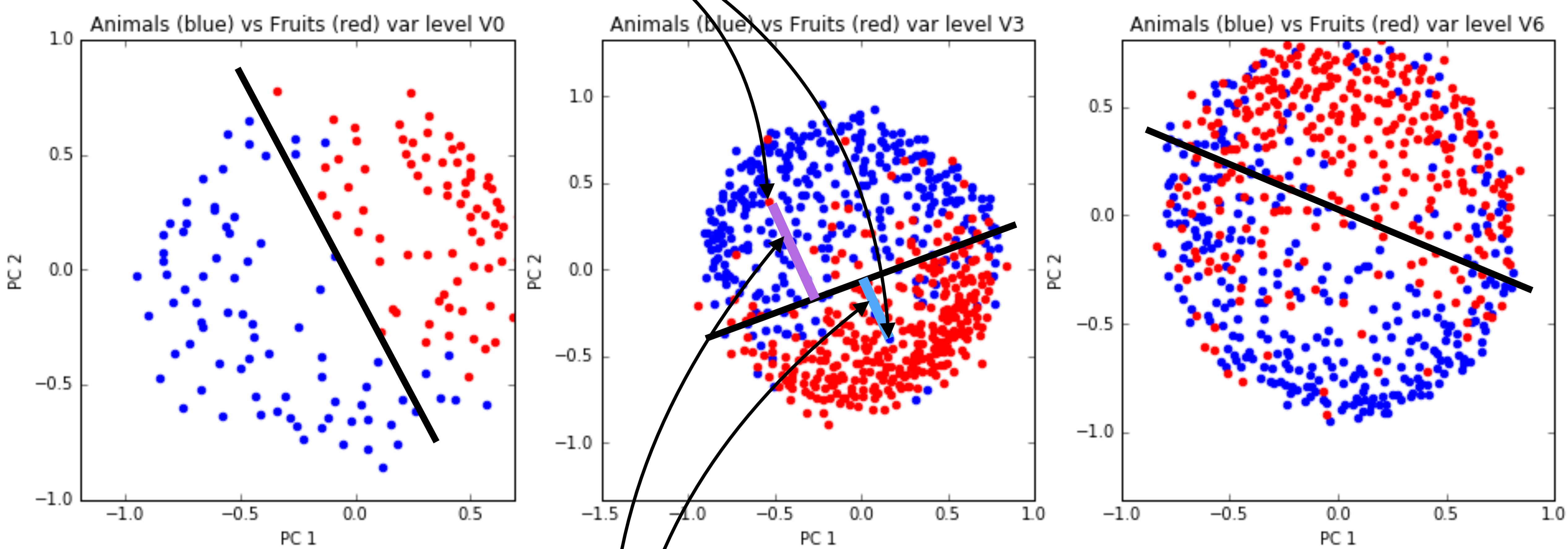



Find w's and b that minimize:

$$\max\left(0, 1 - \text{red}_j * \left[\sum_i w_i \cdot neuron_i(s_j) + b\right]\right)$$

# Support Vector Machines

the formula below penalizes incorrect answers by amount proportional to distance from the boundary




Find w's and b that minimize:

$$\max\left(0, 1 - \mathrm{red}_j * \left[\sum_i w_i \cdot neuron_i(s_j) + b\right]\right)$$

but doesn't reward correct answers

# Support Vector Machines

The general form of this idea is called the "hinge loss":

$$\text{hinge\_loss}(p, a) = max(0, 1 - p \cdot a)$$

where $a$ = actual class (binary true value)

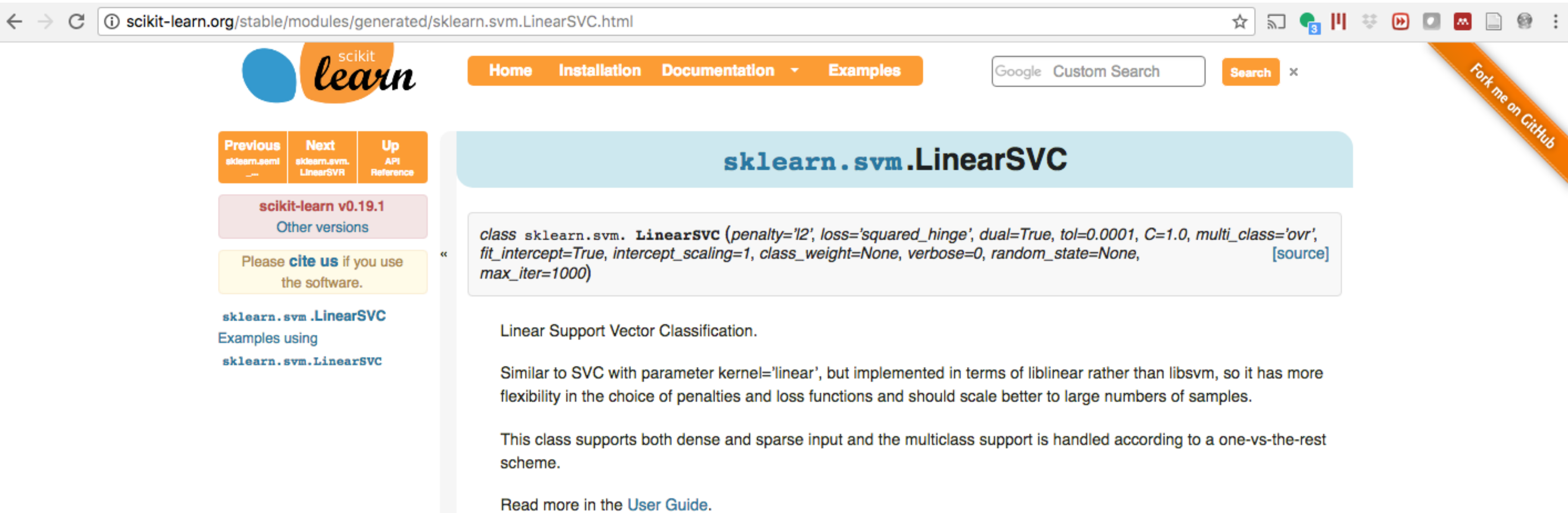
$p$ = predicted (continuous output of decision function)

This loss function is fine too:

$$\text{squared\_hinge\_loss}(p, a) = [max(0, 1 - p \cdot a)]^2$$

since it has the same minima as the hinge_loss. …Turns out the square is often easier to optimize.

# Support Vector Machines



scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

scikit learn

Home Installation Documentation Examples

Google Custom Search Search

Fork me on GitHub

Previous | Next | Up

scikit-learn v0.19.1

Other versions

Please **cite us** if you use the software.

sklearn.svm.LinearSVC

Examples using sklearn.svm.LinearSVC

## sklearn.svm.LinearSVC

*class* sklearn.svm. **LinearSVC** (*penalty='l2', loss='squared_hinge', dual=True, tol=0.0001, C=1.0, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1000*) [source]

Linear Support Vector Classification.

Similar to SVC with parameter kernel='linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.

This class supports both dense and sparse input and the multiclass support is handled according to a one-vs-the-rest scheme.

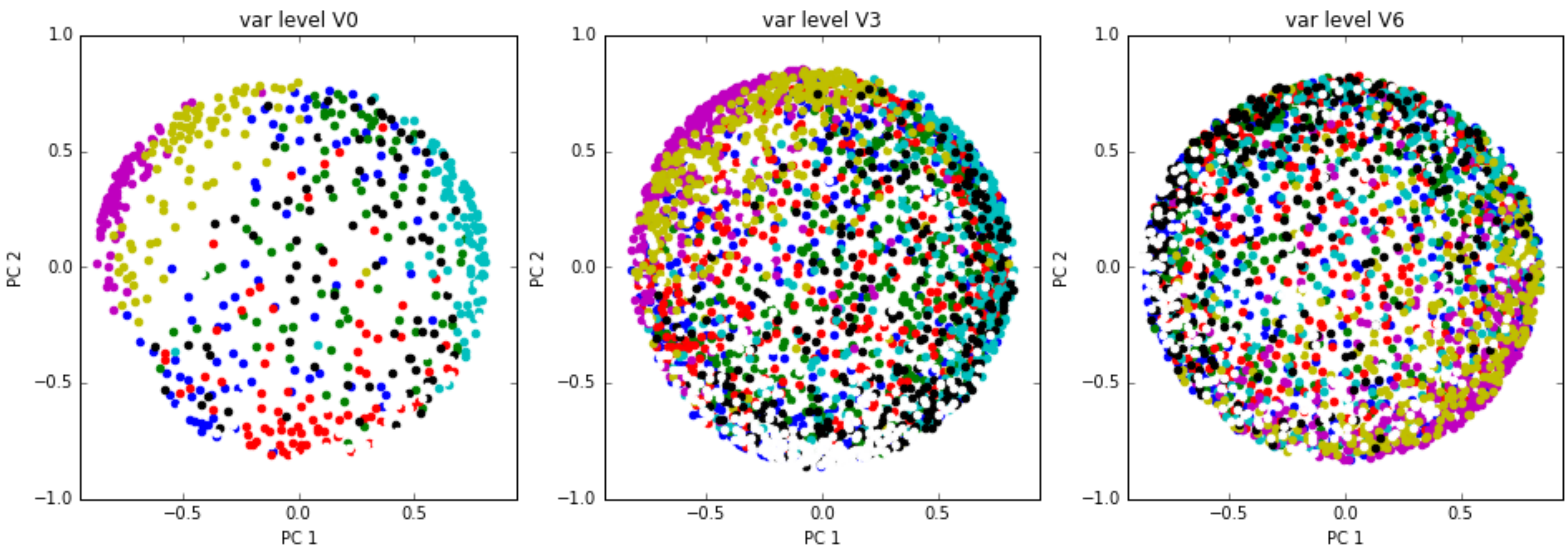
Read more in the User Guide.

# Support Vector Machines

*[IPYNB: Binary SVM]*

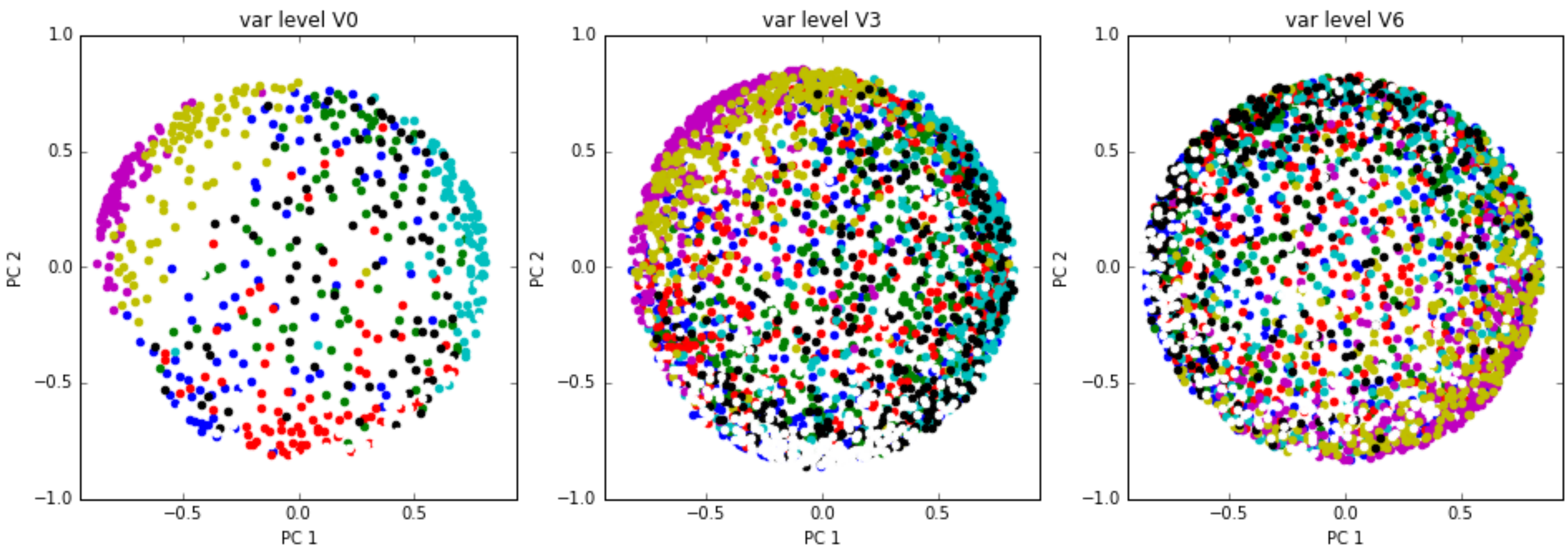
# Support Vector Machines — Multiclass

How do we handle multi-class classification with SVMs?



# Support Vector Machines — Multiclass

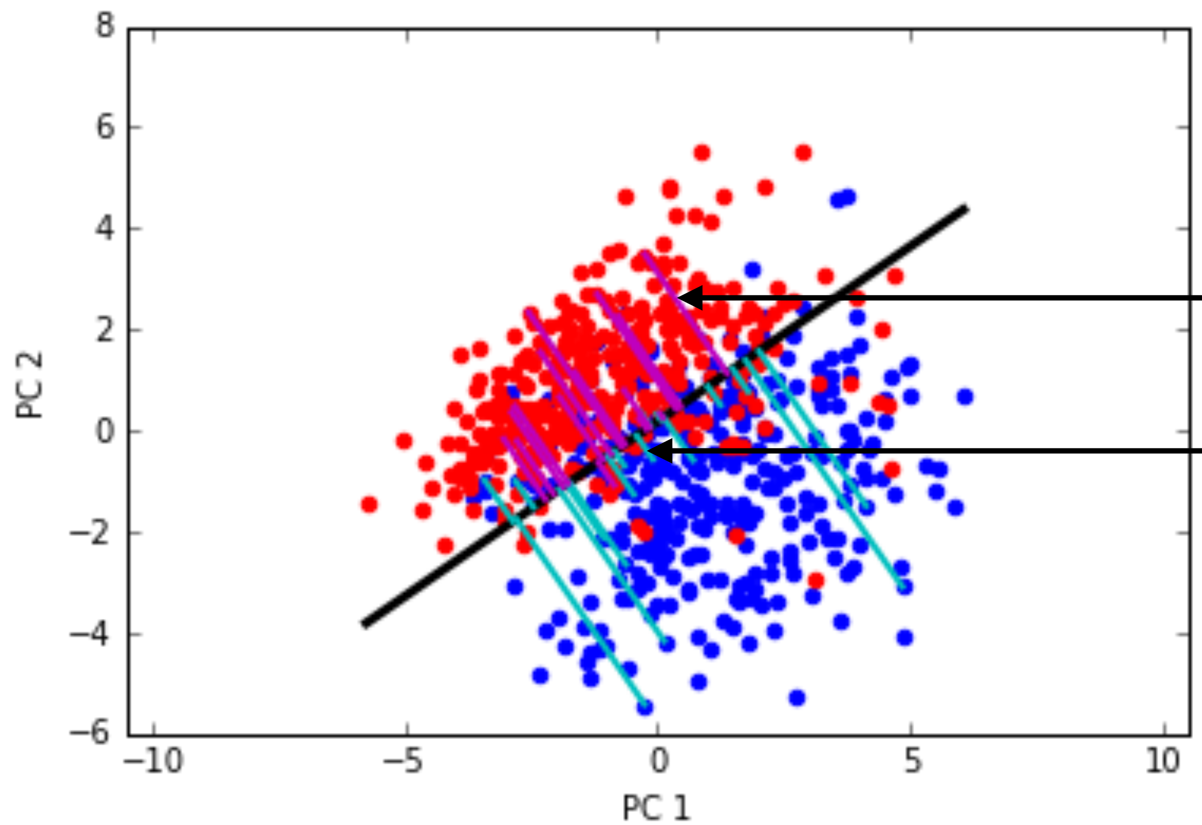
How do we handle multi-class classification with SVMs?




Do K binary one-vs-all classification problems and pick the one with the most confidence.

# Support Vector Machines — Multiclass

We measure confidence with the margin:




Classifier is confident it's red

Classifier is less confident it's blue

$$\mathrm{margins}(s_j) = \sum_i w_i \cdot \mathrm{neuron}_i(s_j) + b$$

…. the continuous value before taking the (discrete) sign.

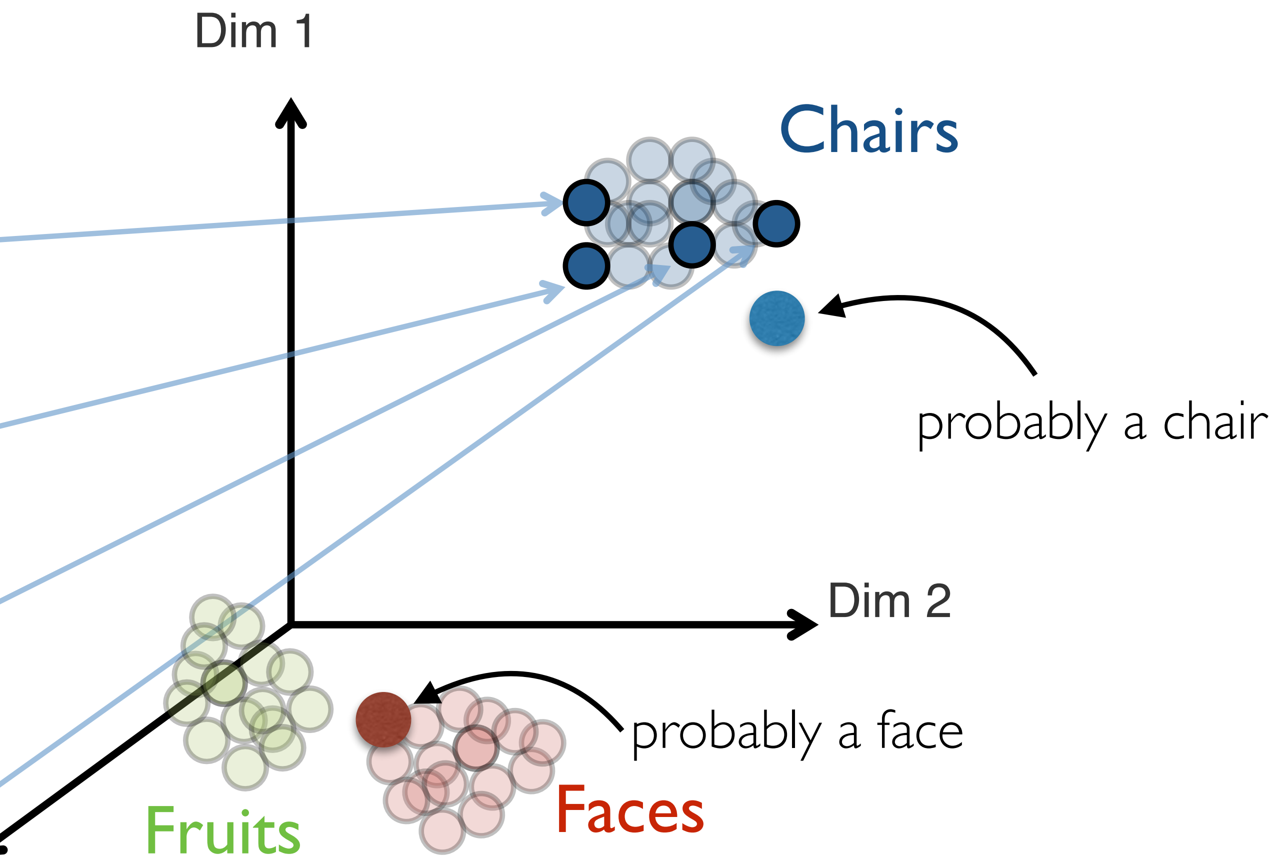
One-Vs-All Multiclass procedure:

```
k = num_categories

for i=0:k-1:
    train class i-vs-not-i binary SVMi

for new test condition (stimulus) x:
    margini = SVMi.decision_function(x)

pick category with largest margin
```

# Support Vector Machines — Multiclass

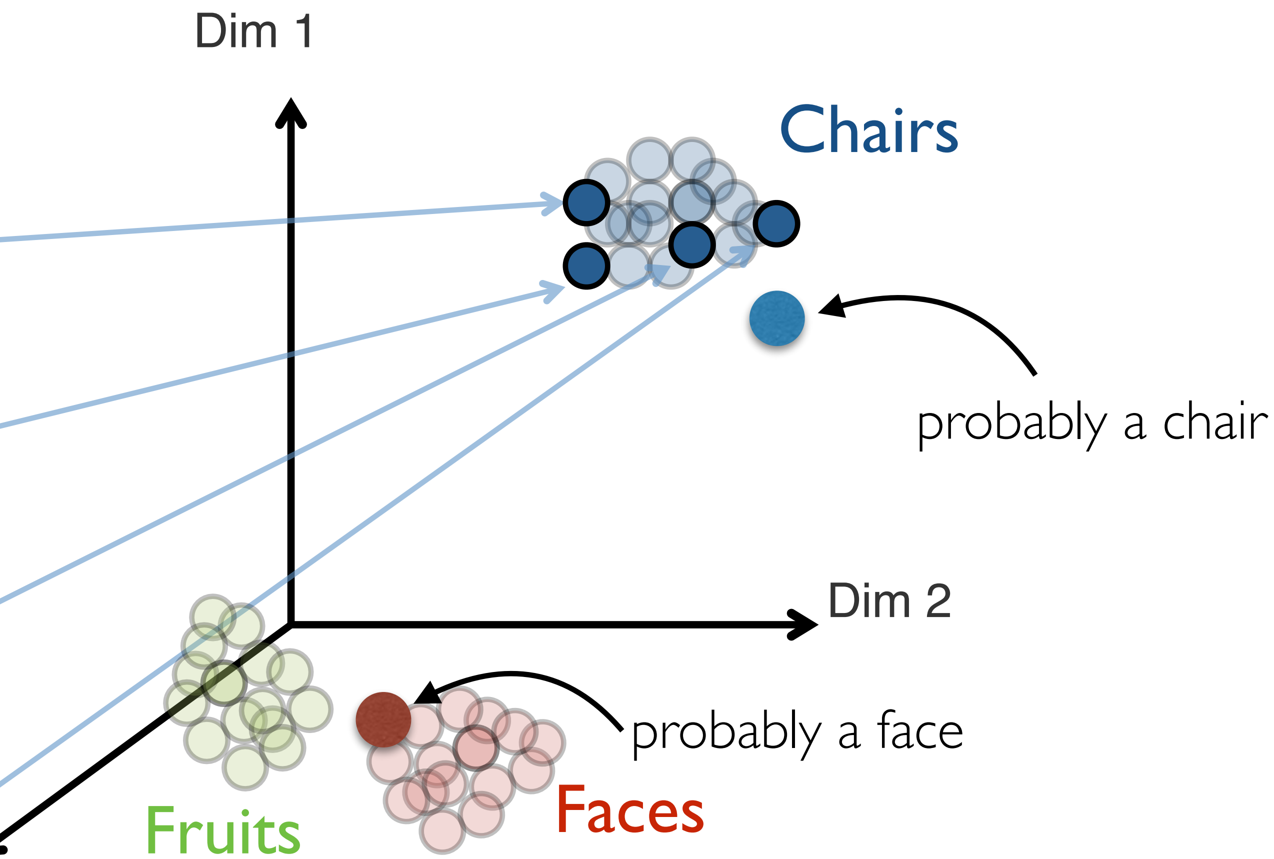
*[IPYNB: Multiclass Classification via One-vs-All]*

# Nearest Neighbor Classifiers




Idea of **minimum distance classifiers**: For any new example, pick class whose mean is closest in neural space.

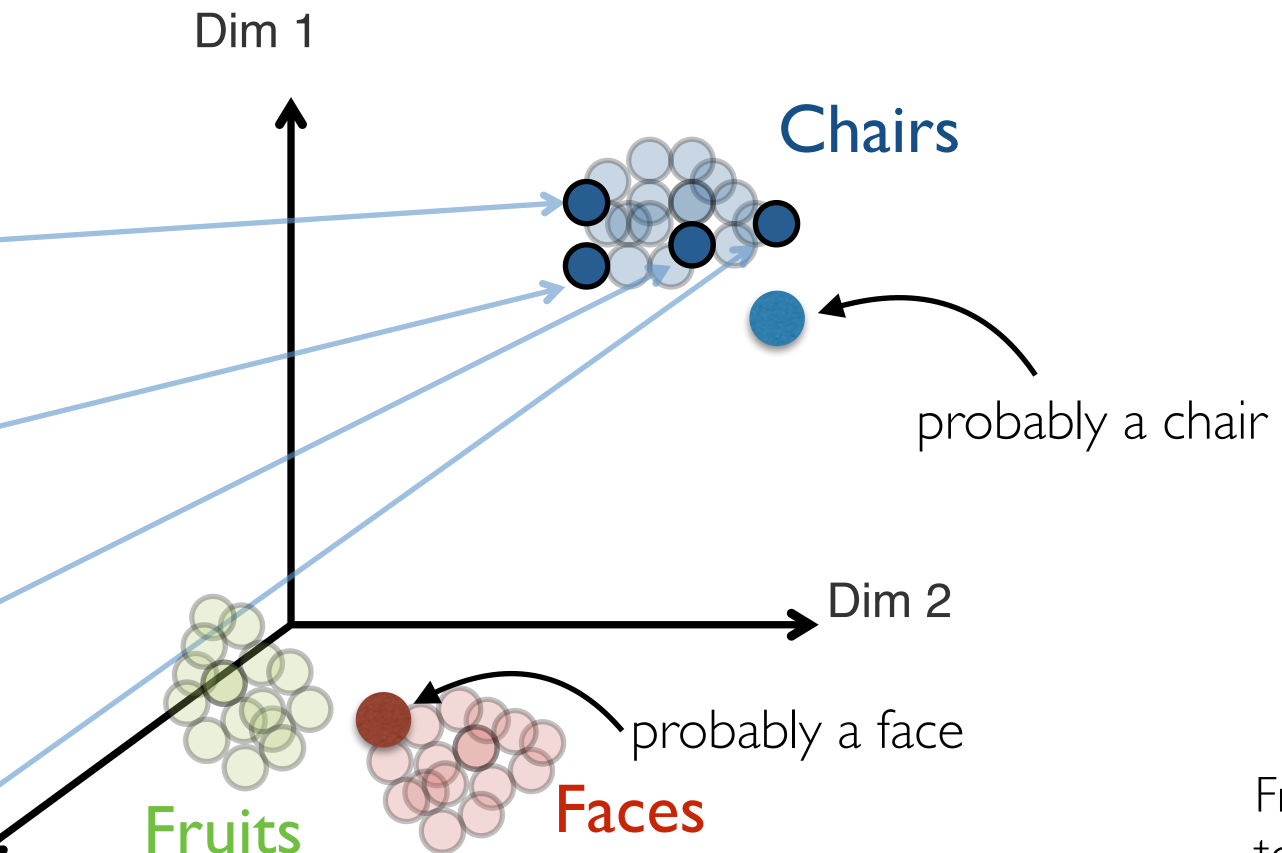
# Nearest Neighbor Classifiers




Idea of **minimum distance classifiers**:
For any new example, pick class whose mean is closest in feature space.

Idea of **nearest-neighbor (NN) classifiers**:
For any new example ***x***, look at class of closest neighbors to ***x*** and pick the one that appears most commonly

# Nearest Neighbor Classifiers

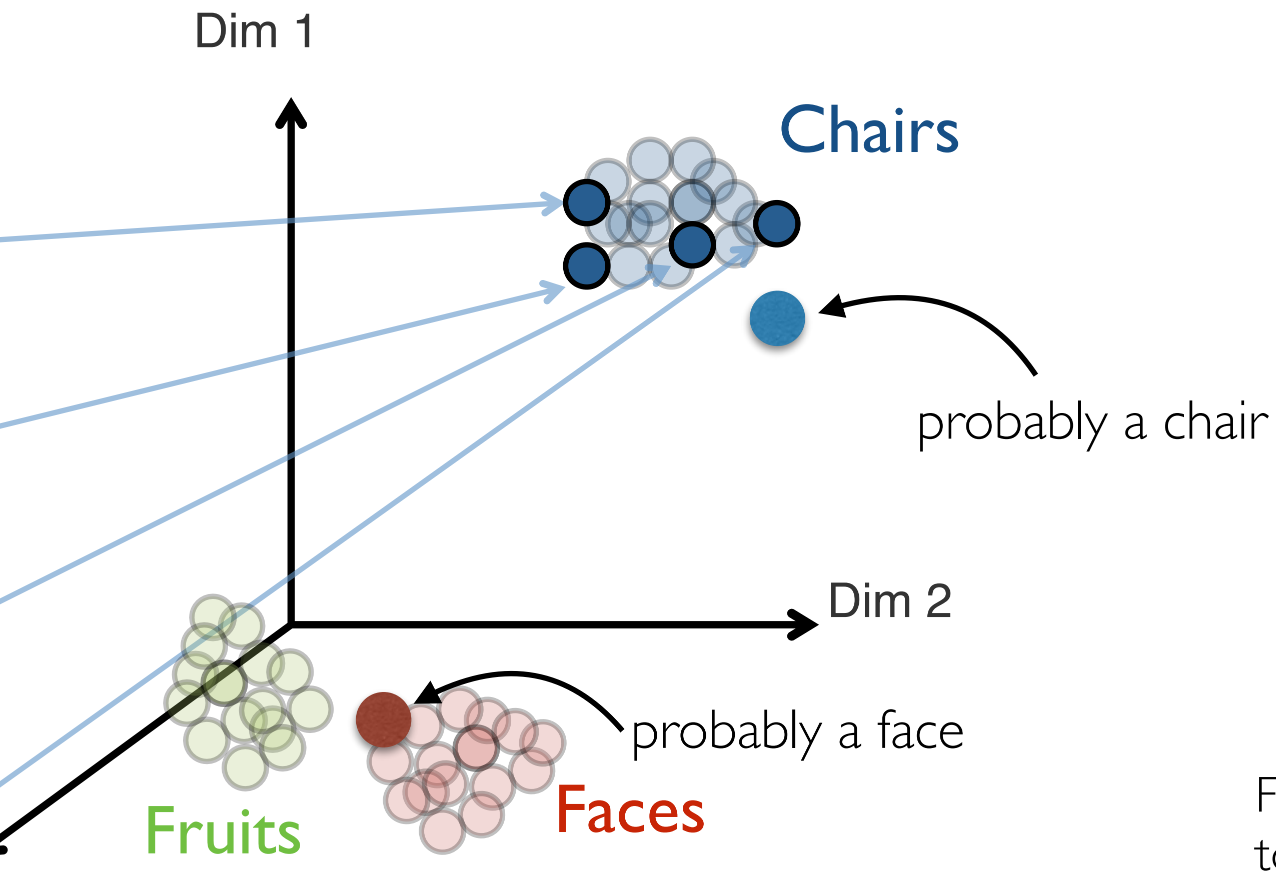



Idea of **minimum distance classifiers**: For any new example, pick class whose mean is closest in neural space.

Idea of **nearest-neighbor (NN) classifiers**: For any new example ***x***, look at class of closest neighbors to ***x*** and pick the one that appears most commonly

Free parameter: how many neighbors to consider. (needs to be cross-validated just like C for the SVM)

# Nearest Neighbor Classifiers




Idea of **minimum distance classifiers**: For any new example, pick class whose mean is closest in neural space.

Idea of **nearest-neighbor (NN) classifiers**: For any new example ***x***, look at class of closest neighbors to ***x*** and pick the one that appears most commonly

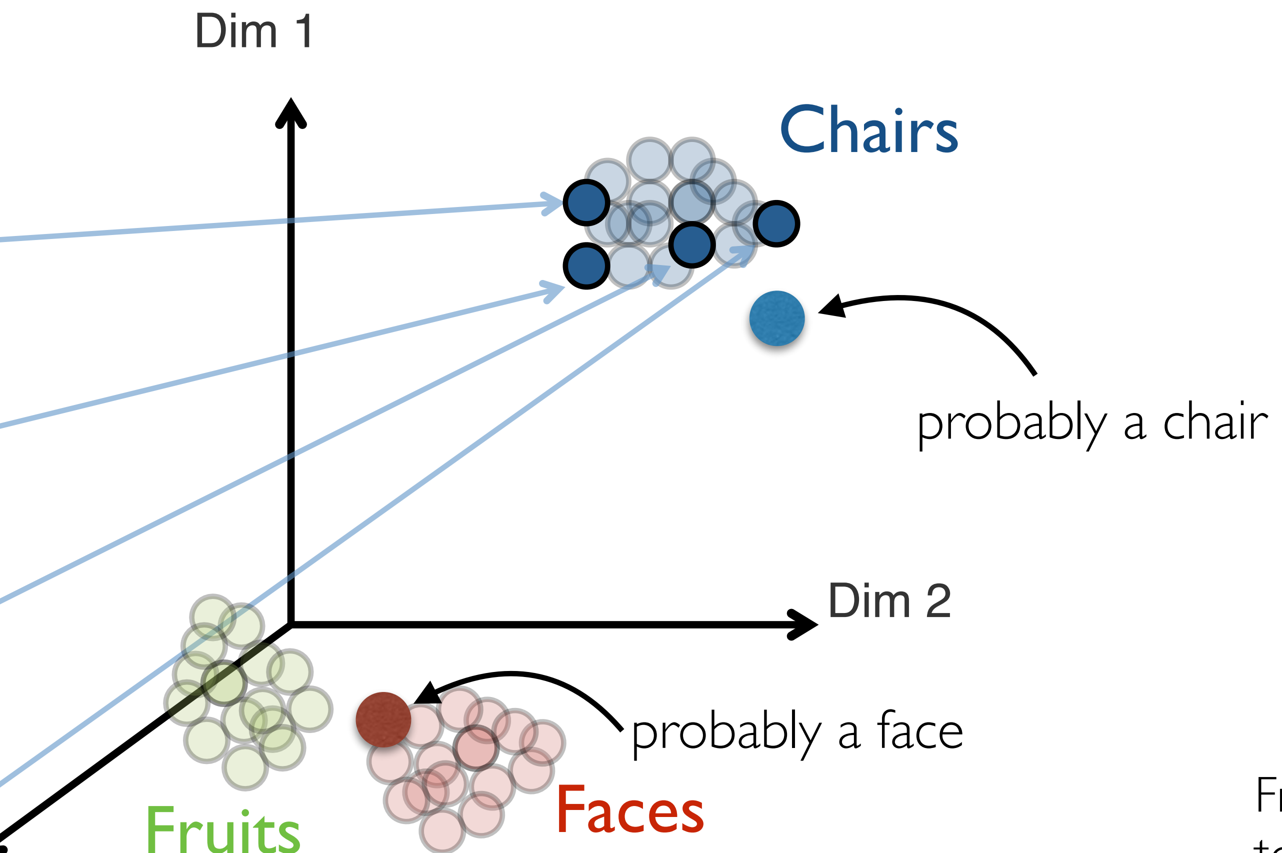
Free parameter: how many neighbors to consider. (needs to be cross-validated just like C for the SVM)

The NN classifier is like the MDC, except that the operations "come in the other order":

MDC : first take the mean (during training) then look at distances (during testing)

NNC : first look at distances (during testing) and then take the "mean" (also during testing)

# Nearest Neighbor Classifiers




Idea of **minimum distance classifiers**: For any new example, pick class whose mean is closest in neural space.

Idea of **nearest-neighbor (NN) classifiers**: For any new example ***x***, look at class of closest neighbors to ***x*** and pick the one that appears most commonly

Free parameter: how many neighbors to consider. (needs to be cross-validated just like C for the SVM)

The NN classifier is like the MDC, except that the operations "come in the other order":

MDC : first take the mean (during training) then look at distances (during testing)

NNC : first look at distances (during testing) and then take the "mode" (also during testing)

NNC is weird because you have to store the whole training set. And slow, because need to compute distances between ***x*** and all training examples …

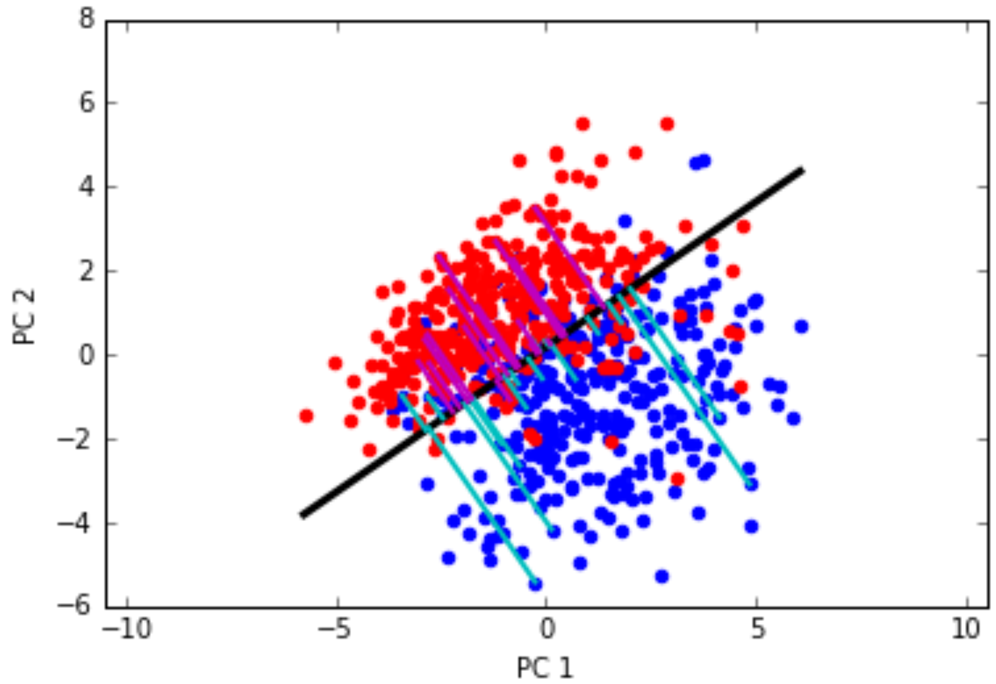
# Nearest Neighbor Classifiers

*[IPYNB:   Nearest Neighbor Classifiers]*

# Other Distances in the SVM Classifier

Recall out hyperplane-based classifiers are based on

$$\text{margins}(s_j) = \sum_i w_i \cdot \text{neuron}_i(s_j) + b$$

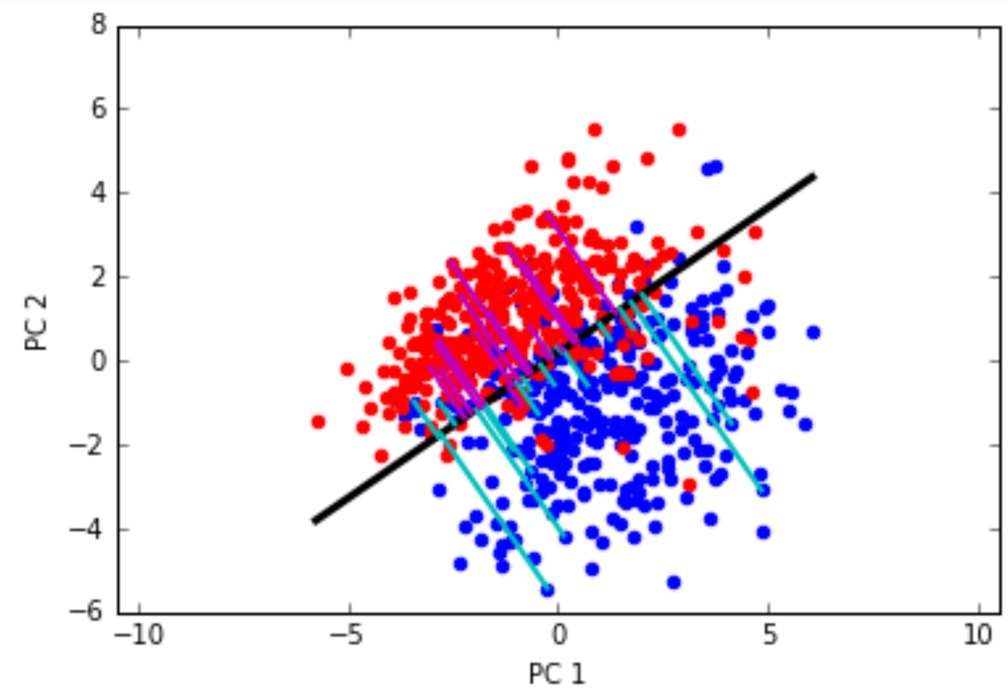


# Other Distances in the SVM Classifier

Recall out hyperplane-based classifiers are based on

$$\mathrm{margins}(s_j) = \sum_i w_i \cdot \mathrm{neuron}_i(s_j) + b$$

Results only depend on matrix of *pairwise similarities* (inverse distances) between training points (s_j's).

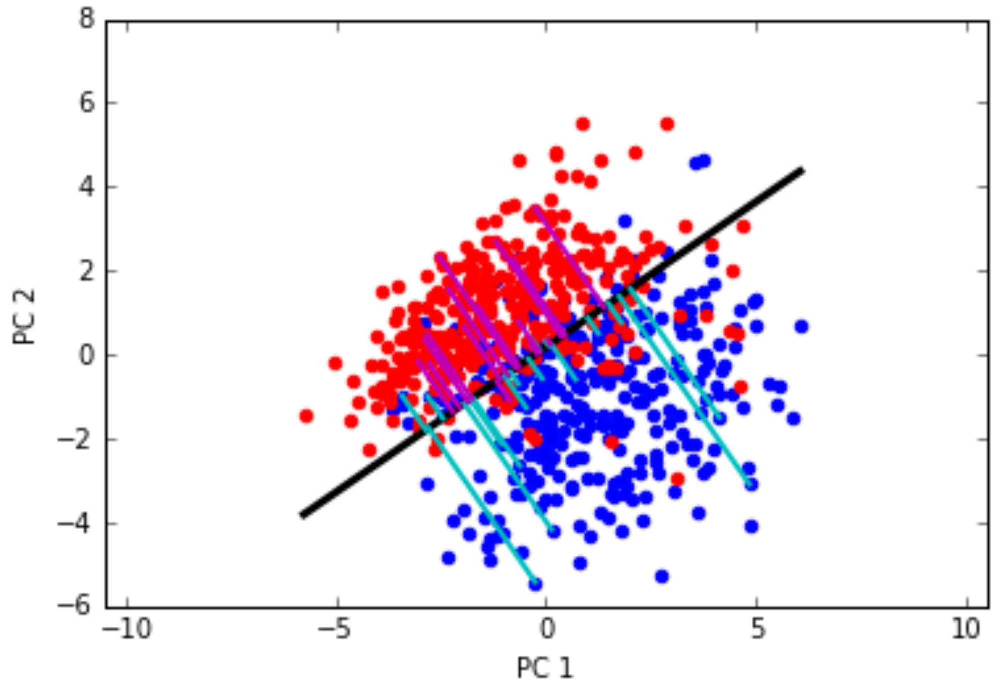


# Other Distances in the SVM Classifier

Recall out hyperplane-based classifiers are based on

$$\text{margins}(s_j) = \sum_i w_i \cdot \text{neuron}_i(s_j) + b$$

Results only depend on matrix of *pairwise similarities* (inverse distances) between training points (s_j's).*




Regular SVM uses euclidean similiarity.

Can actually use *any* similarity measure you want.

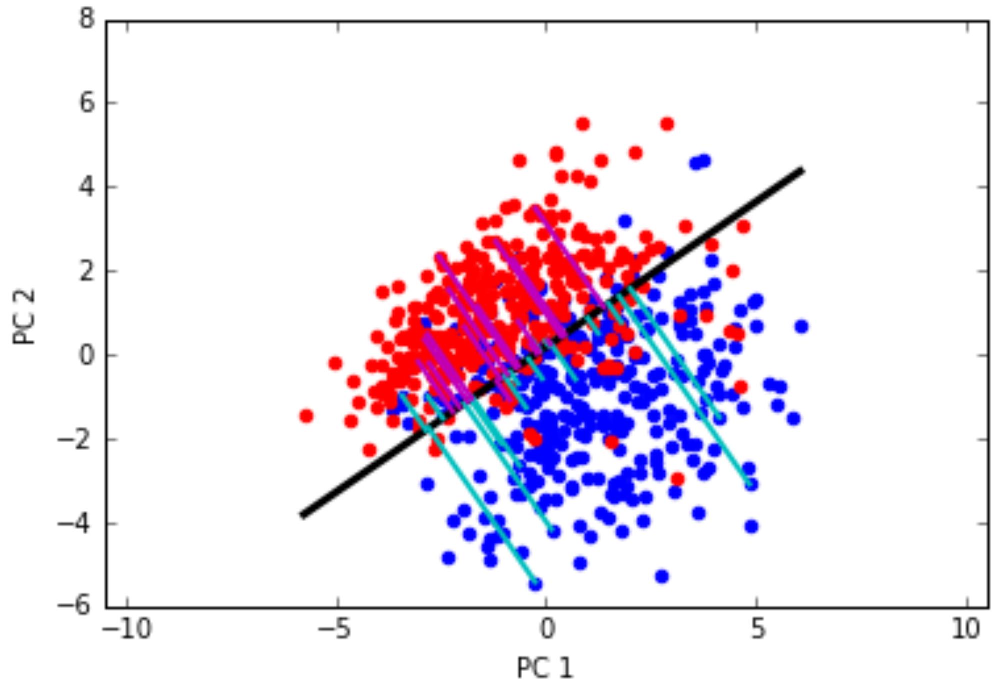
* matrix of pairwise similarities is sometimes called a "kernel" or a "gram matrix"

# Other Distances in the SVM Classifier

Recall out hyperplane-based classifiers are based on

$$\mathrm{margins}(s_j) = \sum_i w_i \cdot \mathrm{neuron}_i(s_j) + b$$

Results only depend on matrix of *pairwise similarities* (inverse distances) between training points (s_j's).*




Regular SVM uses euclidean similiarity.

`sklearn.svm.LinearSVC`

only works with regular euclidean similarity

Can actually use *any* similarity measure you want.

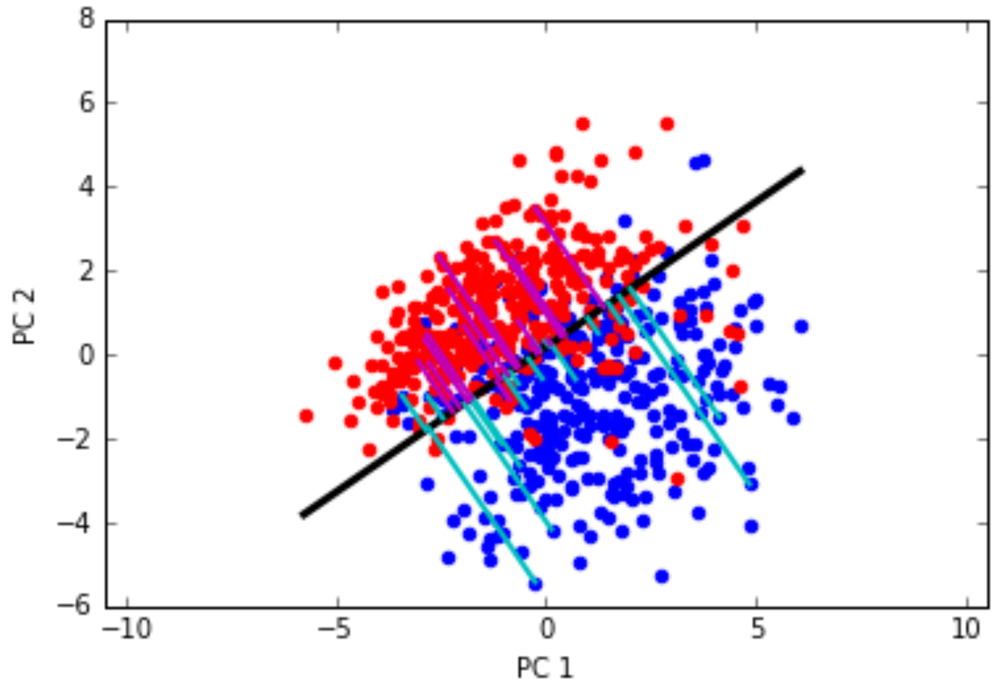
* matrix of pairwise similarities is sometimes called a "kernel" or a "gram matrix"

# Other Distances in the SVM Classifier

Recall out hyperplane-based classifiers are based on

$$\text{margins}(s_j) = \sum_i w_i \cdot \text{neuron}_i(s_j) + b$$

Results only depend on matrix of *pairwise similarities* (inverse distances) between training points (s_j's).*




Regular SVM uses euclidean similiarity.

Can actually use *any* similarity measure you want.

`sklearn.svm.LinearSVC` → `sklearn.svm.SVC`

only works with regular euclidean similarity

you can input any similarity function, even one that's *nonlinear* fn of its inputs

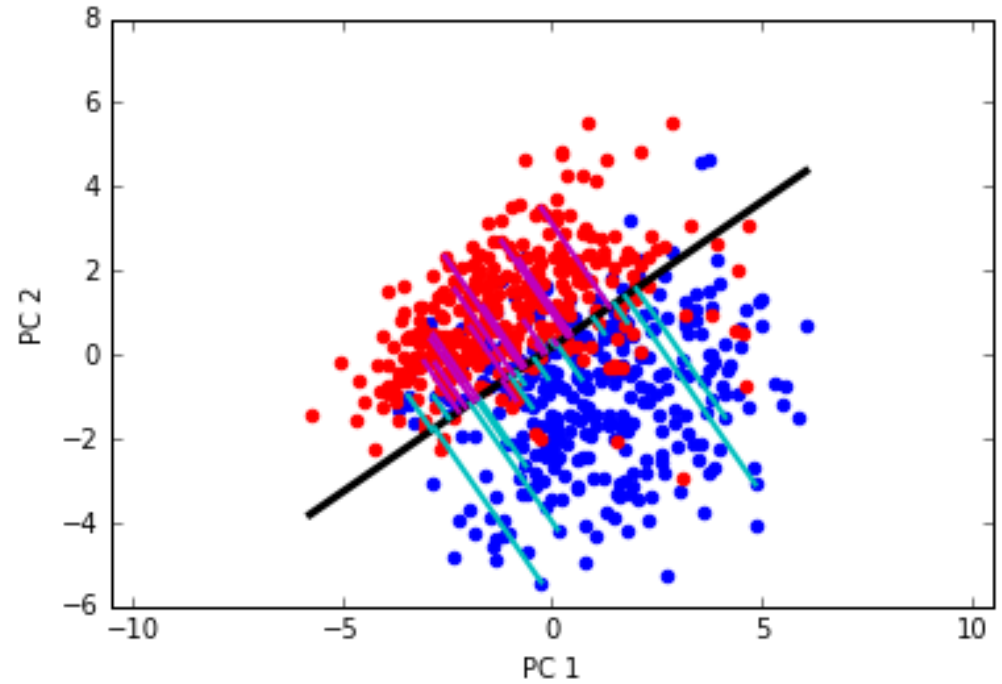
* matrix of pairwise similarities is sometimes called a "kernel" or a "gram matrix"

# Other Distances in the SVM Classifier

Recall out hyperplane-based classifiers are based on

$$\text{margins}(s_j) = \sum_i w_i \cdot \text{neuron}_i(s_j) + b$$

Results only depend on matrix of *pairwise similarities* (inverse distances) between training points (s_j's).*




Regular SVM uses euclidean similiarity.

Can actually use *any* similarity measure you want.

**`sklearn.svm.LinearSVC`** → **`sklearn.svm.SVC`**

only works with regular euclidean similarity

you can input any similarity function, even one that's *nonlinear* fn of its inputs

Most common is the "radial basis function" (RBF) similarity:

$$\mathbf{RBF}_\gamma(s_j, s_k) = e^{-\gamma||s_j - s_k||^2}$$

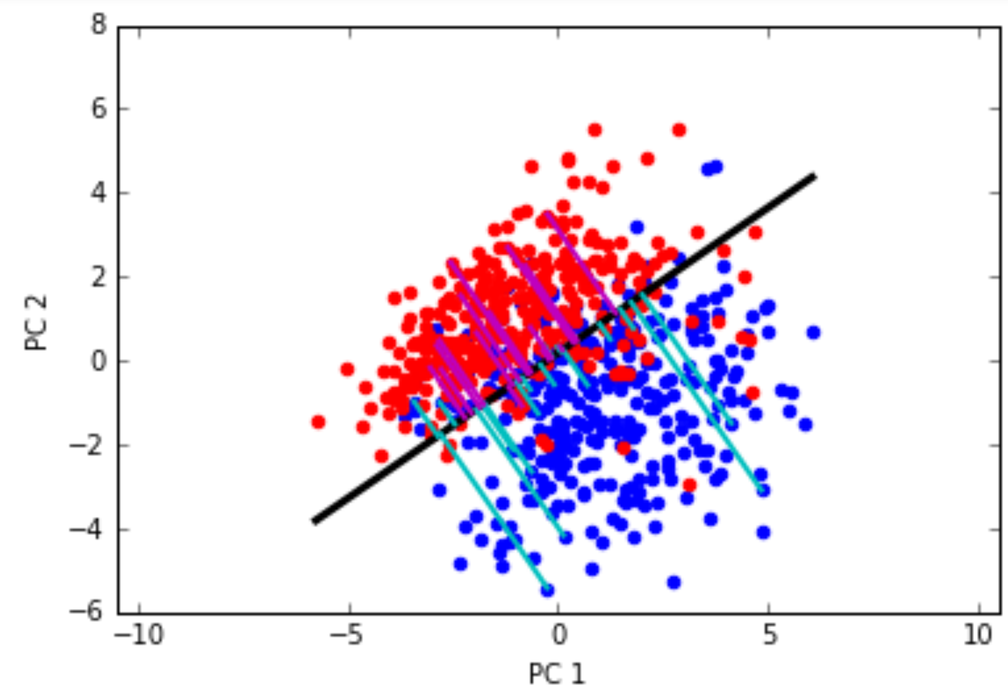
* matrix of pairwise similarities is sometimes called a "kernel" or a "gram matrix"

# Other Distances in the SVM Classifier

Recall out hyperplane-based classifiers are based on

$$\text{margins}(s_j) = \sum_i w_i \cdot \text{neuron}_i(s_j) + b$$

Results only depend on matrix of *pairwise similarities* (inverse distances) between training points (s_j's).*




Regular SVM uses euclidean similiarity.

Can actually use *any* similarity measure you want.

| | | |
|---|---|---|
| `sklearn.svm.LinearSVC` | → | `sklearn.svm.SVC` |
| only works with regular euclidean similarity | | you can input any similarity function, even one that's *nonlinear* fn of its inputs |

Most common is the "radial basis function" (RBF) similarity:

$$\mathbf{RBF}_\gamma(s_j, s_k) = e^{-\boxed{\gamma}||s_j - s_k||^2}$$

free parameter "length scale" — if you use RBF, needs to be cross-validated, just like C for SVM

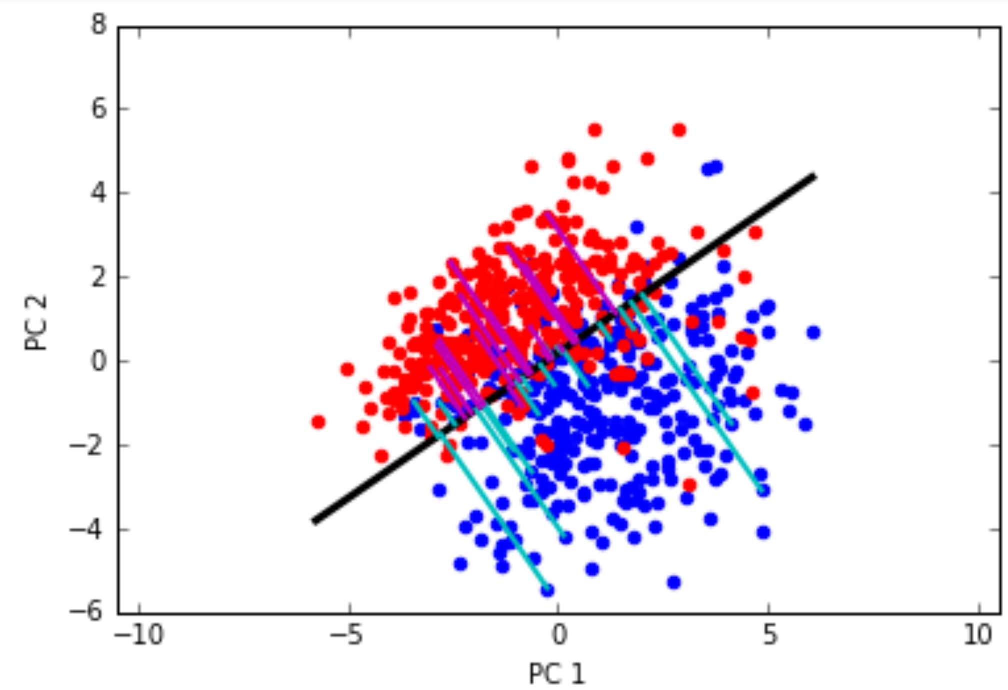
* matrix of pairwise similarities is sometimes called a "kernel" or a "gram matrix"

# Other Distances in the SVM Classifier

Recall out hyperplane-based classifiers are based on

$$\mathrm{margins}(s_j) = \sum_i w_i \cdot \mathrm{neuron}_i(s_j) + b$$

Results only depend on matrix of *pairwise similarities* (inverse distances) between training points (s_j's).*




Regular SVM uses euclidean similarity.

Can actually use *any* similarity measure you want.

`sklearn.svm.LinearSVC` → `sklearn.svm.SVC`

only works with regular euclidean similarity

you can input any similarity function, even one that's *nonlinear* fn of its inputs

NB: This whole idea of nonlinear distance functions with SVM used to be a big deal — was only way of using nonlinearity before deep nets. RBFs often not better than linear SVMs though. (Turns out you really need to *learn* the nonlinearity). So I mostly stick with linear kernels ("usual euclidean distance") when doing SVMs.

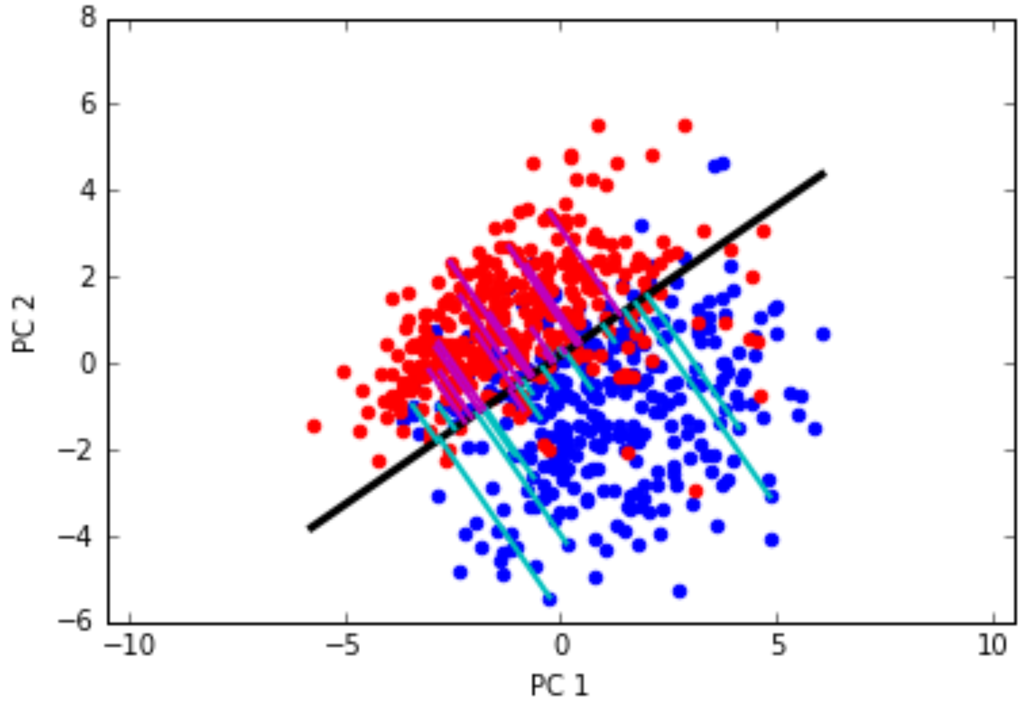
* matrix of pairwise similarities is sometimes called a "kernel" or a "gram matrix"

# Logistic Regression

Called "regression" but actually a hyperplane classification method.

# Logistic Regression

Called "regression" but actually a hyperplane classification method.



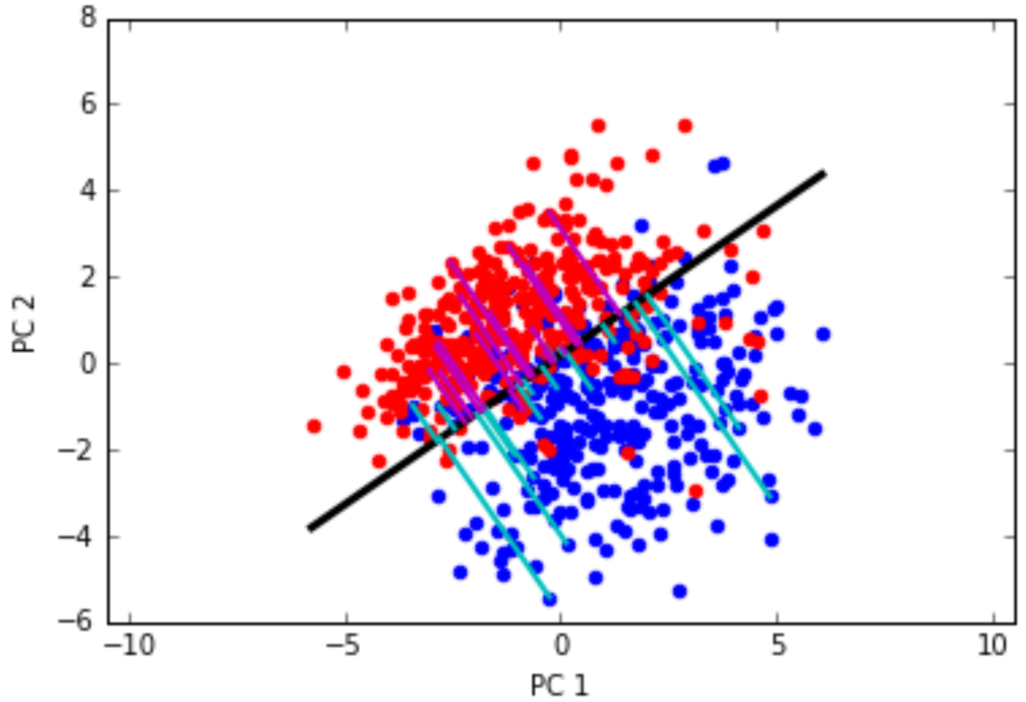

Recall, hinge loss of the SVM:

$$\text{loss} = \max\left(0, 1 - \text{red}_j * \left[\sum_i w_i \cdot neuron_i(s_j) + b\right]\right)$$

$$\text{margins}(s_j) = \sum_i w_i \cdot \text{neuron}_i(s_j) + b$$

# Logistic Regression

Called "regression" but actually a hyperplane classification method.




Recall, hinge loss of the SVM:

$$\text{loss} = \max\left(0, 1 - \text{red}_j * \left[\sum_i w_i \cdot neuron_i(s_j) + b\right]\right)$$

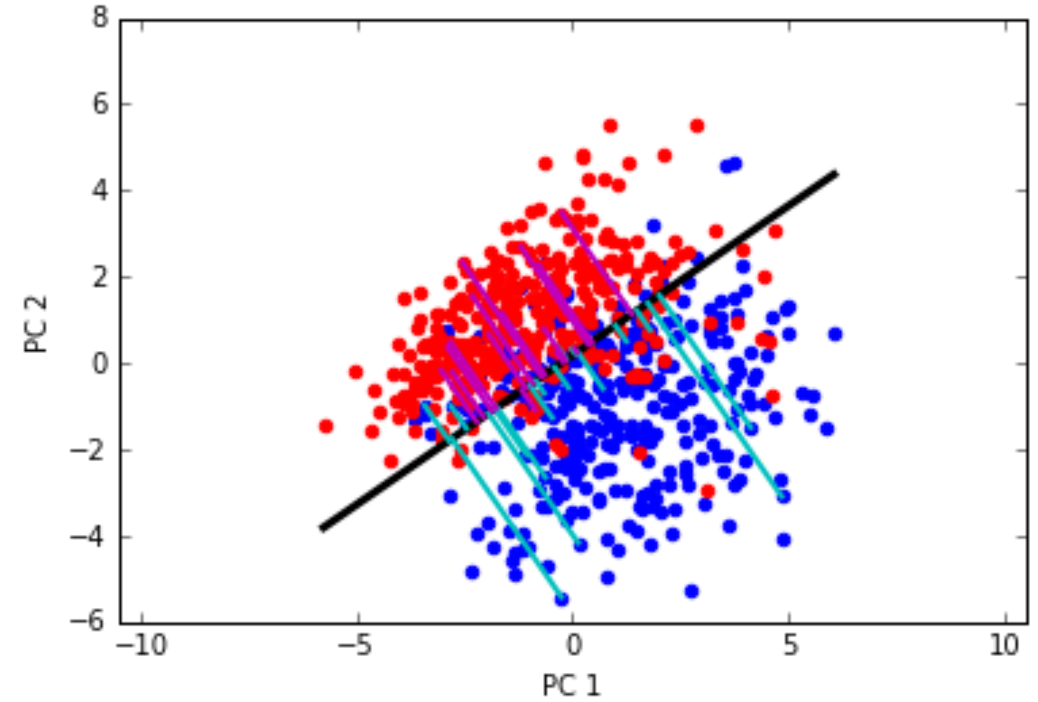
$$\text{margins}(s_j) = \sum_i w_i \cdot \text{neuron}_i(s_j) + b$$

Informally, the idea for the Logistic Regression is:

Let's pretend that the **margins** for a given sample for each possible class are like **probabilities** that the sample is of that class

# Logistic Regression

Called "regression" but actually a hyperplane classification method.




Recall, hinge loss of the SVM:

$$\text{loss} = \max\left(0, 1 - \text{red}_j * \left[\sum_i w_i \cdot neuron_i(s_j) + b\right]\right)$$

$$\text{margins}(s_j) = \sum_i w_i \cdot \text{neuron}_i(s_j) + b$$

Informally, the idea for the Logistic Regression is:

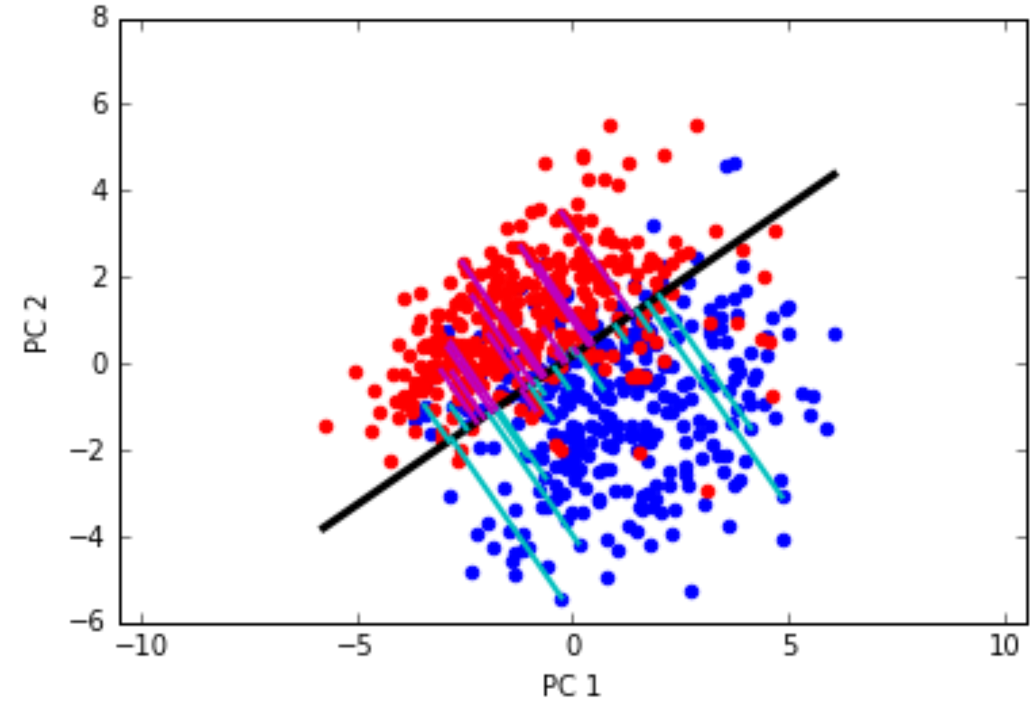
Let's pretend that the **margins** for a given sample for each possible class are like **probabilities** that the sample is of that class

Then, the loss function is:

Let's find weights ***w*** and bias ***b*** such that the log-likelihood of the class given the data is maximized (on the training data).

# Logistic Regression

Called "regression" but actually a hyperplane classification method.




Recall, hinge loss of the SVM:

$$\text{loss} = \max\left(0, 1 - \text{red}_j * \left[\sum_i w_i \cdot neuron_i(s_j) + b\right]\right)$$

$$\text{margins}(s_j) = \sum_i w_i \cdot \text{neuron}_i(s_j) + b$$

Informally, the idea for the Logistic Regression is:

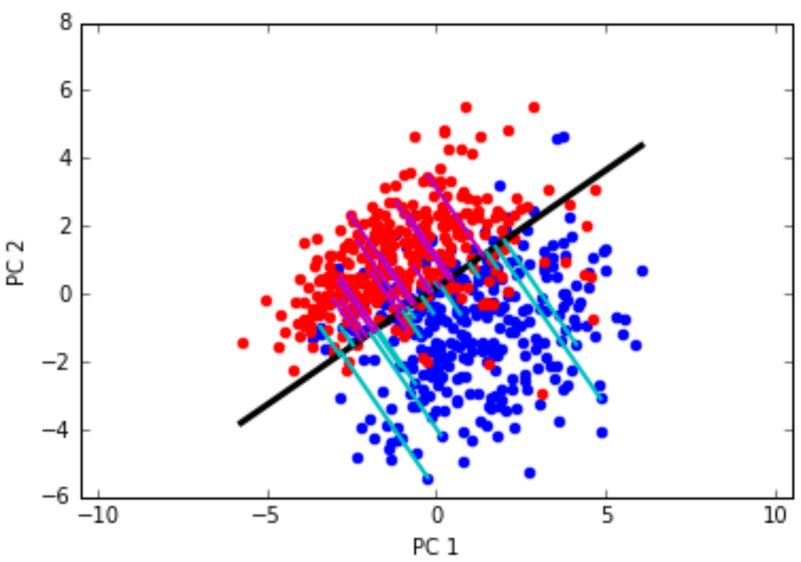
Let's pretend that the **margins** for a given sample for each possible class are like **probabilities** that the sample is of that class

Then, the loss function is:

Let's find weights ***w*** and bias ***b*** such that the **negative** log-likelihood of the class given the data is **minimized** (on the training data).

# Logistic Regression

How do we do this formally? Well, the margins:



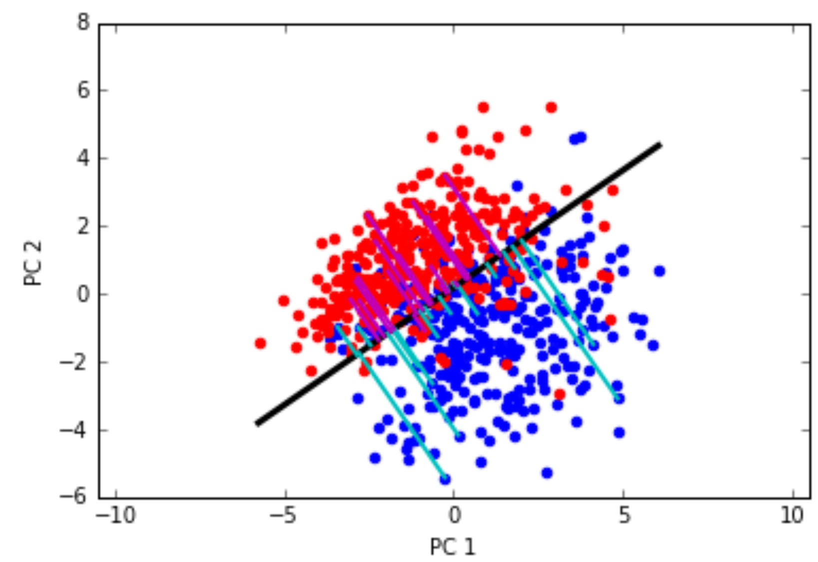

$$\mathrm{margins}(s_j) = \sum_i w_i \cdot \mathrm{neuron}_i(s_j) + b$$

… are signed values that can be negative or positive.

# Logistic Regression

How do we do this formally? Well, the margins:

$$\text{margins}(s_j) = \sum_i w_i \cdot \text{neuron}_i(s_j) + b$$



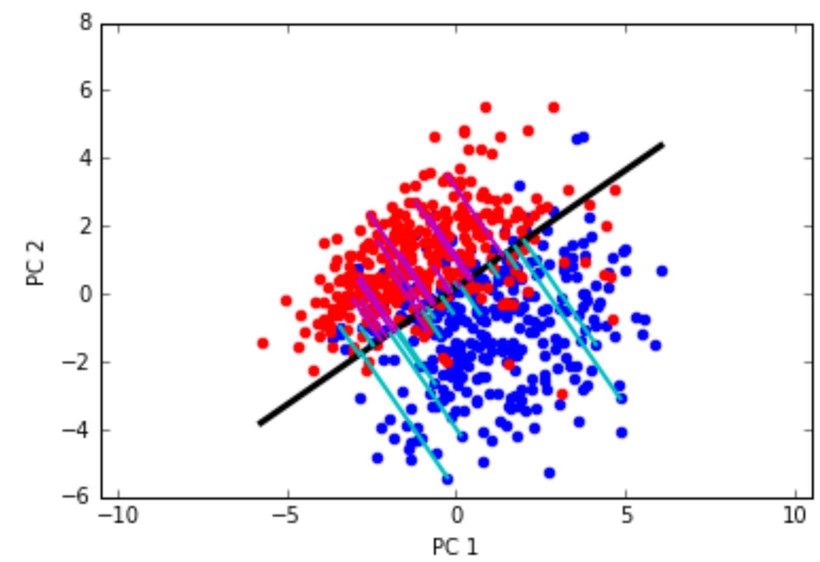

… are signed values that can be negative or positive.

What function maps to the correct range for a probability? [0, 1]

# Logistic Regression

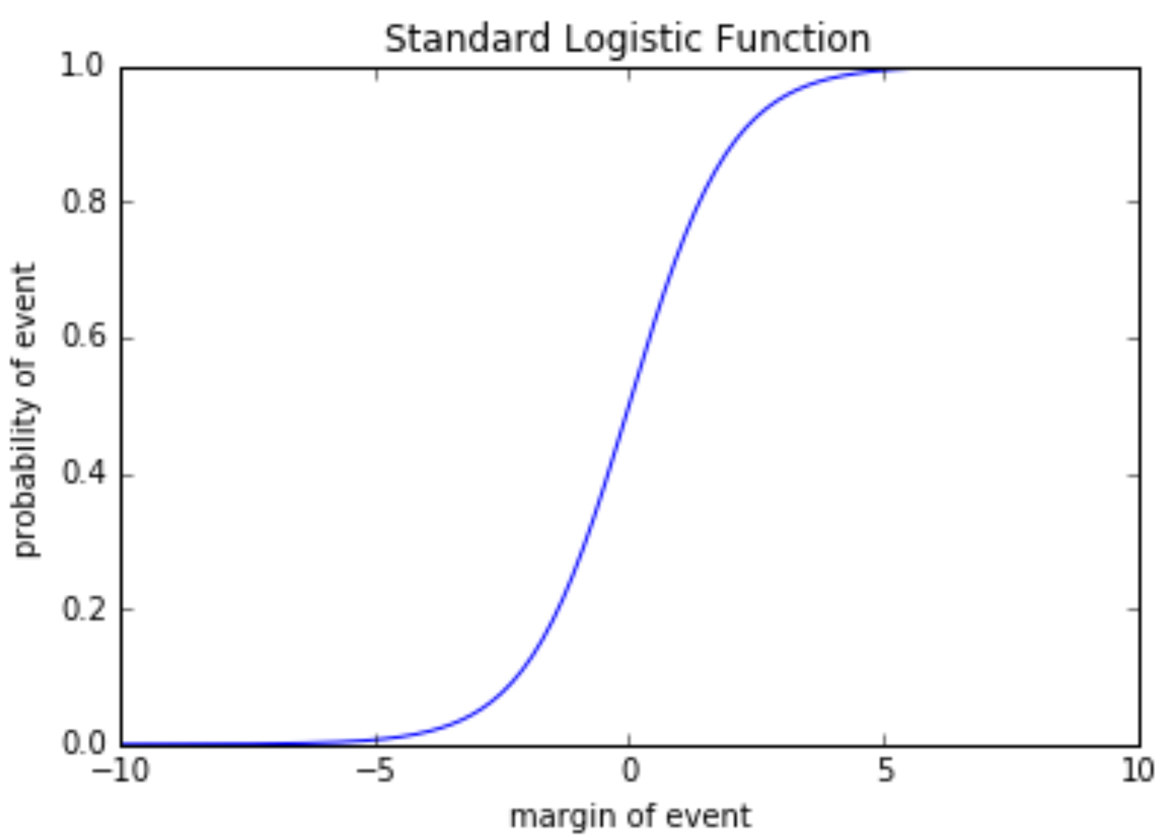
How do we do this formally? Well, the margins:

$$\text{margins}(s_j) = \sum_i w_i \cdot \text{neuron}_i(s_j) + b$$




… are signed values that can be negative or positive.

What function maps to the correct range for a probability? [0, 1]




$$\text{Logistic}(x) = \frac{1}{1+e^{-x}}$$

$$x \to -\infty \Rightarrow \text{Logistic}(x) \to 0$$

$$x \to \infty \Rightarrow \text{Logistic}(x) \to 1$$

*(Formally: Logistic fn: bernoulli prior on classes)*

# Logistic Regression

OK, let's have a look at the likelihood of a class observation given data in this scheme:

$$Pr(\text{red}_j = 1 | s_j) = \text{Logistic}(\text{Margins}(s_j)) \quad \text{red}_j = \begin{cases} 1 & \text{if } s_j \text{ red} \\ 0 & \text{if } s_j \text{ blue} \end{cases}$$

# Logistic Regression

OK, let's have a look at the likelihood of a class observation given data in this scheme:

$$Pr(\text{red}_j = 1 | s_j) = \text{Logistic}(\text{Margins}(s_j)) \quad \text{red}_j = \begin{cases} 1 & \text{if } s_j \text{ red} \\ 0 & \text{if } s_j \text{ blue} \end{cases}$$

thus

$$Pr(\text{red}_j = 0 | s_j)$$

# Logistic Regression

OK, let's have a look at the likelihood of a class observation given data in this scheme:

$$Pr(\text{red}_j = 1 | s_j) = \text{Logistic}(\text{Margins}(s_j)) \quad \text{red}_j = \begin{cases} 1 & \text{if } s_j \text{ red} \\ 0 & \text{if } s_j \text{ blue} \end{cases}$$

thus

$$Pr(\text{red}_j = 0 | s_j) = 1 - \text{Logistic}(\text{Margins}(s_j))$$

# Logistic Regression

OK, let's have a look at the likelihood of a class observation given data in this scheme:

$$Pr(\text{red}_j = 1 | s_j) = \text{Logistic}(\text{Margins}(s_j)) \quad \text{red}_j = \begin{cases} 1 & \text{if } s_j \text{ red} \\ 0 & \text{if } s_j \text{ blue} \end{cases}$$

thus

$$Pr(\text{red}_j = 0 | s_j) = 1 - \text{Logistic}(\text{Margins}(s_j))$$

thus

$$Pr(\text{red}_j | s_j) = \text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}$$

$m_j$ = margin for stim j

# Logistic Regression

OK, let's have a look at the likelihood of a class observation given data in this scheme:

$$Pr(\text{red}_j = 1 | s_j) = \text{Logistic}(\text{Margins}(s_j)) \quad \text{red}_j = \begin{cases} 1 & \text{if } s_j \text{ red} \\ 0 & \text{if } s_j \text{ blue} \end{cases}$$

thus

$$Pr(\text{red}_j = 0 | s_j) = 1 - \text{Logistic}(\text{Margins}(s_j))$$

thus

$$Pr(\text{red}_j | s_j) = \text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}$$

$m_j$ = margin for stim j

thus over all training data

$$Pr(\mathbf{red} | \mathbf{s}) =$$

# Logistic Regression

OK, let's have a look at the likelihood of a class observation given data in this scheme:

$$Pr(\mathrm{red}_j = 1 | s_j) = \mathrm{Logistic}(\mathrm{Margins}(s_j)) \quad \mathrm{red}_j = \begin{cases} 1 & \text{if } s_j \text{ red} \\ 0 & \text{if } s_j \text{ blue} \end{cases}$$

thus

$$Pr(\mathrm{red}_j = 0 | s_j) = 1 - \mathrm{Logistic}(\mathrm{Margins}(s_j))$$

thus

$$Pr(\mathrm{red}_j | s_j) = \mathrm{Logistic}(m_j)^{\mathrm{red}_j} \cdot (1 - \mathrm{Logistic}(m_j))^{1-\mathrm{red}_j}$$

$m_j$ = margin for stim j

thus over all training data

$$Pr(\mathbf{red}|\mathbf{s}) = \prod_{j=0}^{M-1} \mathrm{Logistic}(m_j)^{\mathrm{red}_j} \cdot (1 - \mathrm{Logistic}(m_j))^{1-\mathrm{red}_j}$$

as random variables

M = number of training examples

# Logistic Regression

Given this:

$$Pr(\mathbf{red}|\mathbf{s}) = \prod_{j=0}^{M-1} \text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}$$

M = number of training examples

(recall that's what we want to minimize)

What is the negative log likelihood?

# Logistic Regression

Given this:

$$Pr(\mathbf{red}|\mathbf{s}) = \prod_{j=0}^{M-1} \text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}$$

M = number of training examples

(recall that's what we want to minimize)

What is the negative log likelihood?

Taking log of both sides of the above gives:

$$-\log[Pr(\mathbf{red}|\mathbf{s})] = -\sum_{j=0}^{M-1} \log[\text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}]$$

since log(xy) = log(x) + log(y)

# Logistic Regression

Given this:

$$Pr(\mathbf{red}|\mathbf{s}) = \prod_{j=0}^{M-1} \text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}$$

M = number of training examples

(recall that's what we want to minimize)

What is the negative log likelihood?

$$-\log[Pr(\mathbf{red}|\mathbf{s})] = -\sum_{j=0}^{M-1} \log[\text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}]$$

$$= -\left(\sum_{j=0}^{M-1} \text{red}_j \log[\text{Logistic}(m_j)] + (1 - \text{red}_j) \log[1 - \text{Logistic}(m_j)]\right)$$

since log(a^b) = b * log(a)

# Logistic Regression

Given this:

$$Pr(\mathbf{red}|\mathbf{s}) = \prod_{j=0}^{M-1} \text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}$$

M = number of training examples

(recall that's what we want to minimize)

What is the negative log likelihood?

$$-\log[Pr(\mathbf{red}|\mathbf{s})] = -\sum_{j=0}^{M-1} \log[\text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}]$$

$$= -\left(\sum_{j=0}^{M-1} \boxed{\text{red}_j \log[\text{Logistic}(m_j)]} + \cancelto{0}{(1 - \text{red}_j)} \log[1 - \text{Logistic}(m_j)]\right)$$

If **red** = 1, the summand is:

$$= \log[\text{Logistic}(m_j)]$$

# Logistic Regression

Given this:

$$Pr(\mathbf{red}|\mathbf{s}) = \prod_{j=0}^{M-1} \mathrm{Logistic}(m_j)^{\mathrm{red}_j} \cdot (1 - \mathrm{Logistic}(m_j))^{1-\mathrm{red}_j}$$

M = number of training examples

(recall that's what we want to minimize)

What is the negative log likelihood?

$$-\log[Pr(\mathbf{red}|\mathbf{s})] = -\sum_{j=0}^{M-1} \log[\mathrm{Logistic}(m_j)^{\mathrm{red}_j} \cdot (1 - \mathrm{Logistic}(m_j))^{1-\mathrm{red}_j}]$$

$$= -\left(\sum_{j=0}^{M-1} \overset{0}{\mathrm{red}_j} \log[\mathrm{Logistic}(m_j)] + (1 - \mathrm{red}_j) \boxed{\log[1 - \mathrm{Logistic}(m_j)]}\right)$$

If **red** = 1

$$= \log[\mathrm{Logistic}(m_j)]$$

If **red** = 0 the summand is

$$= \log[1 - \mathrm{Logistic}(m_j)]$$

# Logistic Regression

Given this:

$$Pr(\mathbf{red}|\mathbf{s}) = \prod_{j=0}^{M-1} \text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}$$

M = number of training examples

(recall that's what we want to minimize)

What is the negative log likelihood?

$$-\log[Pr(\mathbf{red}|\mathbf{s})] = -\sum_{j=0}^{M-1} \log[\text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}]$$

$$= -\left(\sum_{j=0}^{M-1} \cancelto{0}{\text{red}_j} \log[\text{Logistic}(m_j)] + (1 - \text{red}_j) \boxed{\log[1 - \text{Logistic}(m_j)]}\right)$$

If **red** = 1

$$= \log[\text{Logistic}(m_j)]$$

If **red** = 0 the summand is

$$= \log[\text{Logistic}(-m_j)]$$

since Logistic is antisymmetric

# Logistic Regression

Given this:

$$Pr(\mathbf{red}|\mathbf{s}) = \prod_{j=0}^{M-1} \text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}$$

M = number of training examples

(recall that's what we want to minimize)

What is the negative log likelihood?

$$-\log[Pr(\mathbf{red}|\mathbf{s})] = -\sum_{j=0}^{M-1} \log[\text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}]$$

$$= -\left(\sum_{j=0}^{M-1} \cancelto{0}{\text{red}_j} \log[\text{Logistic}(m_j)] + (1 - \text{red}_j)\boxed{\log[1 - \text{Logistic}(m_j)]}\right)$$

If **red** = 1

$$= \log[\text{Logistic}(m_j)]$$

If **red** = 0 the summand is

$$= \log[\text{Logistic}(-m_j)]$$

So in either case, summand is $= \log[\text{Logistic}((1 - 2 * \text{red}_j)m_j)]$

# Logistic Regression

Given this:

$$Pr(\mathbf{red}|\mathbf{s}) = \prod_{j=0}^{M-1} \text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}$$

M = number of training examples

(recall that's what we want to minimize)

What is the negative log likelihood?

$$-\log[Pr(\mathbf{red}|\mathbf{s})] = -\sum_{j=0}^{M-1} \log[\text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}]$$

$$= -\sum_{j=0}^{M-1} \log[\text{Logistic}((1 - 2 * \text{red})m_j)]$$

# Logistic Regression

Given this:

$$Pr(\mathbf{red}|\mathbf{s}) = \prod_{j=0}^{M-1} \text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}$$

M = number of training examples

(recall that's what we want to minimize)

What is the negative log likelihood?

$$-\log[Pr(\mathbf{red}|\mathbf{s})] = -\sum_{j=0}^{M-1} \log[\text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}]$$

$$= -\sum_{j=0}^{M-1} \log\left[\frac{1}{1 + e^{(1-2*\text{red})m_j}}\right]$$

By definition of Logistic(x)

# Logistic Regression

Given this:

$$Pr(\mathbf{red}|\mathbf{s}) = \prod_{j=0}^{M-1} \text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}$$

M = number of training examples

(recall that's what we want to minimize)

What is the negative log likelihood?

$$-\log[Pr(\mathbf{red}|\mathbf{s})] = -\sum_{j=0}^{M-1} \log[\text{Logistic}(m_j)^{\text{red}_j} \cdot (1 - \text{Logistic}(m_j))^{1-\text{red}_j}]$$

$$= -\sum_{j=0}^{M-1} \log\left[\frac{1}{1 + e^{(1-2*\text{red})m_j}}\right]$$

$$= \sum_{j=0}^{M-1} \log(1 + e^{(1-2*\text{red})m_j})$$

since log(1/x) = -log(x)

# Logistic Regression

So instead of as in SVM where the loss is:

$$\text{loss} = \max\left(0, 1 - \text{red}_j * \left[\sum_i w_i \cdot neuron_i(s_j) + b\right]\right)$$

$$\text{red}_j = \begin{cases} 1 & \text{if } s_j \text{ red} \\ 0 & \text{if } s_j \text{ blue} \end{cases}$$

now our loss function seeks to find ***w***'s and ***b*** that minimize:

$$\text{loss} = \sum_{j=0}^{M-1} \log\left[1 + exp\left((1 - 2 * \text{red}_j)\left(\sum_i w_i \cdot \text{neuron}_i(s_j) + b\right)\right)\right]$$

for training stimuli s_0 …. s_{M-1}

# Logistic Regression

So instead of as in SVM where the loss is:

$$\text{loss} = \max\left(0, 1 - \text{red}_j * \left[\overbrace{\sum_i w_i \cdot neuron_i(s_j) + b}^{\text{hyperplane margins}}\right]\right)$$

$$\text{red}_j = \begin{cases} 1 & \text{if } s_j \text{ red} \\ 0 & \text{if } s_j \text{ blue} \end{cases}$$

now our loss function seeks to find w's and b that minimize:

$$\text{loss} = \sum_{j=0}^{M-1} \log\left[1 + exp\left((1 - 2 * \text{red}_j)\left(\overbrace{\sum_i w_i \cdot \text{neuron}_i(s_j) + b}^{\text{hyperplane margins}}\right)\right)\right]$$

for training stimuli s_0 …. s_{M-1}

# Logistic Regression

Let's unpack

$$\text{loss} = \sum_{j=0}^{M-1} \log\left[1 + exp\left((1 - 2 * \text{red}_j)\left(\overbrace{\sum_i w_i \cdot \text{neuron}_i(s_j) + b}^{\text{hyperplane margins}}\right)\right)\right]$$

# Logistic Regression

Let's unpack

$$\text{loss} = \sum_{j=0}^{M-1} \log\left[1 + exp\left((1 - 2 * \text{red}_j)\left(\underbrace{\sum_i w_i \cdot \text{neuron}_i(s_j) + b}_{\text{hyperplane margins}}\right)\right)\right]$$

If **red_j** = 1 (the stimulus is of the red class) then

$$\text{loss} = \sum_{j=0}^{M-1} \log\left[1 + exp\left(-\left(\underbrace{\sum_i w_i \cdot \text{neuron}_i(s_j) + b}_{\text{hyperplane margins}}\right)\right)\right]$$

# Logistic Regression

Let's unpack

$$\text{loss} = \sum_{j=0}^{M-1} \log\left[1 + exp\left((1 - 2 * \text{red}_j)\left(\overbrace{\sum_i w_i \cdot \text{neuron}_i(s_j) + b}^{\text{hyperplane margins}}\right)\right)\right]$$

If **red_j** = 1 (the stimulus is of the red class) then

$$\text{loss} = \sum_{j=0}^{M-1} \log\left[1 + exp\left(-\left(\overbrace{\sum_i w_i \cdot \text{neuron}_i(s_j) + b}^{\text{hyperplane margins}}\right)\right)\right]$$

so if the margin is big and positive (the guess is correct) then

$$1 + exp(-\text{margin}_j)$$

is close to 1 (since exp (- big numb) is small),

# Logistic Regression

Let's unpack

$$\text{loss} = \sum_{j=0}^{M-1} \log\left[1 + exp\left((1 - 2 * \text{red}_j)\left(\underbrace{\sum_i w_i \cdot \text{neuron}_i(s_j) + b}_{\text{hyperplane margins}}\right)\right)\right]$$

If **red_j** = 1 (the stimulus is of the red class) then

$$\text{loss} = \sum_{j=0}^{M-1} \log\left[1 + exp\left(-\left(\underbrace{\sum_i w_i \cdot \text{neuron}_i(s_j) + b}_{\text{hyperplane margins}}\right)\right)\right]$$

so if the margin is big and positive (the guess is correct) then

$$1 + exp(-\text{margin}_j)$$

is close to 1 (since exp (- big numb) is small), so <u>**loss is close to 0**</u>.

# Logistic Regression

Let's unpack

$$\text{loss} = \sum_{j=0}^{M-1} \log\left[1 + exp\left((1 - 2 * \text{red}_j)\left(\overbrace{\sum_i w_i \cdot \text{neuron}_i(s_j) + b}^{\text{hyperplane margins}}\right)\right)\right]$$

If **red_j** = 1 (the stimulus is of the red class) then

$$\text{loss} = \sum_{j=0}^{M-1} \log\left[1 + exp\left(-\left(\overbrace{\sum_i w_i \cdot \text{neuron}_i(s_j) + b}^{\text{hyperplane margins}}\right)\right)\right]$$

while if the margin is big and negative (the guess is very wrong) then

$$1 + exp(margin_j)$$

is >> 1 (since exp (big numb) is big), so **loss >> 0.**

# Logistic Regression

Let's unpack

$$\text{loss} = \sum_{j=0}^{M-1} \log \left[ 1 + exp \left( (1 - 2 * \text{red}_j) \left( \overbrace{\sum_i w_i \cdot \text{neuron}_i(s_j) + b}^{\text{hyperplane margins}} \right) \right) \right]$$

If **red_j** = 0 (the stimulus is of the blue class) then

$$\text{loss} = \sum_{j=0}^{M-1} \log \left[ 1 + exp \left( \overbrace{\sum_i w_i \cdot \text{neuron}_i(s_j) + b}^{\text{hyperplane margins}} \right) \right]$$

# Logistic Regression

Let's unpack

$$\text{loss} = \sum_{j=0}^{M-1} \log\left[1 + exp\left((1 - 2 * \text{red}_j)\left(\underbrace{\sum_i w_i \cdot \text{neuron}_i(s_j) + b}_{\text{hyperplane margins}}\right)\right)\right]$$

If **red_j** = 0 (the stimulus is of the blue class) then

$$\text{loss} = \sum_{j=0}^{M-1} \log\left[1 + exp\left(\underbrace{\sum_i w_i \cdot \text{neuron}_i(s_j) + b}_{\text{hyperplane margins}}\right)\right]$$

so if the margin is big and negative (the guess is correct) then

$$1 + exp(margin_j)$$

is close to 1 (since exp (- big numb) is small), so <u>**loss is close to 0**</u>.

# Logistic Regression

Let's unpack

$$\text{loss} = \sum_{j=0}^{M-1} \log\left[1 + exp\left((1 - 2 * \text{red}_j)\left(\overbrace{\sum_i w_i \cdot \text{neuron}_i(s_j) + b}^{\text{hyperplane margins}}\right)\right)\right]$$

If **red_j** = 0 (the stimulus is of the blue class) then

$$\text{loss} = \sum_{j=0}^{M-1} \log\left[1 + exp\left(\overbrace{\sum_i w_i \cdot \text{neuron}_i(s_j) + b}^{\text{hyperplane margins}}\right)\right]$$

while if the margin is big and positive (the guess is very wrong) then

$$1 + exp(margin_j)$$

is >> 1 (since exp (big numb) is big), so **loss >> 0.**

# Logistic Regression

How do we do multi-class classification with Logistic regression?

1. One-vs-Rest, just like with SVM

# Logistic Regression

*[IPYNB:   Logistic Regression]*

# Logistic Regression

How do we do multi-class classification with Logistic regression?

1. One-vs-Rest, just like with SVM

2. By using probabilities directly:

$$\text{loss} = -\sum_{j=0}^{M-1} \log\left[\frac{e^{m_{j,y_j}}}{\sum_k e^{m_{j,k}}}\right]$$

margin for stimulus j
for **just** correct class y_j

sum over margins for all
categories *k* for stimulus j

# Logistic Regression

$$\frac{e^{m_{j,y_j}}}{\sum_k e^{m_{j,k}}}$$

margin for stimulus j for **just** correct class y_j

sum over margins for all categories *k* for stimulus j

# Logistic Regression

$$\frac{e^{m_{j,y_j}}}{\sum_k e^{m_{j,k}}}$$

margin for stimulus j for **just** correct class y_j

sum over margins for all categories *k* for stimulus j

If correct class = 1 ("red")

$$\frac{e^{m_{j,1}}}{e^{m_{j,1}} + e^{m_{j,0}}}$$

# Logistic Regression

$$\frac{e^{m_{j,y_j}}}{\sum_k e^{m_{j,k}}}$$

margin for stimulus j for **just** correct class y_j

sum over margins for all categories *k* for stimulus j

If correct class = 1 ("red")

$$\frac{e^{m_{j,1}}}{e^{m_{j,1}} + e^{m_{j,0}}} = \frac{e^{m_{j,1}}}{e^{m_{j,1}} + e^{-m_{j,1}}}$$

# Logistic Regression

$$\frac{e^{m_{j,y_j}}}{\sum_k e^{m_{j,k}}}$$

margin for stimulus j for **just** correct class y_j

sum over margins for all categories *k* for stimulus j

If correct class = 1 (“red”)

$$\frac{e^{m_{j,1}}}{e^{m_{j,1}} + e^{m_{j,0}}} = \frac{e^{m_{j,1}}}{e^{m_{j,1}} + e^{-m_{j,1}}} = \frac{e^{m_j}}{e^{m_j} + e^{-m_j}}$$

# Logistic Regression

$$\frac{e^{m_{j,y_j}}}{\sum_k e^{m_{j,k}}}$$

margin for stimulus j for **just** correct class y_j

sum over margins for all categories *k* for stimulus j

If correct class = 1 (“red”)

$$\frac{e^{m_{j,1}}}{e^{m_{j,1}} + e^{m_{j,0}}} = \frac{e^{m_{j,1}}}{e^{m_{j,1}} + e^{-m_{j,1}}} = \frac{e^{m_j}}{e^{m_j} + e^{-m_j}} = \frac{1}{1 + e^{-2m_j}}$$

# Logistic Regression

$$\frac{e^{m_{j,y_j}}}{\sum_k e^{m_{j,k}}}$$

margin for stimulus j for **just** correct class y_j

sum over margins for all categories *k* for stimulus j

If correct class = 1 ("red")

$$\frac{e^{m_{j,1}}}{e^{m_{j,1}} + e^{m_{j,0}}} = \frac{e^{m_{j,1}}}{e^{m_{j,1}} + e^{-m_{j,1}}} = \frac{e^{m_j}}{e^{m_j} + e^{-m_j}} = \frac{1}{1 + e^{-2m_j}}$$

$$\frac{e^{m_{j,0}}}{e^{m_{j,1}} + e^{m_{j,0}}} = \frac{-e^{m_{j,1}}}{e^{m_{j,1}} + e^{-m_{j,1}}} = \frac{-e^{m_j}}{e^{m_j} + e^{-m_j}} = \frac{1}{1 + e^{2m_j}}$$

If correct class = 0 ("blue")

# Logistic Regression

$$\frac{e^{m_{j,y_j}}}{\sum_k e^{m_{j,k}}}$$

margin for stimulus j for **just** correct class y_j

sum over margins for all categories *k* for stimulus j

If correct class = 1 ("red")

$$\frac{e^{m_{j,1}}}{e^{m_{j,1}} + e^{m_{j,0}}} = \frac{e^{m_{j,1}}}{e^{m_{j,1}} + e^{-m_{j,1}}} = \frac{e^{m_j}}{e^{m_j} + e^{-m_j}} = \frac{1}{1 + e^{-2m_j}}$$

$$\frac{e^{m_{j,0}}}{e^{m_{j,1}} + e^{m_{j,0}}} = \frac{-e^{m_{j,1}}}{e^{m_{j,1}} + e^{-m_{j,1}}} = \frac{-e^{m_j}}{e^{m_j} + e^{-m_j}} = \frac{1}{1 + e^{2m_j}}$$

If correct class = 0 ("blue")

in all cases $$= \frac{1}{1 + e^{(1-2*\mathrm{red}_j)*2m_j}}$$

Exactly same formula from binary case.

# Logistic Regression

$$\frac{e^{m_{j,y_j}}}{\sum_k e^{m_{j,k}}}$$

margin for stimulus j for **just** correct class y_j

sum over margins for all categories *k* for stimulus j

If correct class = 1 ("red")

$$\frac{e^{m_{j,1}}}{e^{m_{j,1}} + e^{m_{j,0}}} = \frac{e^{m_{j,1}}}{e^{m_{j,1}} + e^{-m_{j,1}}} = \frac{e^{m_j}}{e^{m_j} + e^{-m_j}} = \frac{1}{1 + e^{-2m_j}}$$

$$\frac{e^{m_{j,0}}}{e^{m_{j,1}} + e^{m_{j,0}}} = \frac{-e^{m_{j,1}}}{e^{m_{j,1}} + e^{-m_{j,1}}} = \frac{-e^{m_j}}{e^{m_j} + e^{-m_j}} = \frac{1}{1 + e^{2m_j}}$$

If correct class = 0 ("blue")

in all cases $= \dfrac{1}{1 + e^{(1-2*\mathrm{red}_j)*\boxed{2}m_j}}$

Exactly same formula from binary case, … except extra factor of 2.

# Logistic Regression

*[IPYNB:   Logistic Regression with softmax]*