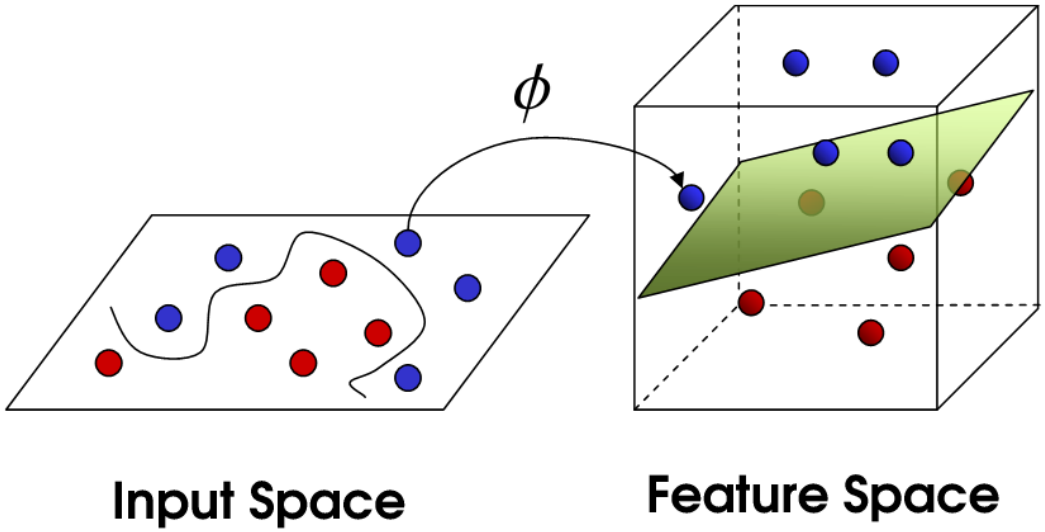
CS109 – Data Science
SVM, Performance evaluation

Joe Blitzstein, Hanspeter Pfister, Verena Kaynig-Fittkau



http://i.stack.imgur.com/1gvce.png

Announcements

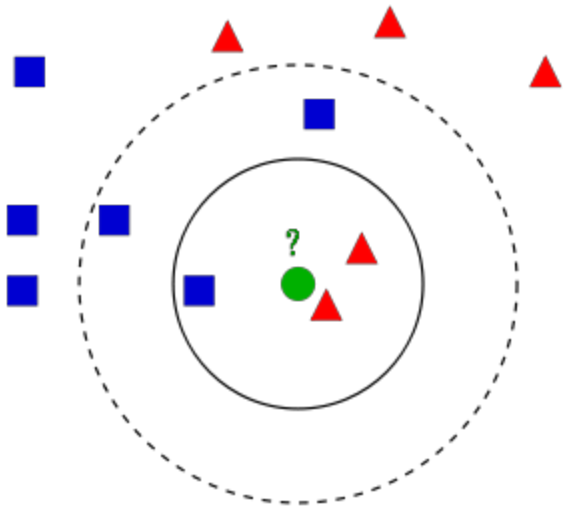
- HW1 grades went out yesterday
- They are looking really good, well done everyone!

- HW2 is due this Thursday!

- You should submit an executed notebook
- But please without pages of test output

Recap K-NN

- Keeps all training data
- Training is fast
- Prediction is slow

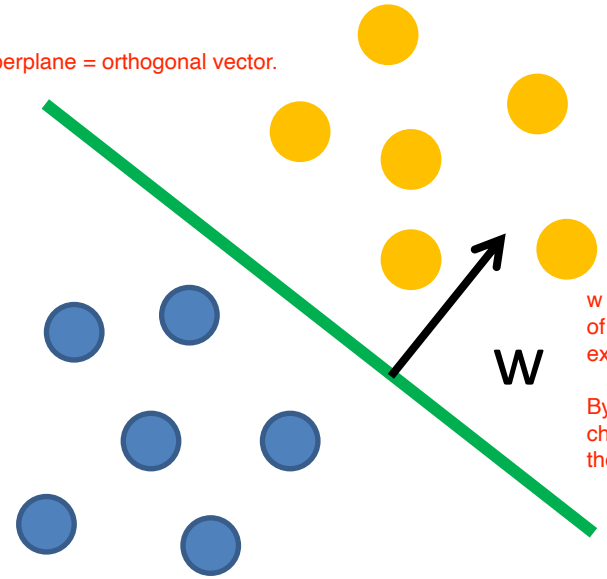


Separating Hyperplane

the goal is to estimate w

- x: data point
- y: label $\in \{-1, +1\}$
- w: weight vector

perpendicular to the hyperplane = orthogonal vector.



w defines the orientation of the hyperplane, in this example.

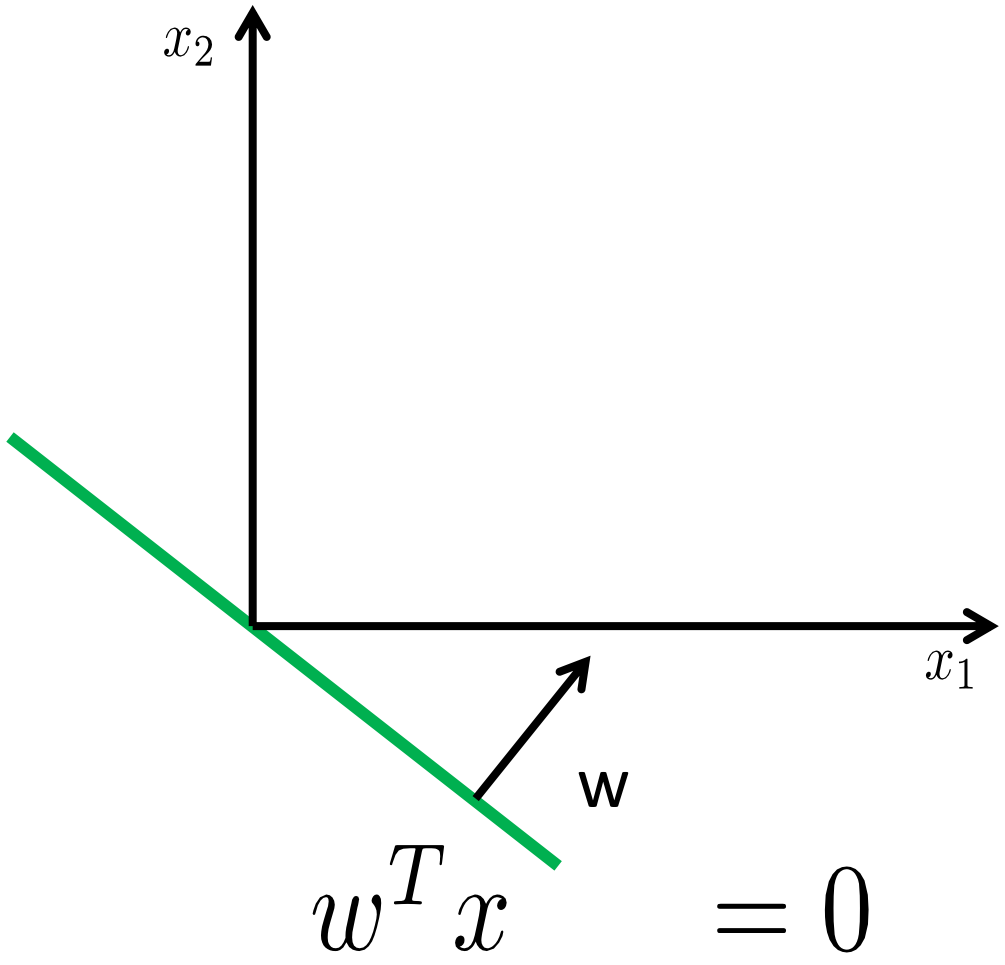
By changing w, you change the orientation of the hyperplane.

Here, the yellow colors are considered to have a label of +1, while blue has the label of -1.

$$w^T x = 0$$

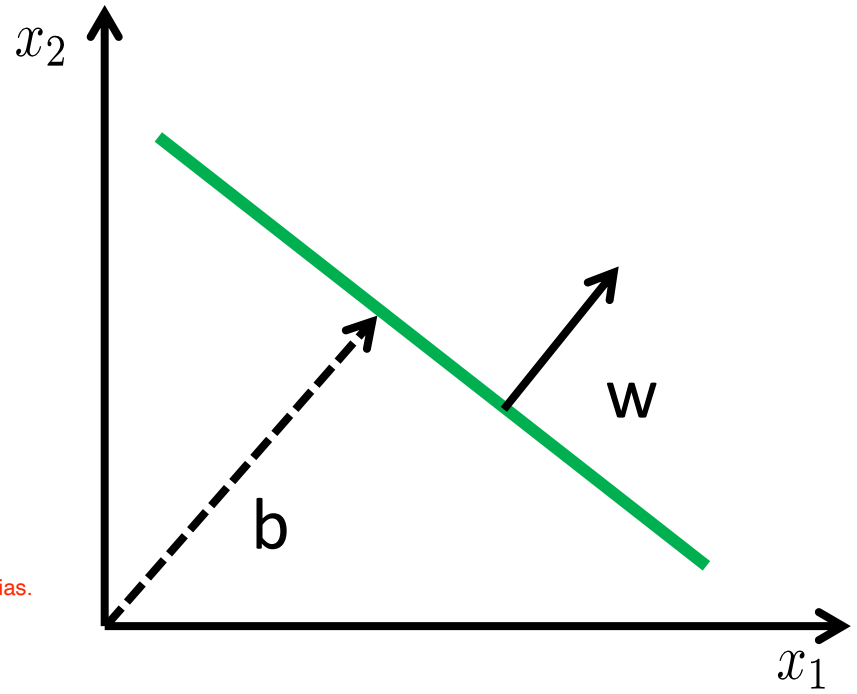
Separating Hyperplane

- x: data point
- y: label $\in \{-1, +1\}$
- w: weight vector



Separating Hyperplane

- x: data point
- y: label $\in \{-1, +1\}$
- w: weight vector
- b: bias



Two things needed to know the separating hyperplane: weight & bias.

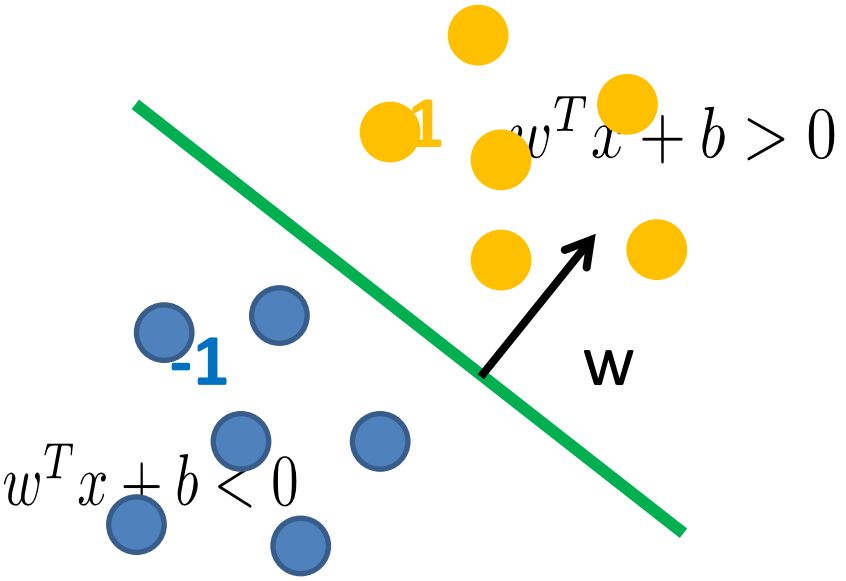
weight, to change orientation
bias to change the height of separating hyperplane.

You can then optimize them randomly and have the hyperplane 'wiggle' in between the two classes/labels.

$$w^T x + b = 0$$

Separating Hyperplane

- x: data point
- y: label $\in \{-1, +1\}$
- w: weight vector
- b: bias



Once you have w & b, prediction becomes much easier.

Here, you take your data point x and you evaluate this equation: w*x+b

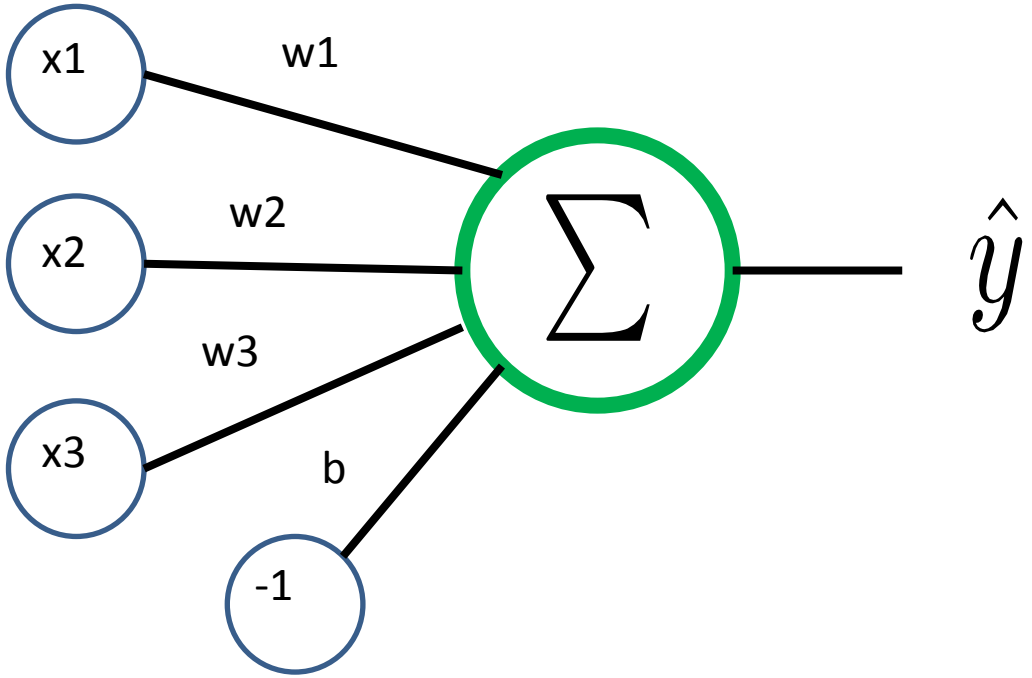
You look at the sine of that result. If it's greater than 0, then it's on the right side of the hyperplane, and if it's less than 0, it's on the other side of the hyperplane.

All you'd need to store, rather than the training data like in KNN, are the above parameters.

However, the tradeoff is that you're now restricted to a line, while KNN can be more tailored to the data points.

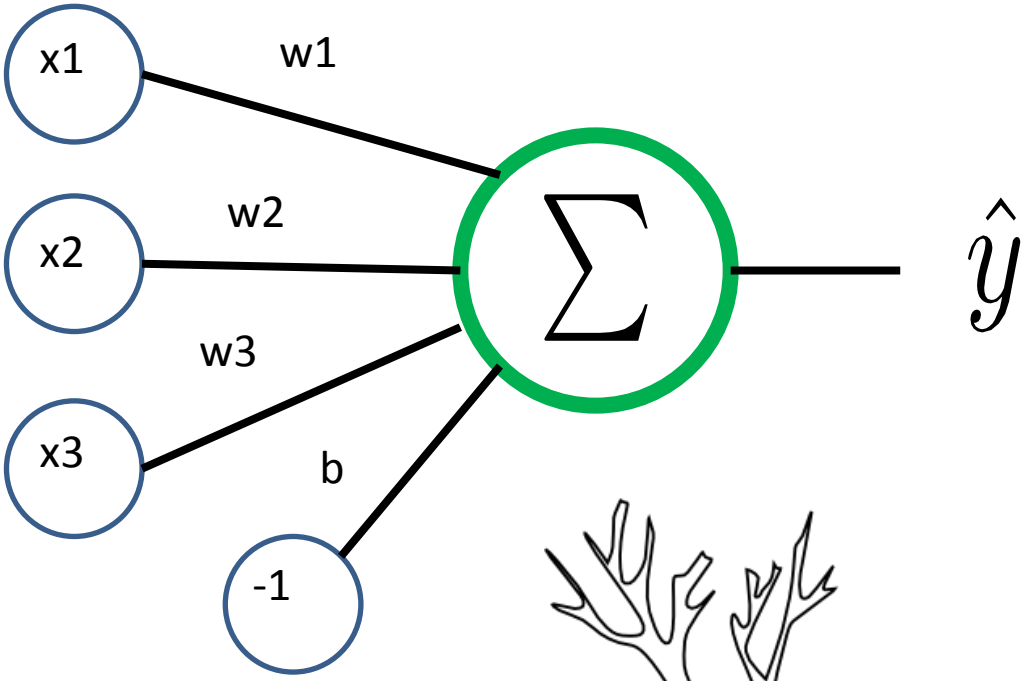
The goal is to keep all the positive things about this without the negative costs of KNN.

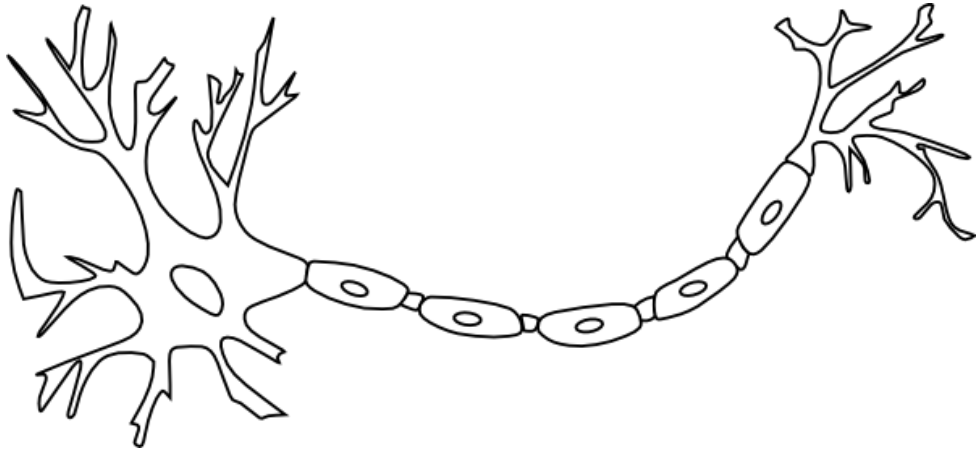
Perceptron



$$w^T x + b = 0$$

Perceptron





Perceptron History

- invented 1957
- by Frank Rosenblatt

- the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence. (NYT 1958)

  (http://en.wikipedia.org/wiki/Perceptron

These perceptron machines were the basic components of what would become Deep Learning.

This was the theory behind the neural network, that it looked like a neuron and therefore how a human brain lerns.

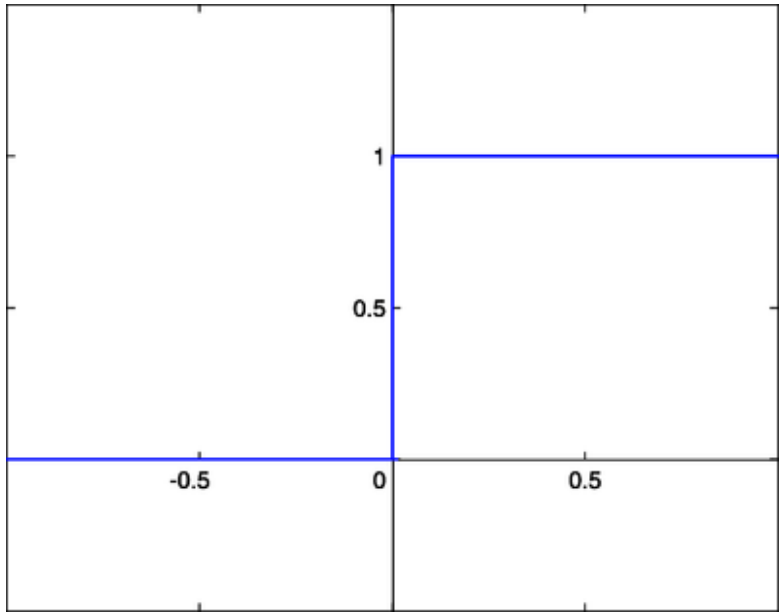
Perceptron.mp4

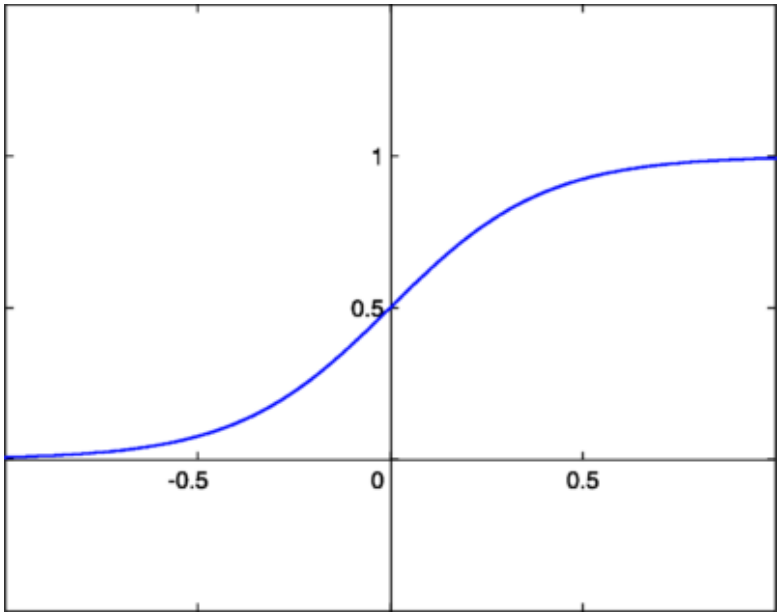
https://www.youtube.com/watch?v=cNxadbrN_aI&list=PLdVOMWcqwwIlaygvb9ZteZ1r4Br6kRuBO

Side Note: Step vs Sigmoid Activation

A sigmoid function is a continuous approximation of the step function.

This is actually a representation of a logistic regression.





$$s(x) = \frac{1}{1 + e^{-cx}}$$

The Critics

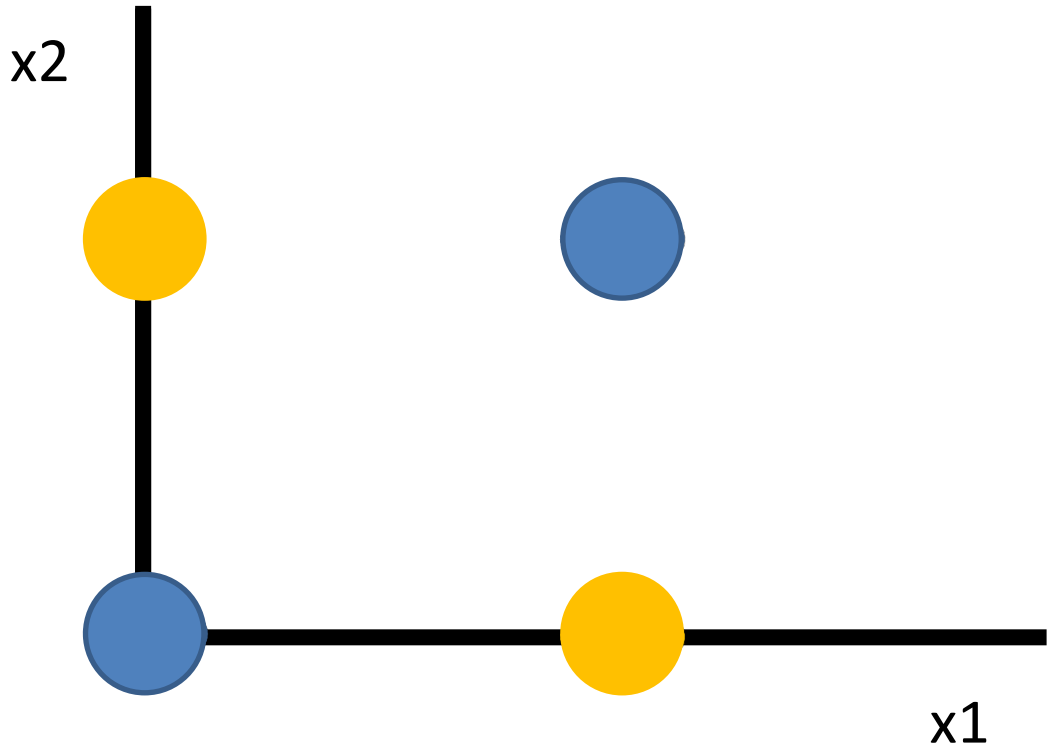
- 1969: Minsky and Papert publish their book “Perceptrons”

- Very controversial book, some blame the book for causing the whole research area to stagnate.

https://en.wikipedia.org/wiki/Perceptrons_(book)

The XOR Problem

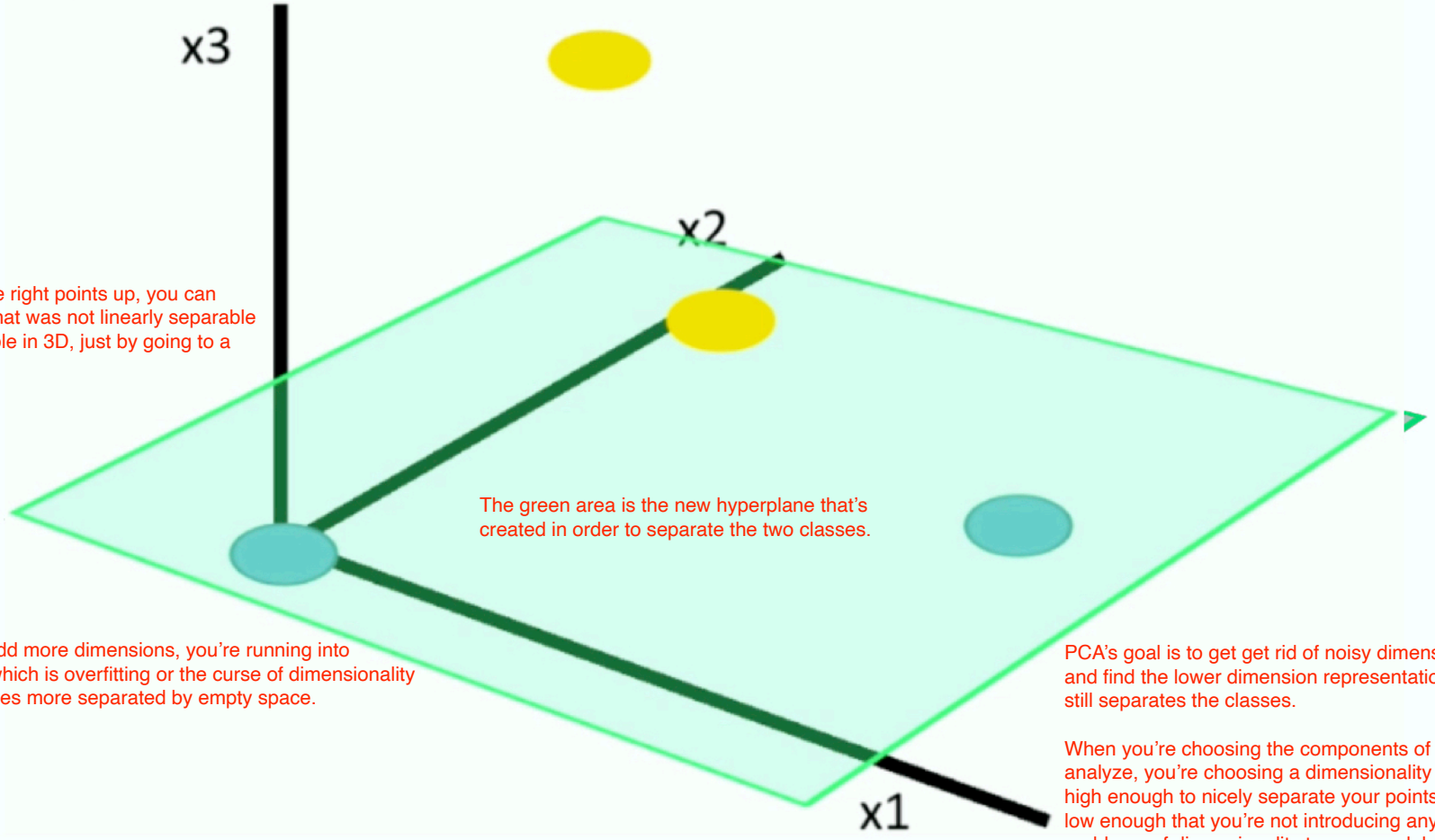
There isn't a single line here that can separate the two classes, cleanly.



The XOR Problem

The solution is to add a third dimension and do a trick right, in order to create that decision boundary.

You can elevate the yellow data points/observations to a third dimension and allow for classification.



The trick is to lift the right points up, you can have the problem that was not linearly separable in 2D to be separable in 3D, just by going to a higher dimension.

The green area is the new hyperplane that's created in order to separate the two classes.

However, as you add more dimensions, you're running into another problem, which is overfitting or the curse of dimensionality as you data becomes more separated by empty space.

PCA's goal is to get get rid of noisy dimensions, and find the lower dimension representation that still separates the classes.

When you're choosing the components of PCA to analyze, you're choosing a dimensionality that's high enough to nicely separate your points, but low enough that you're not introducing any problems of dimensionality to your model.

Support Vector Machine

- Widely used for all sorts of classification problems

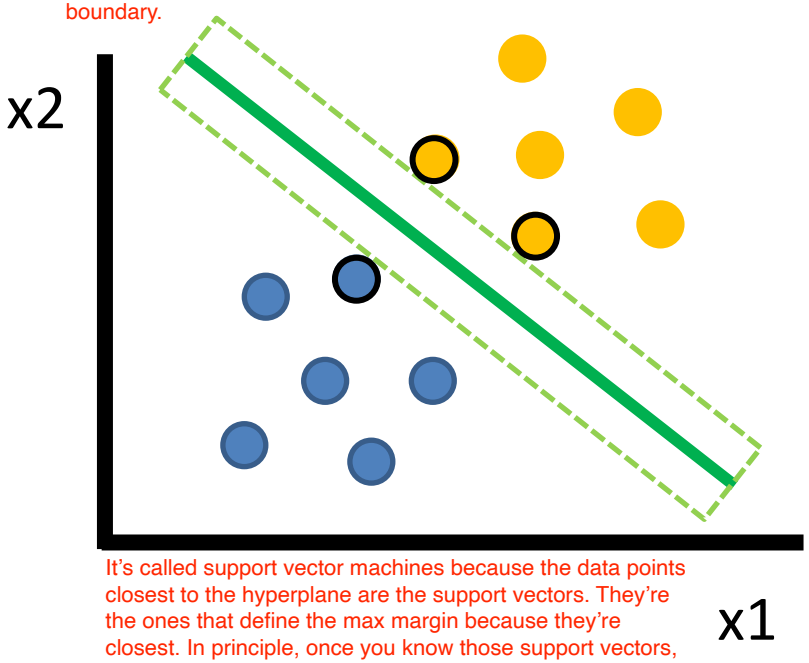
- Some people say it is the best of the shelf classifier out there

Lecturer: I don't agree with the latter point because you have to tune some parameters to make SVMs perform well.

Maximum Margin Classification

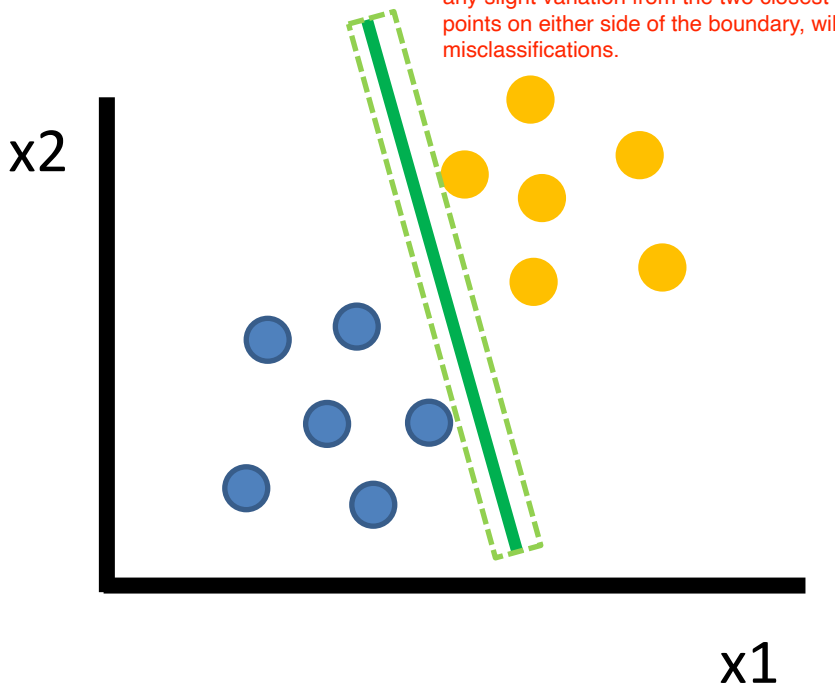
For the Perceptron, either works because there aren't any mistakes and all data is correctly identified..

SVMs doing this kind of classification work by trying to maximize the distance between the closest data points on iether side of the decision boundary.



It's called support vector machines because the data points closest to the hyperplane are the support vectors. They're the ones that define the max margin because they're closest. In principle, once you know those support vectors, you could forget about the rest of them.

Because the boundary is very close, it's not going to generalize very well with new data because any slight variation from the two closest data points on either side of the boundary, will result in misclassifications.

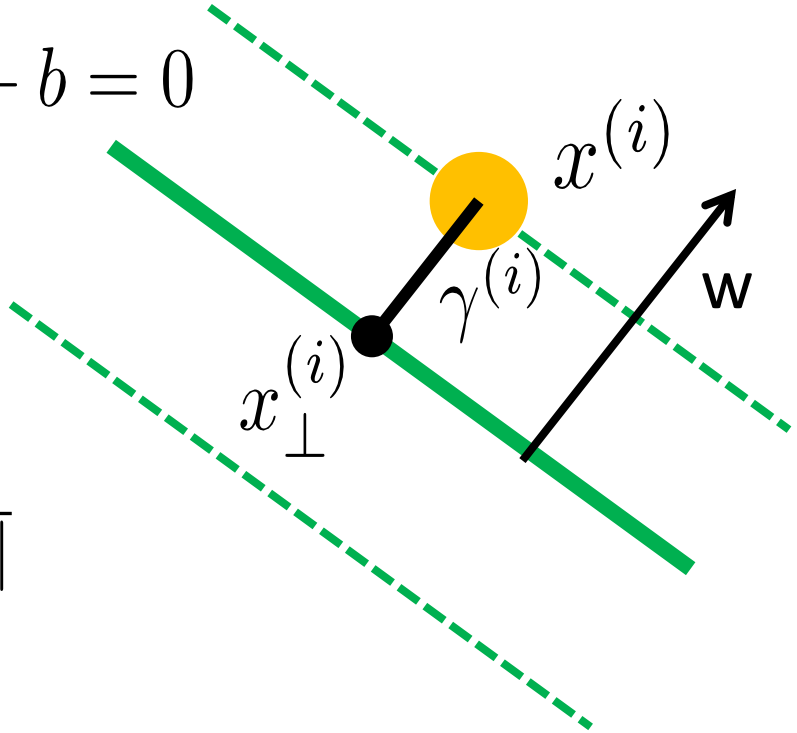


Solution depends only on the support vectors!

However, you'd want as much data as possible so that you'll be able to identify the right support vectors.

Maximum Margin Classification

gamma = the distance to the maximum margin from the hyperplane boundary.



margin:

$$x_\perp^{(i)} = x^{(i)} - \gamma^{(i)} \cdot \frac{w}{||w||}$$

$$w^T x_\perp^{(i)} + b = 0$$

$$\gamma^{(i)} = \left(\frac{w^T x^{(i)} + b}{||w||} \right)$$

The direction to x-bottom, is given by the opposite of w (orientation of hyperplane).

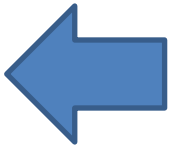
w can have any length, so you need to normalize its length, 1 and then use gamma to know how many steps of w, one has to go the get to x-bottom.

Ultimately, you're trying to find the value of w such that your gamma is as wide as possible.

Maximum Margin Classification

$$\gamma^{(i)} = y^{(i)}(w^T x + b)$$

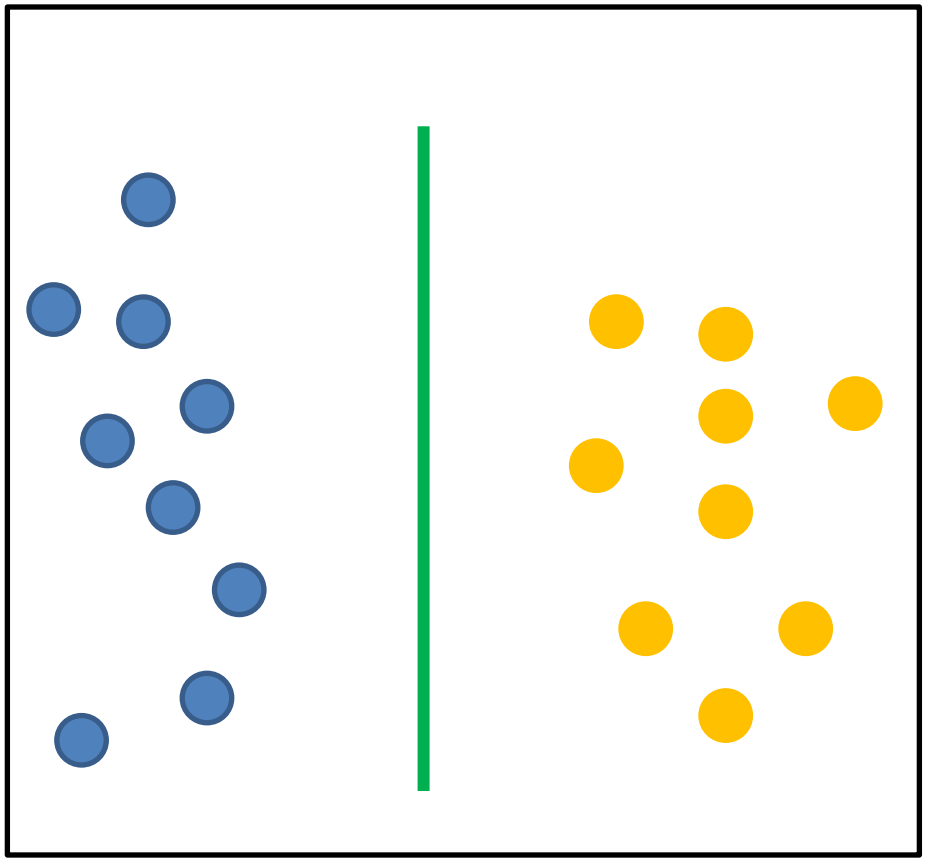
$$\begin{aligned} \max_{\gamma,w,b} \quad & \gamma \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq \gamma, \quad i = 1, \ldots, m \\ & ||w|| = 1. \end{aligned}$$



non-convex

software generally handles it for you and you wouldn't need to provide your own optimizer for support verctor machines.

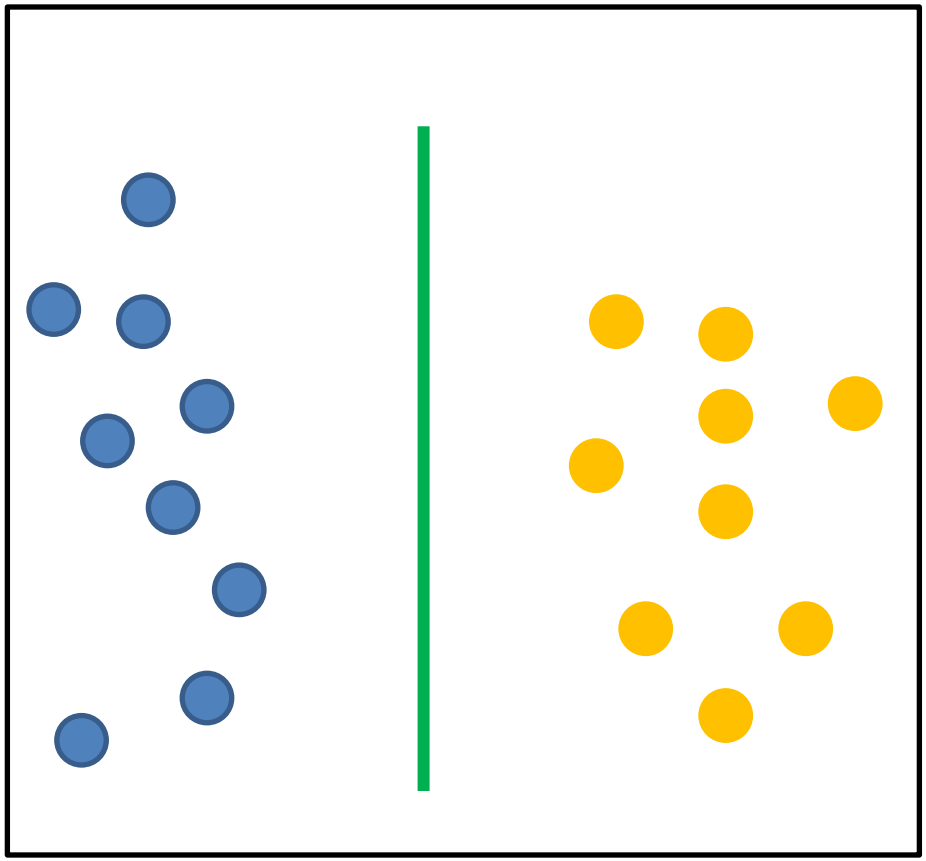
This Is Kind of Odd

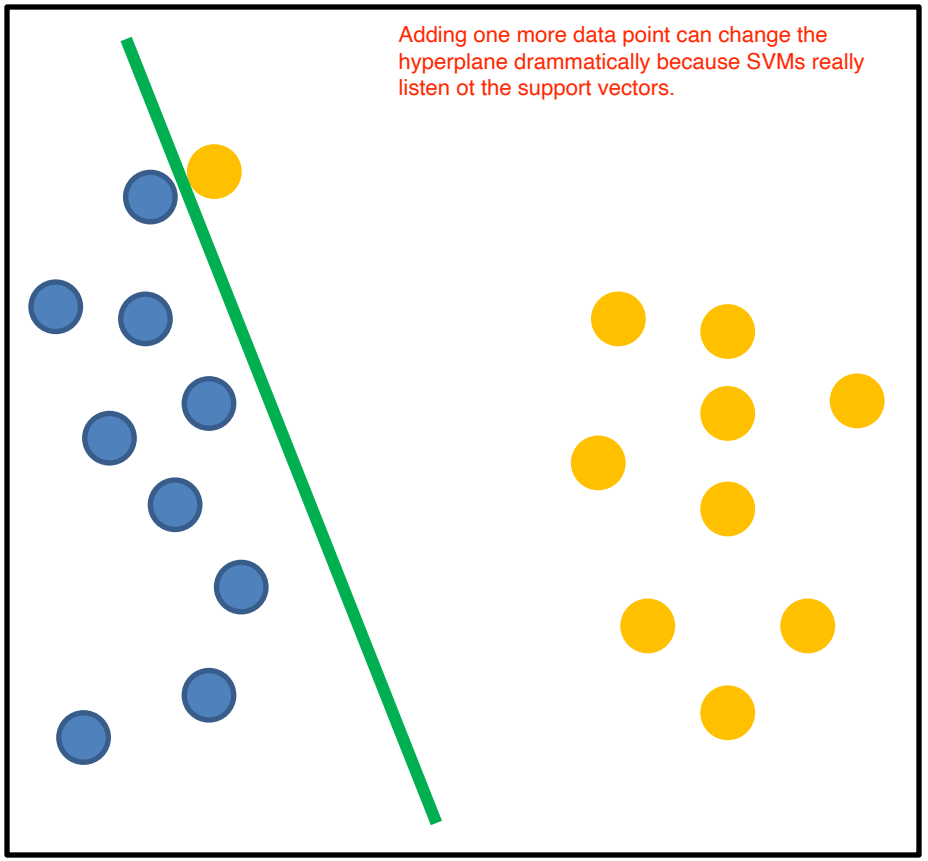


- Which data points do we care the most about?
- What would those samples look like?

The SVM cares most about the borderline cases:
- If we're looking at the color classes that separate an apple from an orange, then SVMs look for the most apple-like orange and the most orange-like apple.
- It doesn't care about the idealized apple nor orange, but those where the boundary almost overlap.

Two Very Similar Problems





What about outliers?

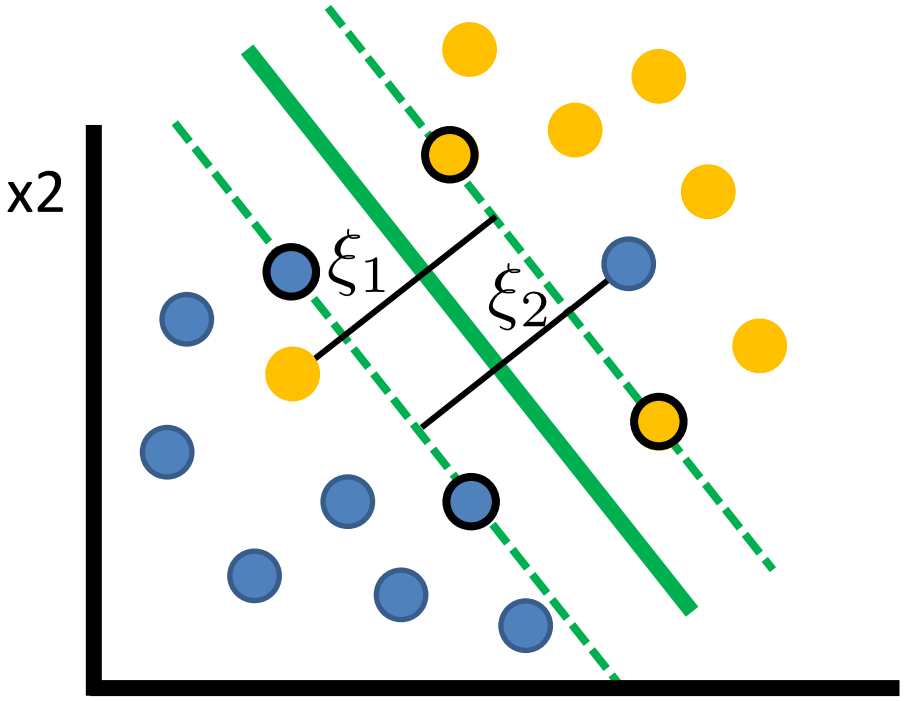
ξ_i: slack variables

$$\min_{w,b,\xi} \frac{1}{2}||w||^2$$

subject to:

$$y^{(i)}(w^T x^{(i)} + b) \geq 1$$

$$(i = 1, \ldots, n)$$



Here we have variables in the wrong data-cloud. We do not want a perfect classification on our training data.

The solution is to tell the SVM to allow for slack variables. It's calculating the outliers' distance from the margin they should actually lie on, on the opposite side of the decision boundary.

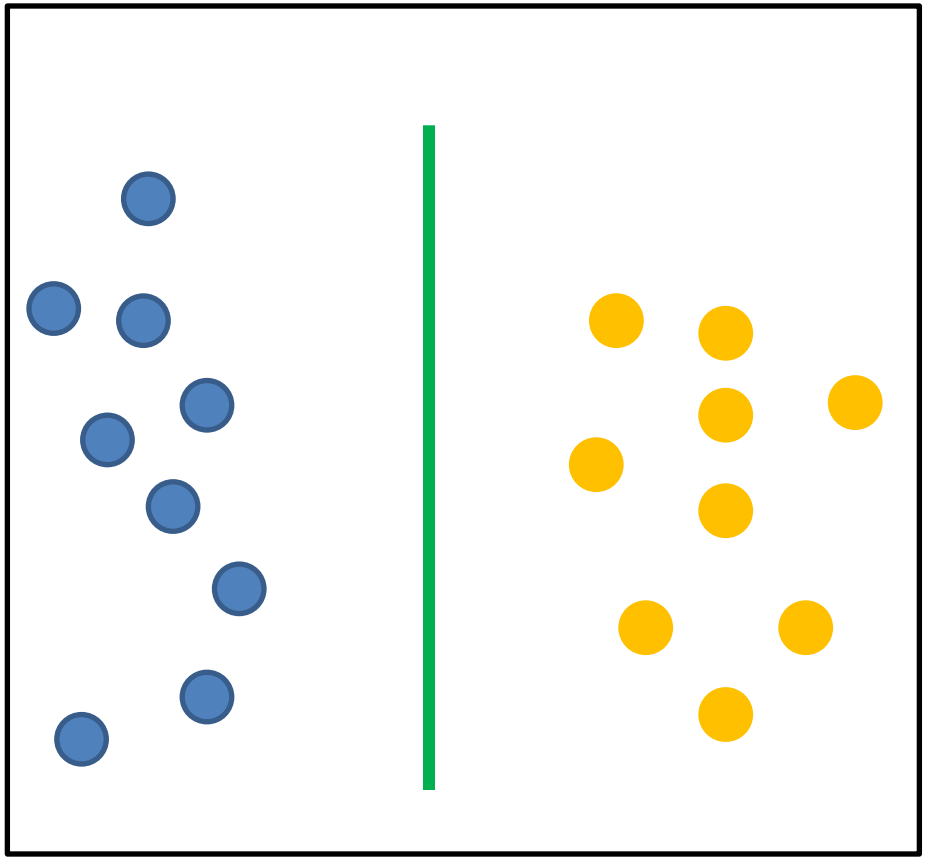
When you make C large (infinity) then the SVM will try to minimize the values of the slack variables. So, C is large means there's not a lot of slack allowed.

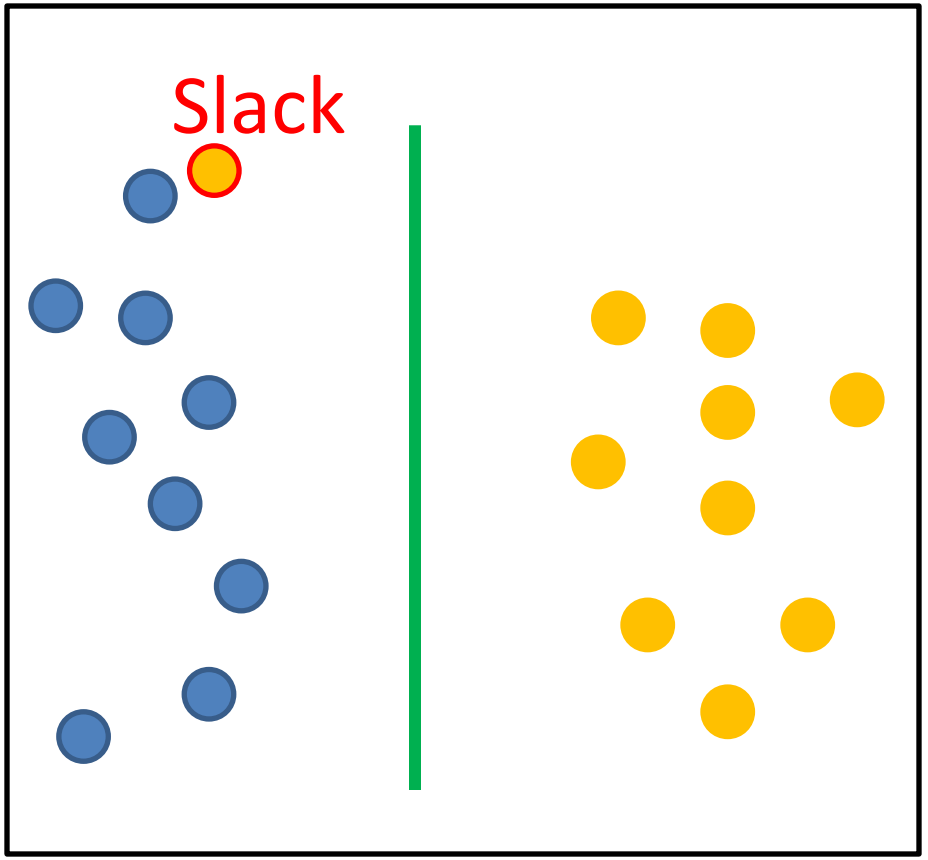
A lower value of C allows the SVM to misclassify things (have more slack variables distance).

Two Very Similar Problems

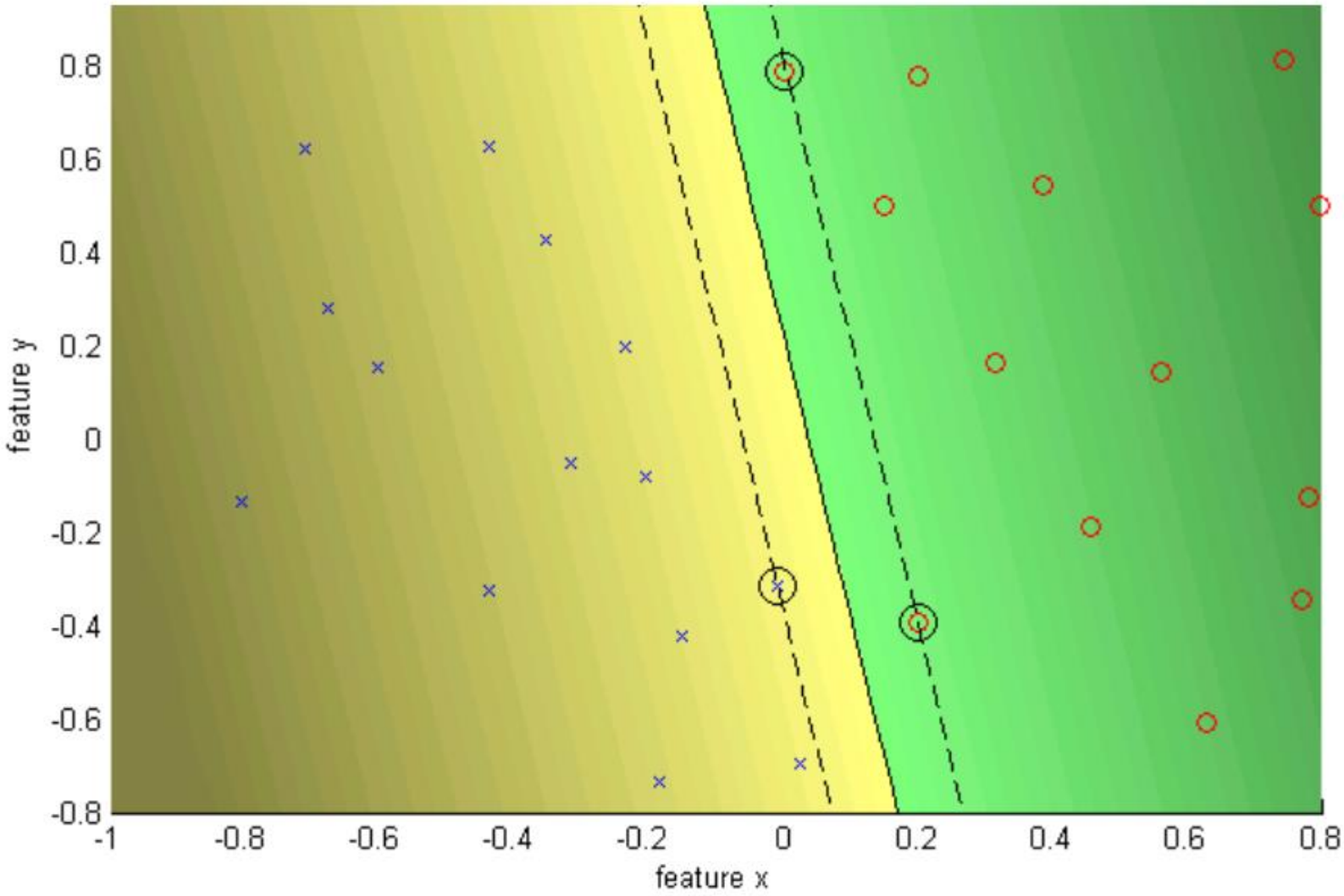
With the slack, the decision boundary isn't altered, at least dramatically, and it leads to a similar solution.

If you make C small enough then that slack variable will allow for the decision boundary to not be affected.



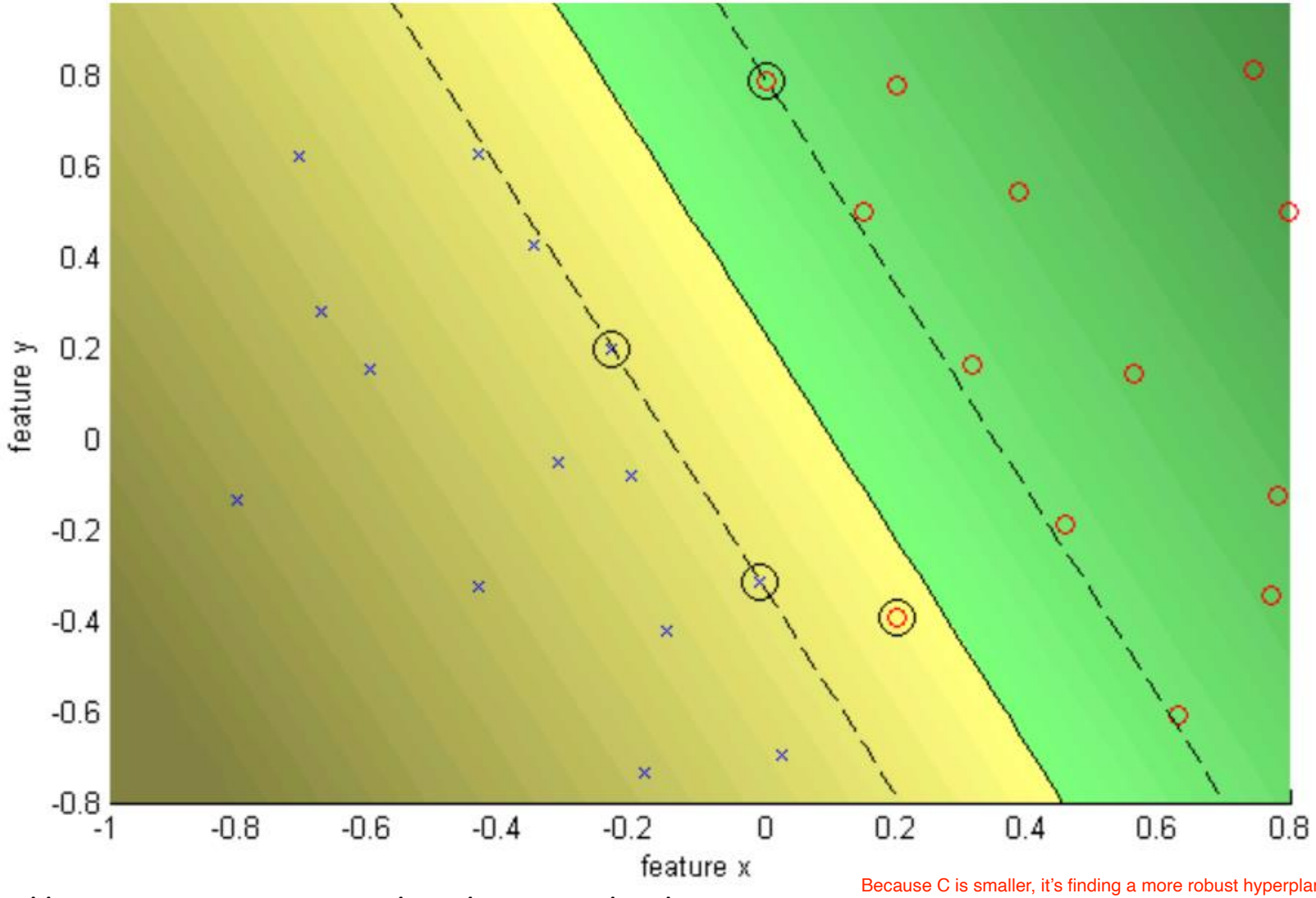


Hard Margin (C = Infinity)



http://www.robots.ox.ac.uk/~az/lectures/ml/lect2.pdf

Soft Margin (C = 10)

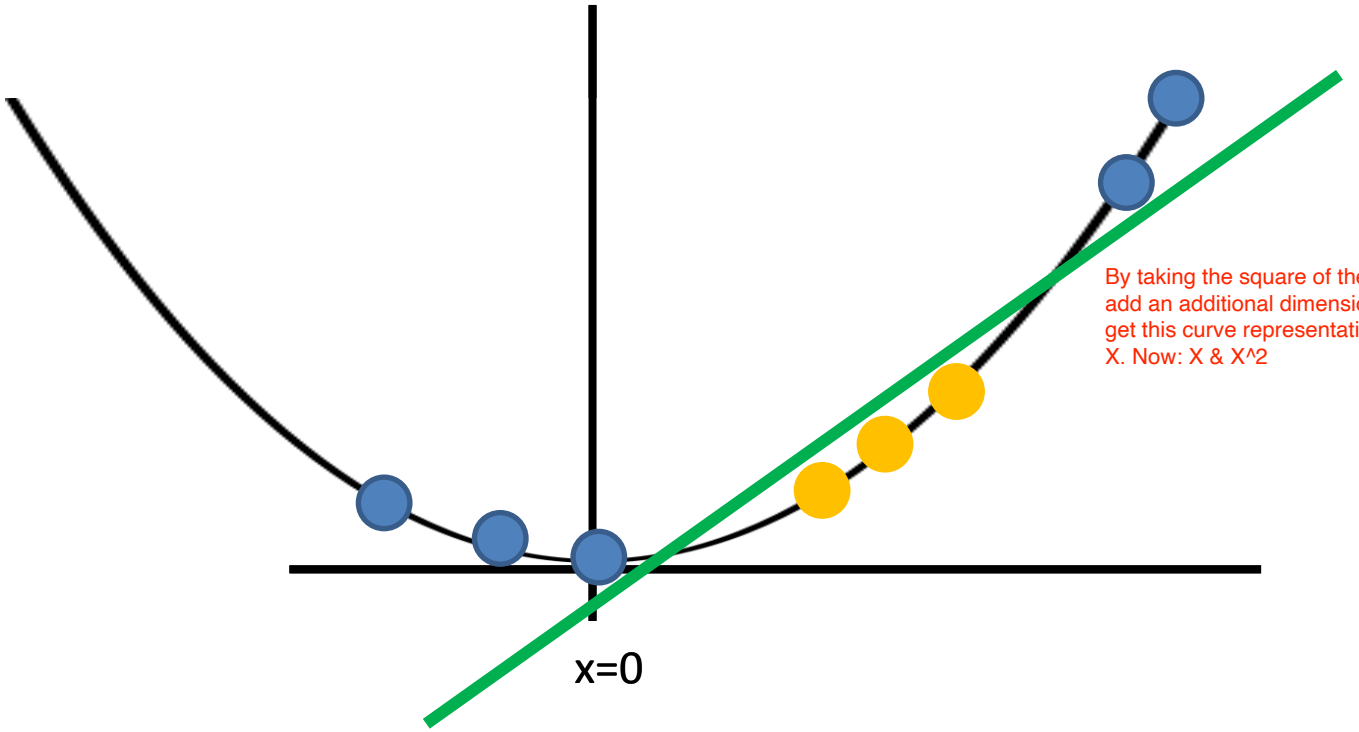


http://www.robots.ox.ac.uk/~az/lectures/ml/lect2.pdf

Because C is smaller, it's finding a more robust hyperplane and the cost of misclassifying a point. It can do this because it has the slack variables.

XOR problem revised

If the data were represented in one dimension, it's similar to the original XOR problem where a nice line doesn't exist to separate the colors/data points well.

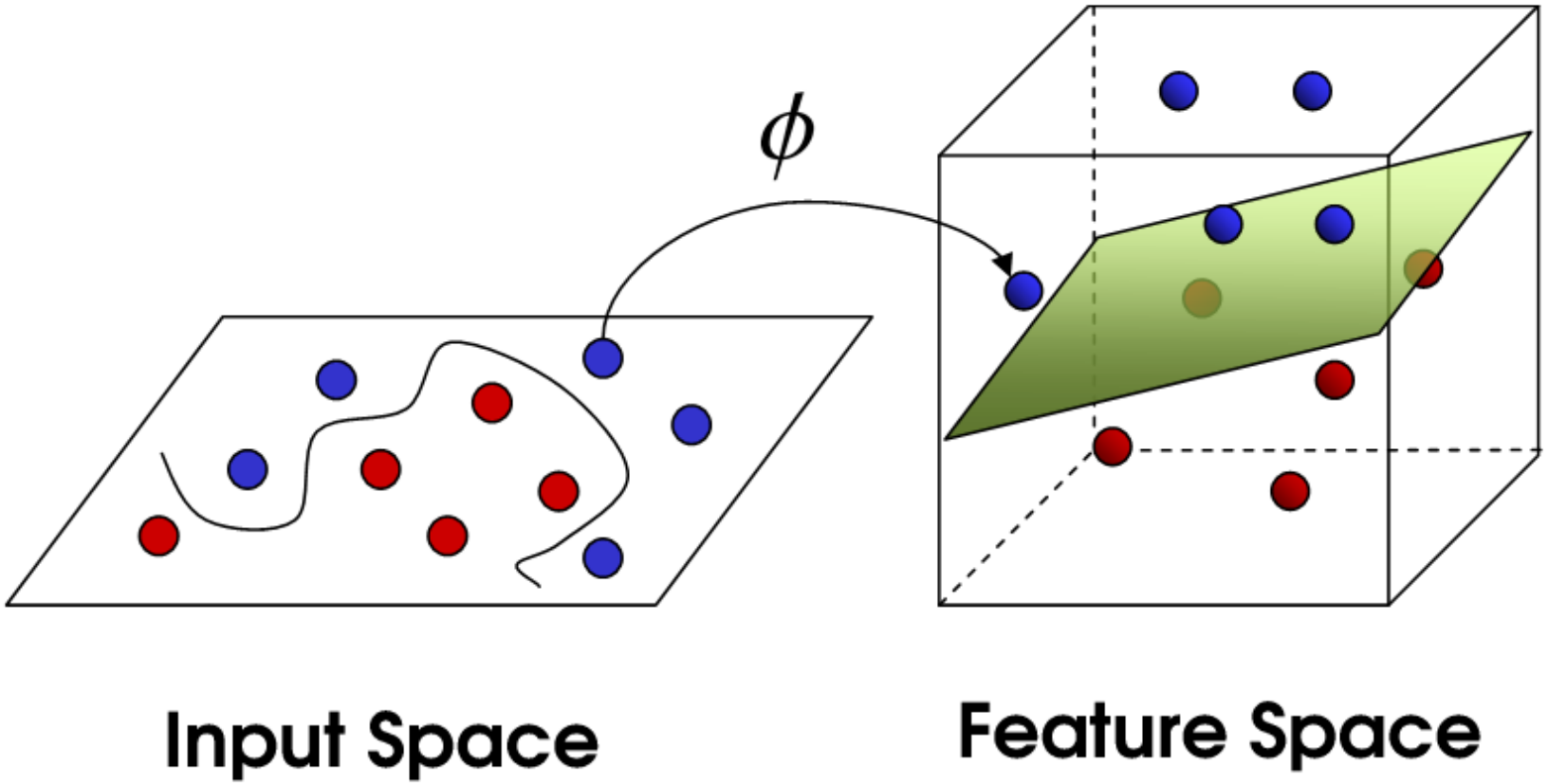


By taking the square of the data, you add an additional dimension and you get this curve representation. Before: X. Now: X & X^2

Did we add information to make the problem seperable?

No, because you're using the data you already have, not adding additional data points.

Non-Linear Decision Boundary



SVM with a polynomial Kernel visualization

Created by:
Udi Aharoni

Quadratic Kernel

By using this trick, you can get non-linear decision boundaries even though the math you're doing is linear: (X -> X^2 is a linear relationship)

$$x = (x_1, x_2)$$

$$\Phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

Referenced as a 'quadratic expansion'

$$\Phi(x) \cdot \Phi(z) = 1 + 2\sum_{i=1}^{d} x_i z_i$$

$$+ \sum_{i=1}^{d} x_i^2 z_i^2 + 2\sum_{i=1}^{d} \sum_{j=i+1}^{d} x_i x_j z_i z_j$$

$$= (1 + x \cdot z)^2$$

The trick is that using this formula you can get the values of the higher dimensions without having to calculate those higher dimnesinoal values being done in the above formulas.

Kernel Functions

$$K(x, z) = \Phi(x) \cdot \Phi(z)$$

- Polynomial:

$$K(x, z) = (1 + x \cdot z)^s$$

For the polynomial kernel function you need to tune the degree.

- Radial basis function (RBF):

$$K(x, z) = \exp(-\gamma(x - z)^2)$$

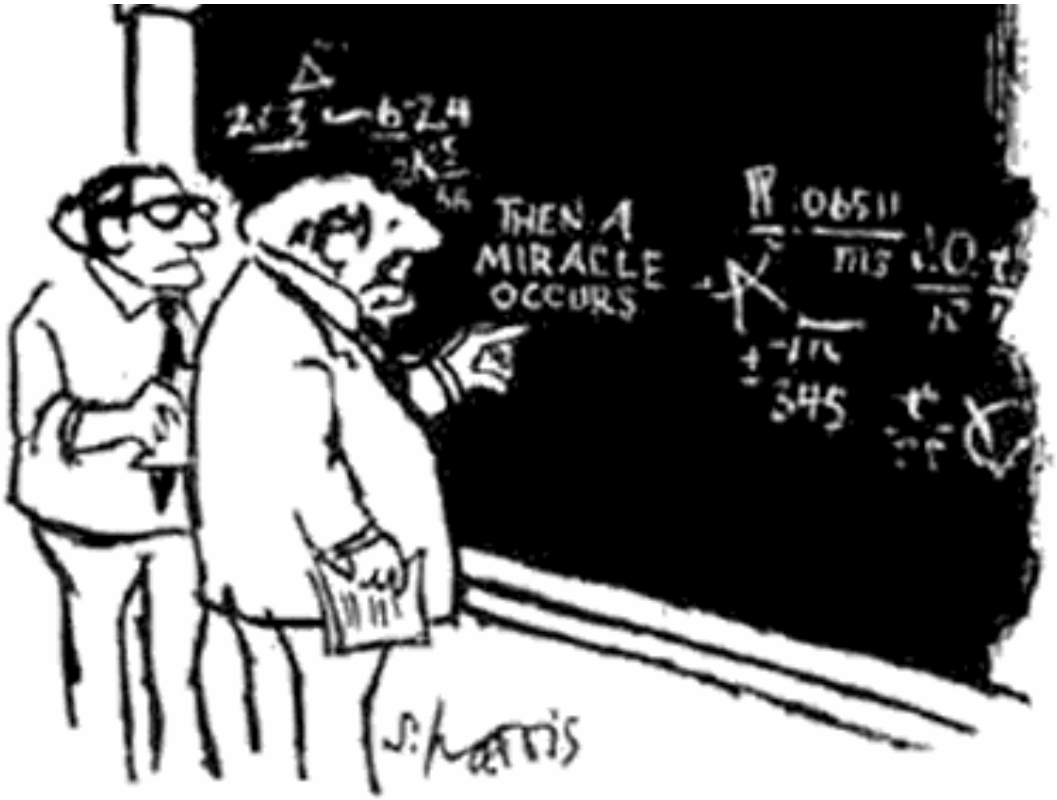
For the RBF, you're tuning the gamma. And you can go to infinite dimensions because you're never actually calculating those values as you only care about the dot-product.

So what is the excitement?

$\max_\alpha \sum$

s.t. α_i

$\sum$



arg r

s.t. y

"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

$(i)^T x^{(j)}$

So what is the excitement?

$$\max_\alpha \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j x^{(i)^T} x^{(j)}$$

$$\text{s.t. } \alpha_i \geq 0, \ i = 1, \ldots, m$$

$$\sum_{i=1}^m \alpha_i y^{(i)} = 0$$

$$K(x^{(i)}, x^{(j)})$$

You can get the dot product of the two, xi, xj by using the kernel.

That means you're not computing the higher dimensional values of them and saves you the computational costs.

$$\arg\min_{w,b} \frac{1}{2} ||w||^2$$

$$\text{s.t. } y^{(i)}(w^T x^{(i)} + b) \geq 1$$

Prediction

$$w^T x + b = \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b.$$

- Again we can use the kernel trick!
- Prediction speed depends on number of support vectors

So, the calculations preserved are the support vectors, and not the whole training set, like you have with KNN.

The Miracle Explained

- Andrew Ng does this really well

- http://cs229.stanford.edu/notes/cs229-notes3.pdf
- Course is also on Youtube, ItunesU, etc.

Kernel Trick for SVMs

- Arbitrary many dimensions
- Little computational cost
- Maximal margin helps with curse of dimensionality

Because you're trying to get the largest margin, it will generalize very well without the sparse distribution of points that a more dimensions incurs.

Face Recognition

Face Recognition

- Load image data
- Put your test data aside
- Extract Eigenfaces PCA on images
- Train SVM
- Evaluate performance

- Red are cross validation steps

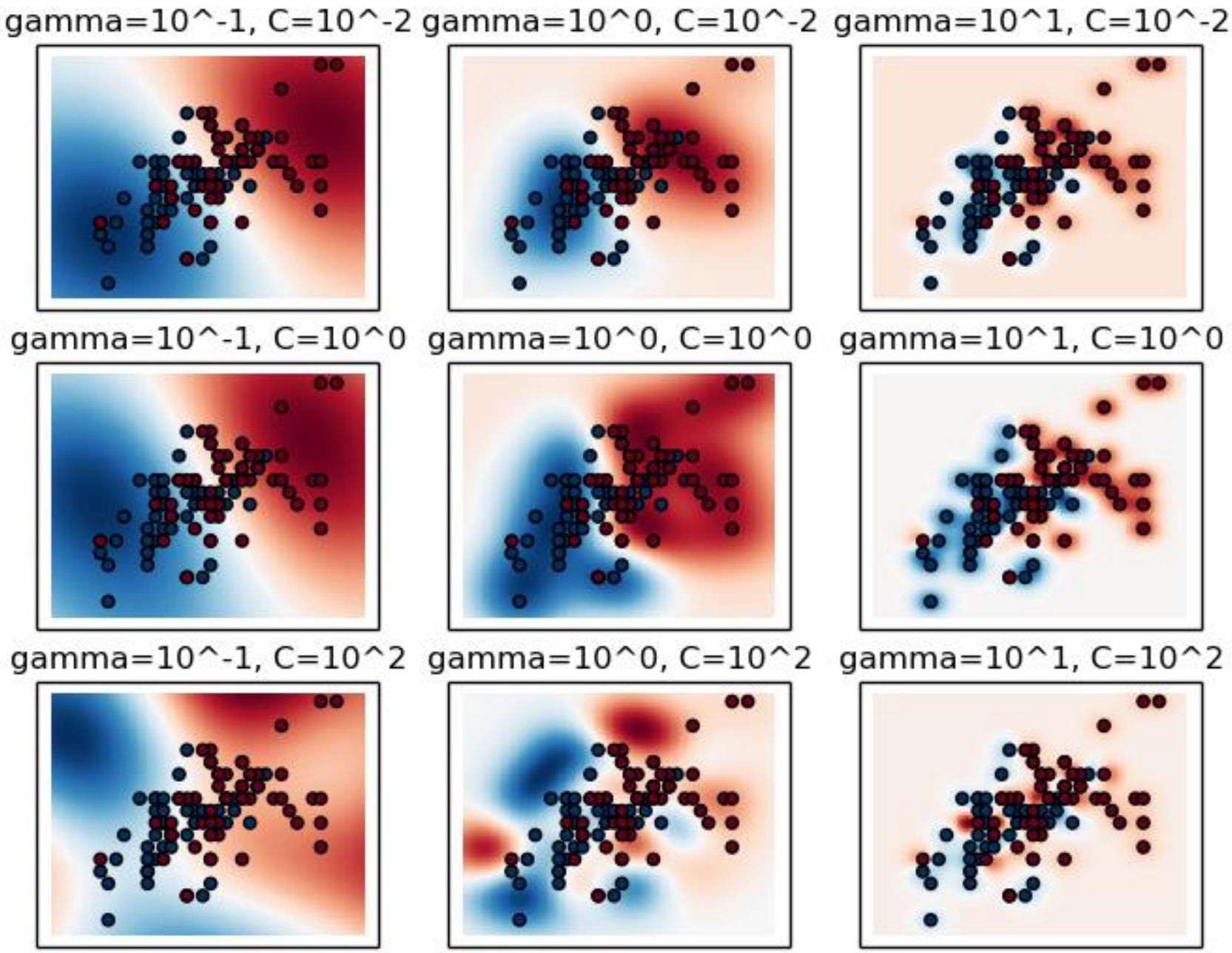
Parameters to tune:
- The # of components for your PCA
- Which kernel to use for your SVM & which parameters for that kernel (C for polynomial or gamma for Radial Boundary/RBF)

http://scikit-learn.org/stable/auto_examples/applications/face_recognition.html#example-applications-face-recognition-py

SVM_sign_language.mp4

Jhon Gonzalez

https://www.youtube.com/watch?v=cxHMgl2_5zg



http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

Tips and Tricks

- SVMs are not scale invariant

The features need to have similar scales, so if data is in miles and nanometers, it needs to be scaled.

- Check if your library normalizes by default

Most likely, it doesn't so you might need to care about it.

- Normalize your data
  - mean: 0 , std: 1
  - map to [0,1] or [-1,1]
- Normalize test set in same way!

Tips and Tricks

- RBF kernel is a good default

If you're not sure about your data and it's relationships, you can always try RBF.

- For parameters try exponential sequences
- Read:

  Chih-Wei Hsu et al., "A Practical Guide to Support Vector Classification", Bioinformatics (2010)

SVM vs KNN

- What are the main key differences?

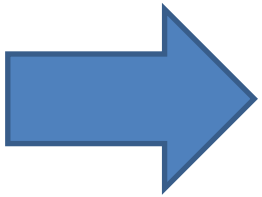
  With KNN you need to keep all the training data. There's no need to do this with SVMs, with it only caring about support vectors.

  KNN only have the K to tune, whereas with SVMs we have C and the parameters of our kernel to choose in the first place.

Parameter Tuning

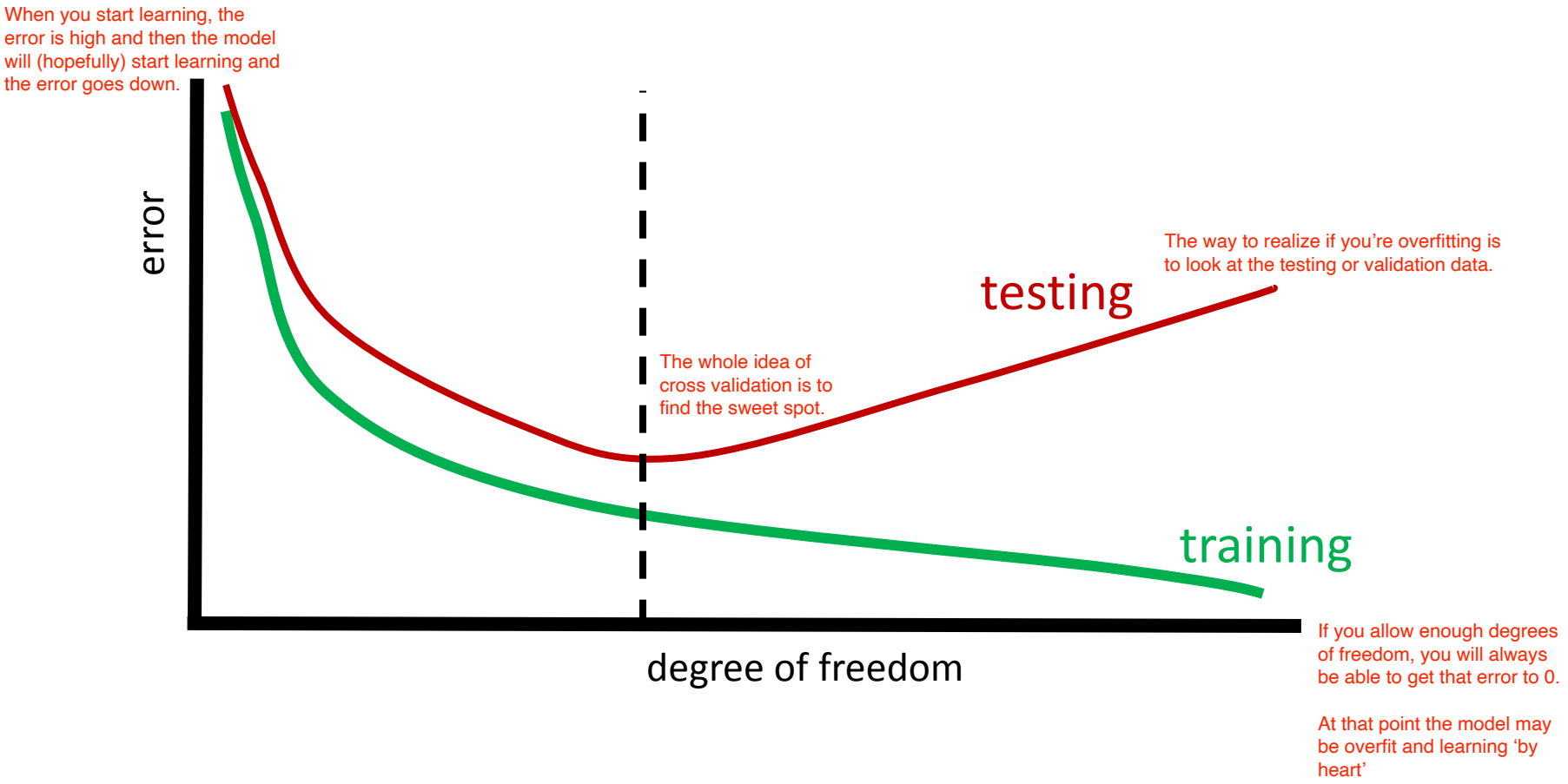
- Given a classification task

- Which kernel ?
- Which kernel parameter values?
- Which value for C?



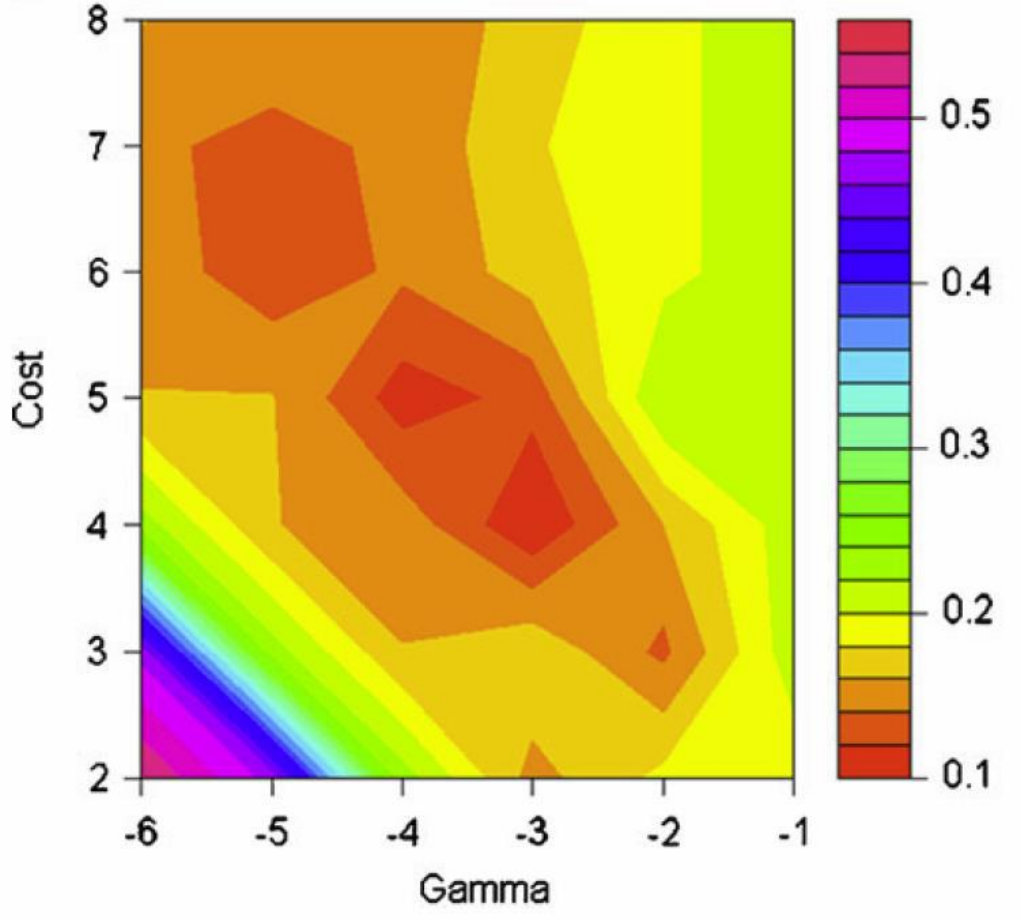
Try different combinations
and take the best.

Train vs. Test Error



Where is KNN on this graph for K=1, or for K=Inf?

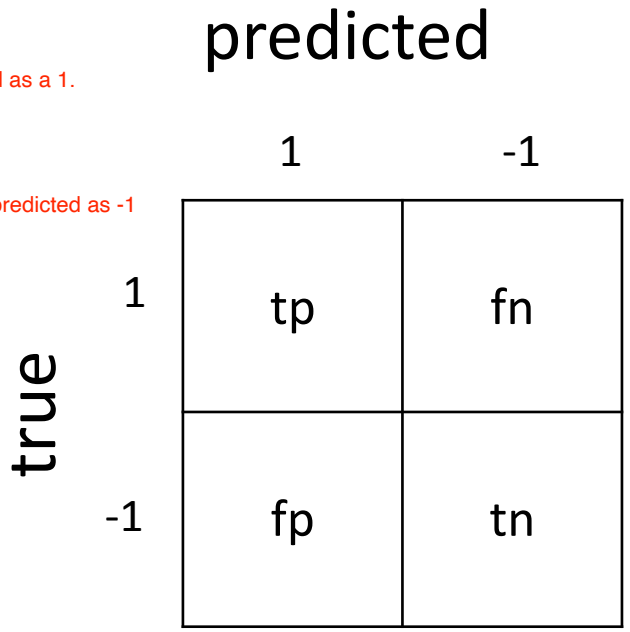
Grid Search



Zang et al., “Identification of heparin samples that contain impurities or contaminants by chemometric pattern recognition analysis of proton NMR spectral data”, Anal Bioanal Chem (2011)

Error Measures

- True positive (tp) data labeled as a 1 and predicted as a 1.
- True negative (tn) data labeled as -1 and model predicted as -1
- False positive (fp) data labeled as -1 but model predicted 1/positive
- False negative (fn) data labeled as true but model predicted as -1



TPR and FPR

- True Positive Rate:

$$\frac{tp}{tp+fn}$$

Of those labeled true, how many did model actually predict correctly.

predicted

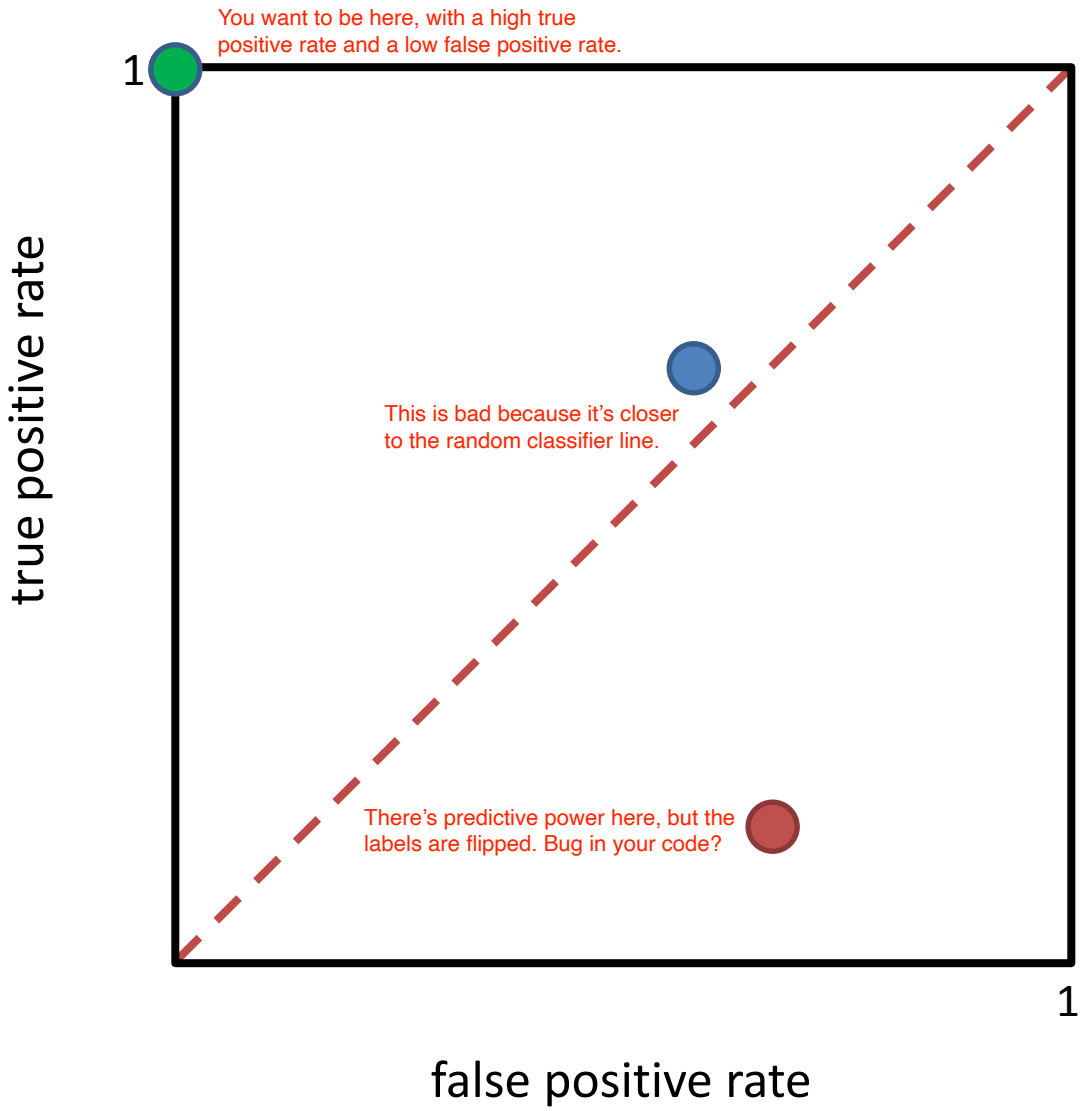
true \ predicted	1	-1
1	tp	fn
-1	fp	tn

- False Positive Rate:

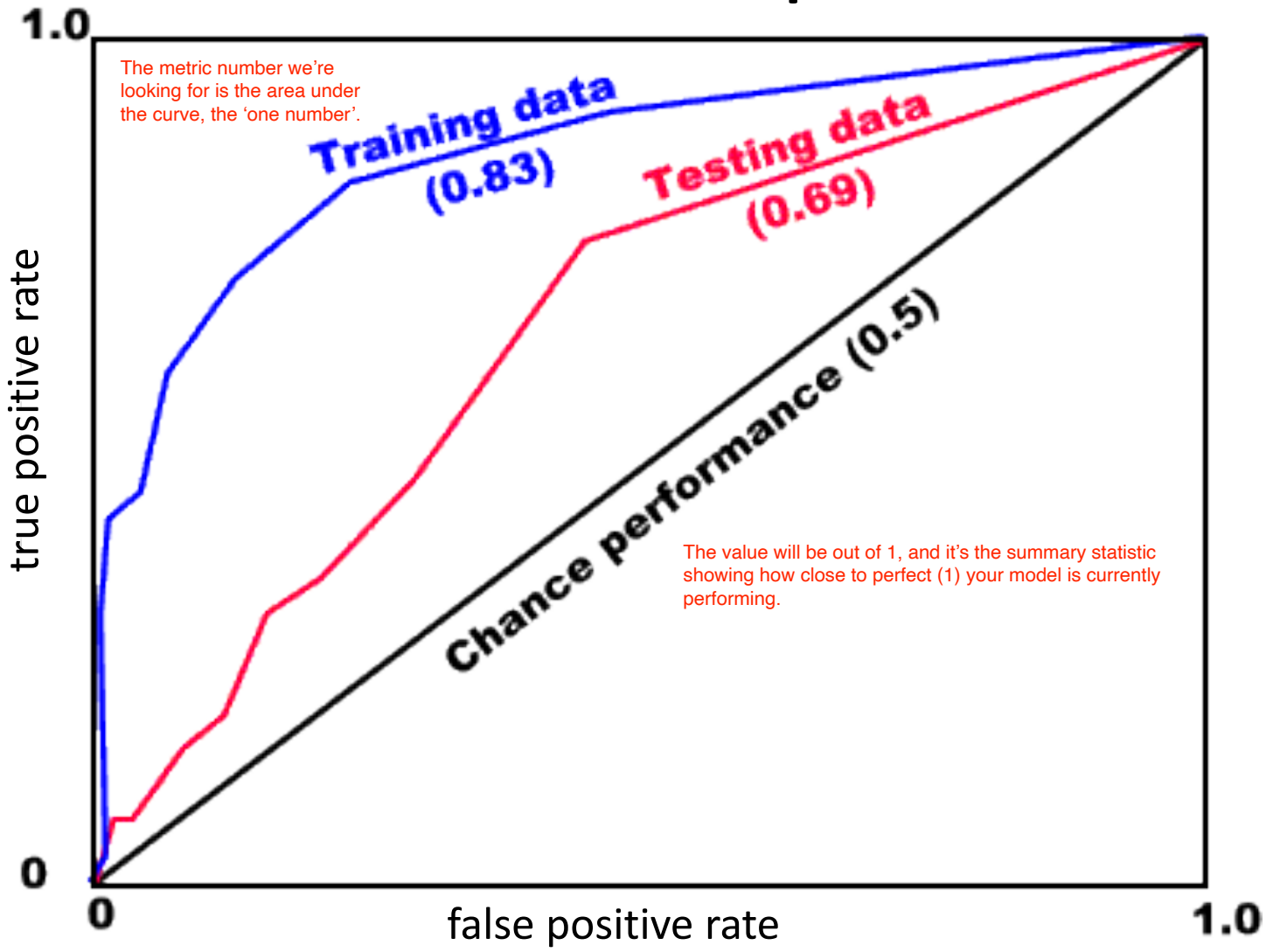
$$\frac{fp}{fp+tn}$$

everything labeled as negative class and looking at those that were mistook as positive examples.

Reciever Operating Characteristic



ROC Example



https://inclass.kaggle.com/c/ca-2015/details/evaluation

Precision Recall

- Recall: $\frac{tp}{tp+fn}$

This is the same as the True Positive Rate we saw before in ROC

- Precision: $\frac{tp}{tp+fp}$

This changes from what we saw under the False Positive Rate.

Here, we see the true positives we predicted out of the errors in predicting positives from data that was actually labelled negative.

If you have an imbalanced dataset, your positive data is only a smaller position of your data than the negative, this is a better option as you'll have many more true negatives.

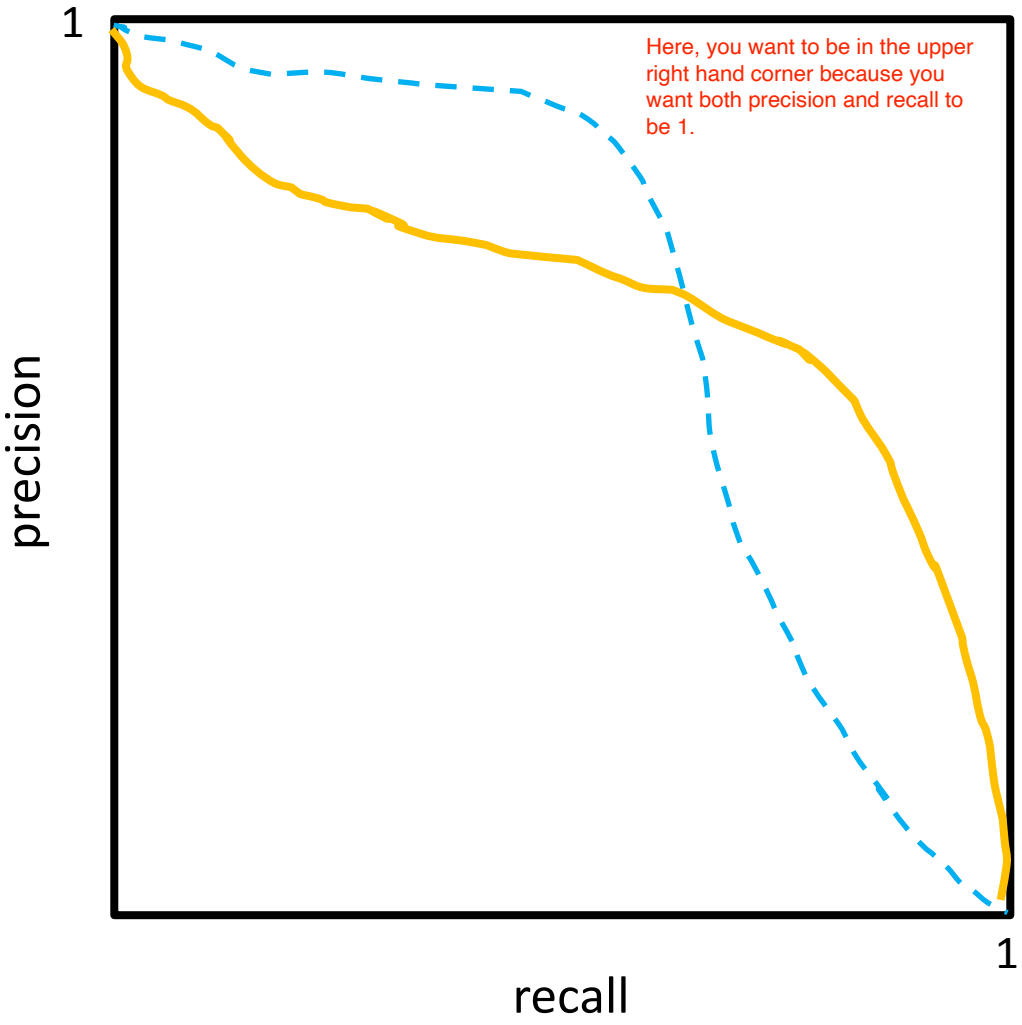
		predicted	
		1	-1
true	1	tp	fn
	-1	fp	tn

Precision Recall

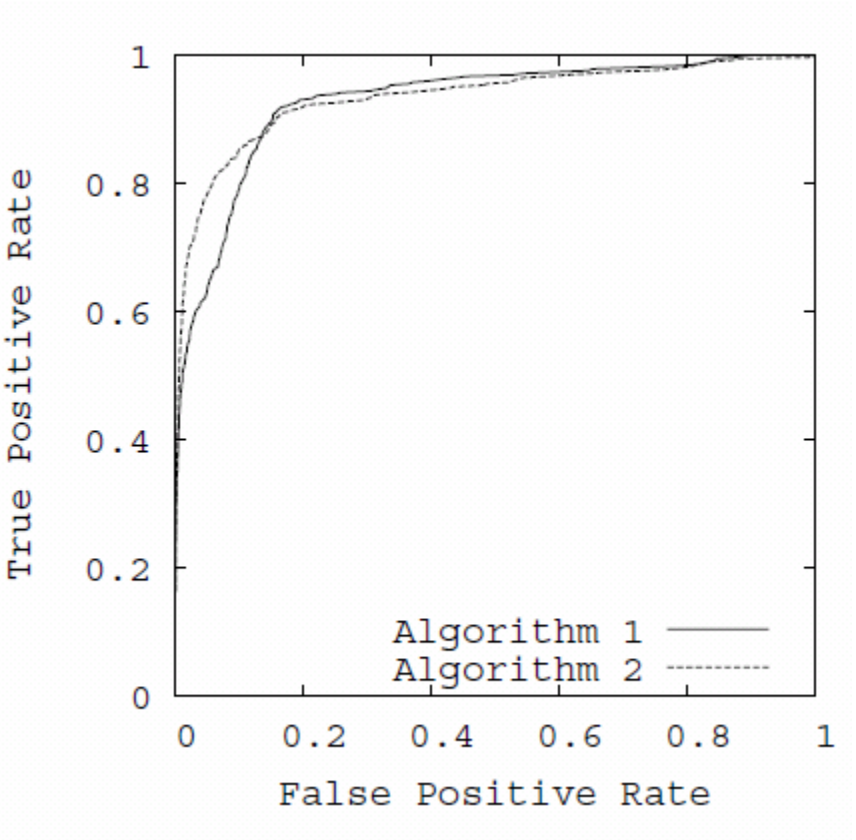
- **Recall**: If I pick a random positive example, what is the probability of making the right prediction?

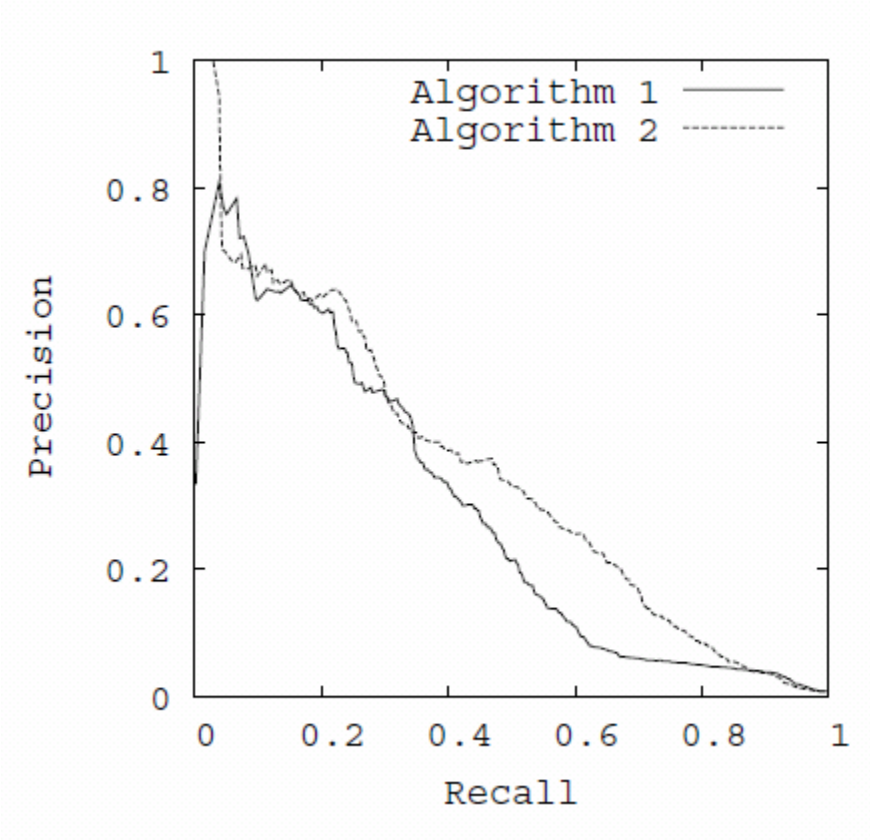
- **Precision**: If I take a positive prediction example, what is the probability that it is indeed a positive example?

Precision Recall Curve



Comparison





J. Davis & M. Goadrich,
"The Relationship Between Precision-Recall and ROC Curves.",
ICML (2006)

F-measure

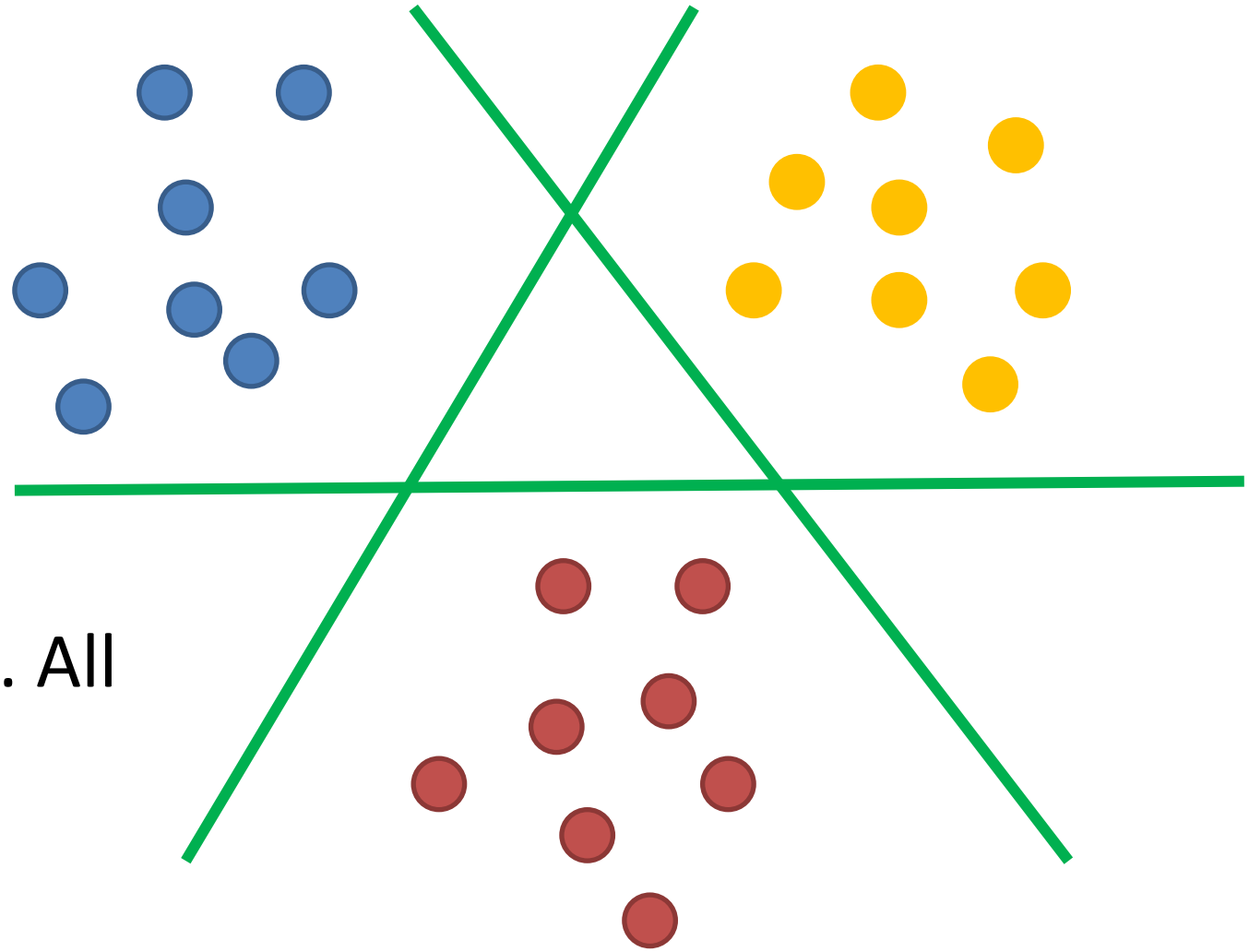
Just like there's the area under the curve for ROC, there's a summary statistic for precision recall curves called F-measure or the harmonic mean of the curve.

- Weighted average of precision and recall

$$F_\beta = \frac{(\beta^2+1)\cdot P \cdot R}{\beta^2 \cdot P + R}$$

- Usual case: $\beta = 1$
- Increasing β allocates weight to recall

Multi Class

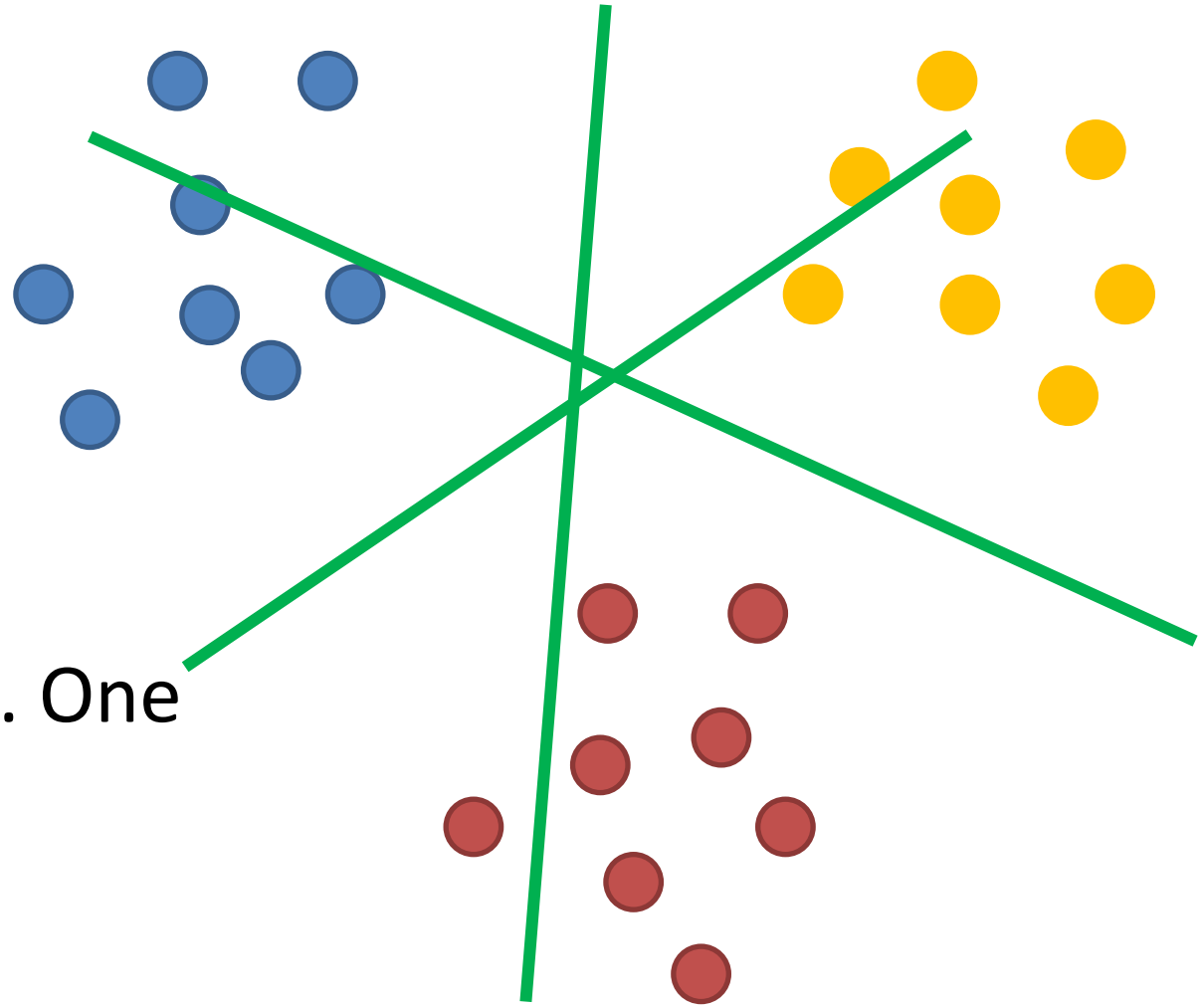


One vs. All

One vs All

- Train n classifier for n classes
- Take classification with greatest margin
- Slow training

Multi Class

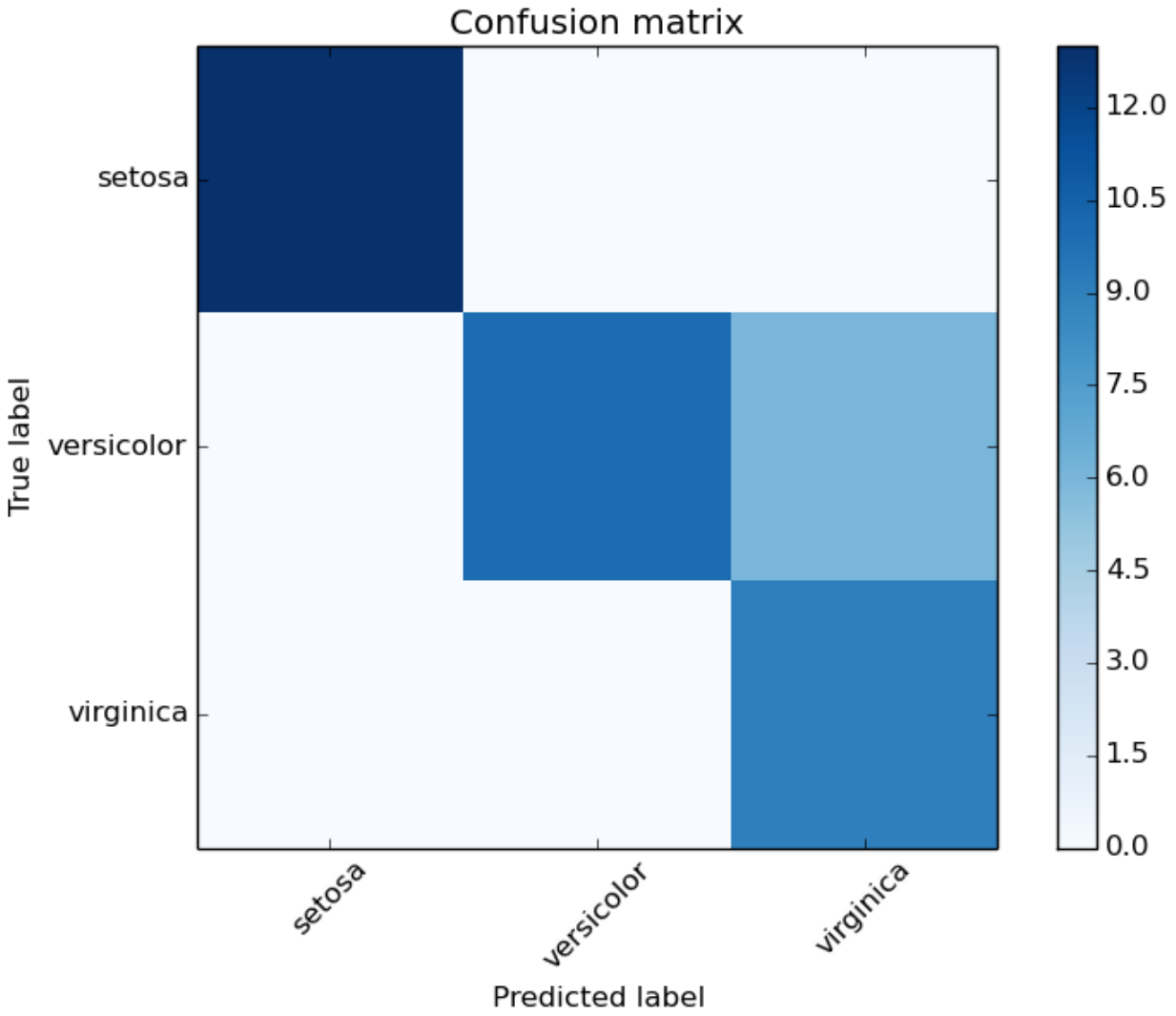


One vs. One

One vs One

- Train n(n-1)/2 classifiers
- Take majority vote
- Fast training

Confusion Matrix



http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

Recap

- Perceptrons are great
- But really just a separating hyperplane
- So is SVM
- Kernels are neat
- Evaluation metrics are important