

CPSC 340: Machine Learning and Data Mining

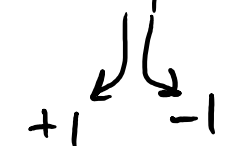
More Linear Classifiers

Last Time: Classification using Regression and SVMs

- Binary classification using sign of linear models:

Fit model $o_i = w^T x_i$ and predict using $\text{sign}(o_i)$

+1 -1



- We considered different training “error” functions:

- Squared error: $(o_i - y_i)^2$.

- If $y_i = +1$ and $o_i = +100$, then squared error $(o_i - y_i)^2$ is huge.

- 0-1 classification error: $(\text{sign}(o_i) = y_i)?$

- Ideal error but non-convex and non-continuous so hard to minimize in terms of ‘w’.
- Optimization only tractable if perfect classifier exists → perceptron algorithm.

- Hinge loss: $\max\{0, 1 - y_i o_i\}$.

- Convex upper bound on number of classification errors.
- With L2-regularization, it’s called a support vector machine (SVM).

Logistic Loss

- We can alternately smooth the degenerate loss with log-sum-exp:

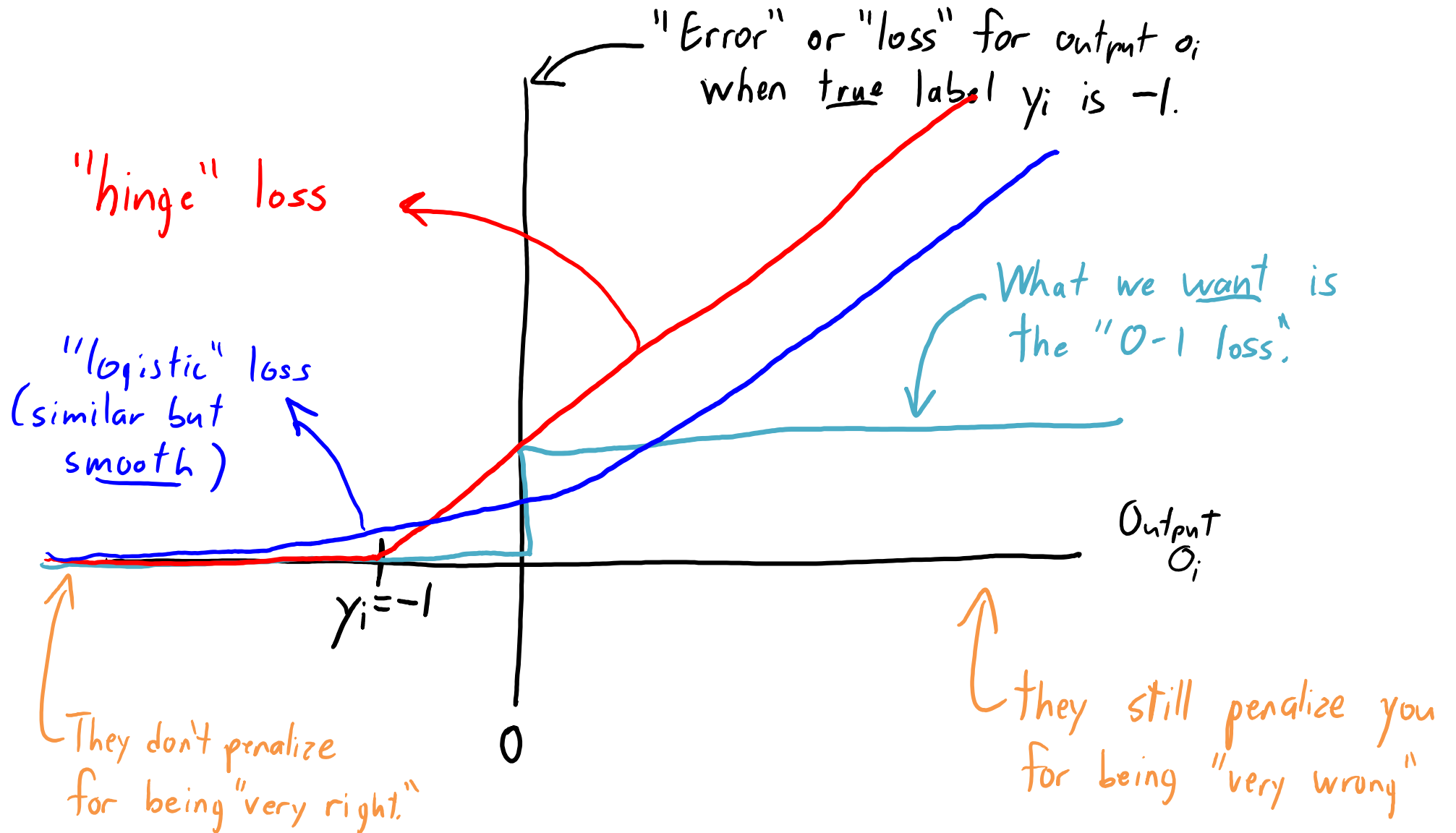
$$\max\{0, -y_i o_i\} \approx \log(\underbrace{\exp(0)}_1 + \exp(-y_i o_i))$$

- Summing over all examples gives:

$$f(w) = \sum_{i=1}^n \log(1 + \exp(-y_i \underbrace{w^T x_i}_{o_i}))$$

- This is the “logistic loss” and model is called “logistic regression”.
 - It’s not degenerate: $w=0$ now gives an error of $\log(2)$ instead of 0.
 - Convex and differentiable: minimize this with gradient descent.
 - You should also add regularization.
 - We will see later that it has a probabilistic interpretation.

Convex Approximations to 0-1 Loss



Logistic Regression and SVMs

- Logistic regression and SVMs are used EVERYWHERE!
 - Fast training and testing.
 - Training on huge datasets using “stochastic” gradient descent (next week).
 - Prediction is just computing $w^T x_i$ and then taking sign.
 - Weights w_j are easy to understand.
 - It’s how much w_j changes the prediction and in what direction.
 - We can often get a good test error.
 - With low-dimensional features using RBFs and regularization.
 - With high-dimensional features and regularization.
 - Smoother predictions than random forests.

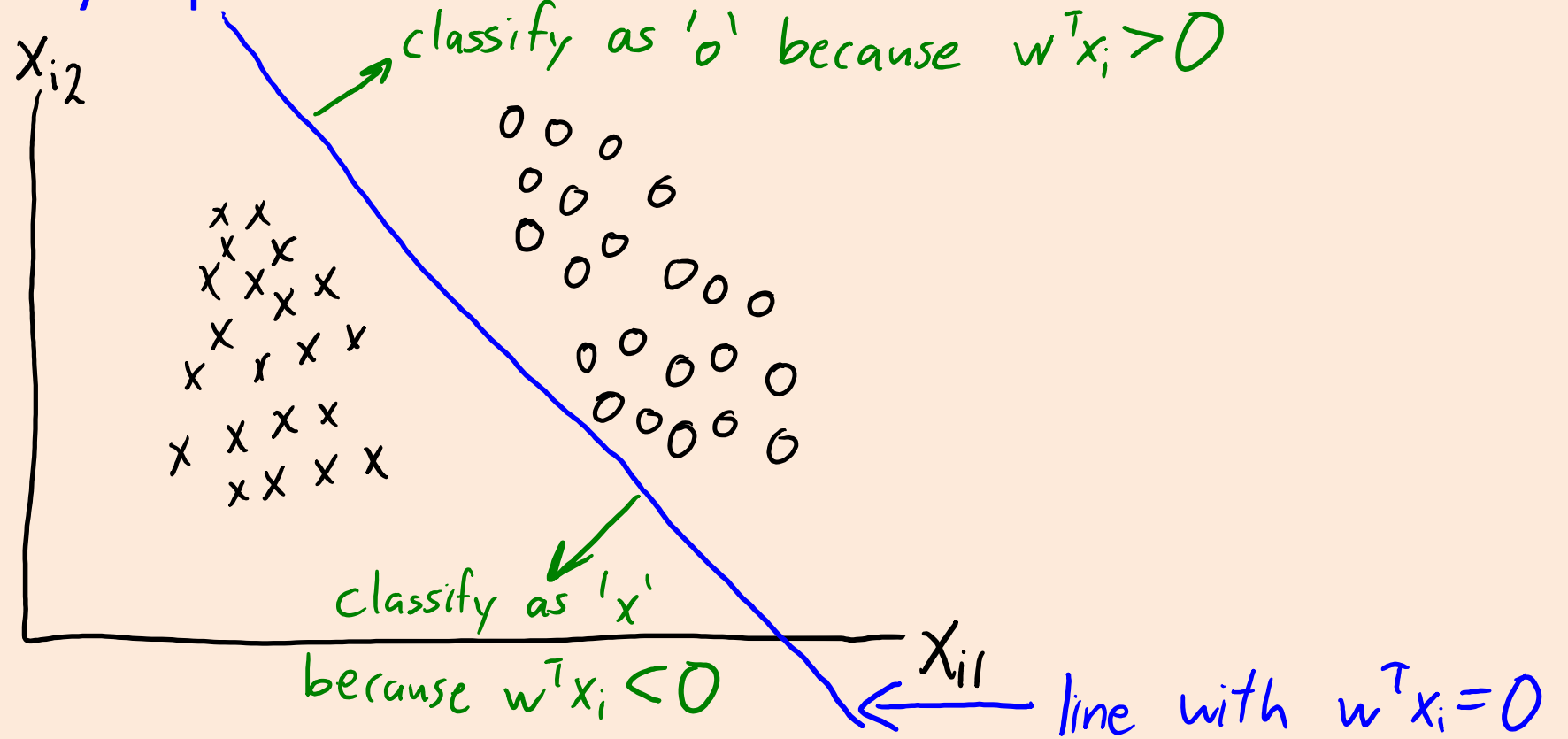
Comparison of “Black Box” Classifiers

- Fernandez-Delgado et al. [2014]:
 - “Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?”
- Compared 179 classifiers on 121 datasets.
- Random forests are most likely to be the best classifier.
- Next best class of methods was SVMs (L2-regularization, RBFs).
- “Why should I care about logistic regression if I know about deep learning?”

Next Topic: Maximizing the Margin

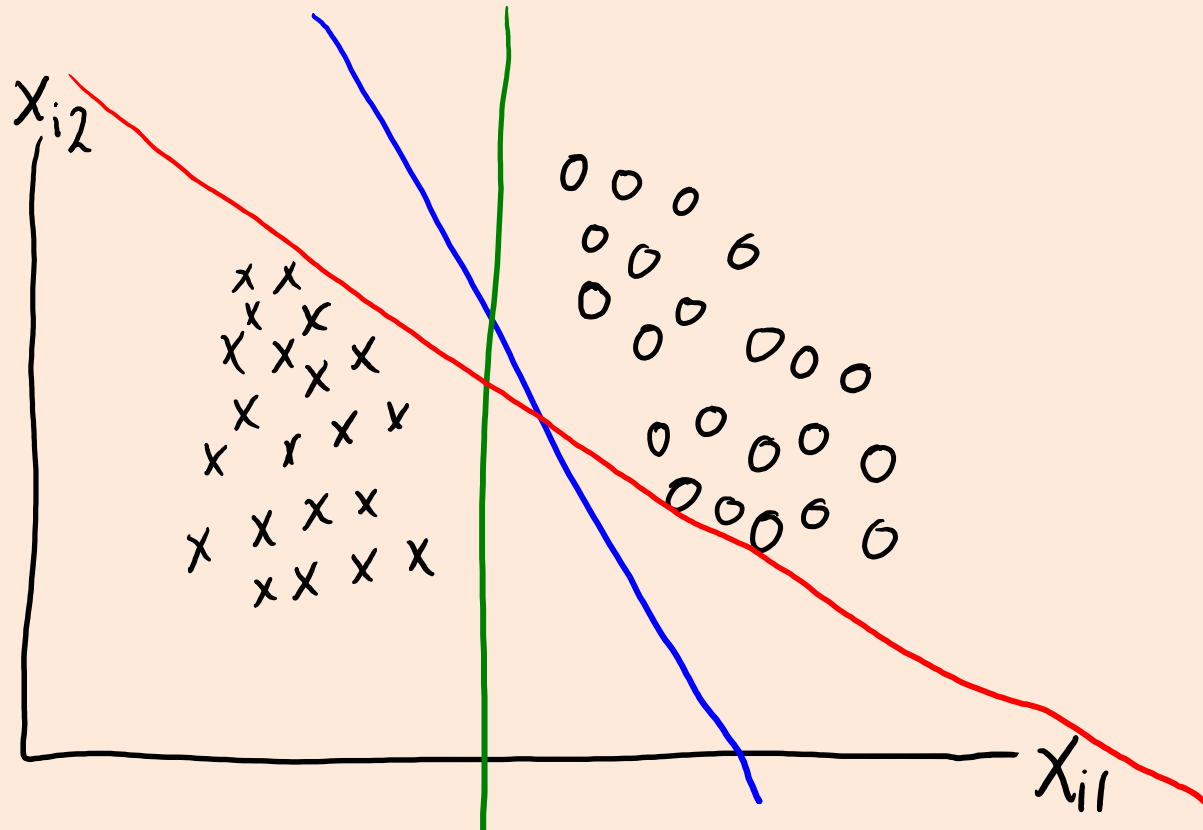
Maximum-Margin Perspective

- Consider a **linearly-separable** dataset.



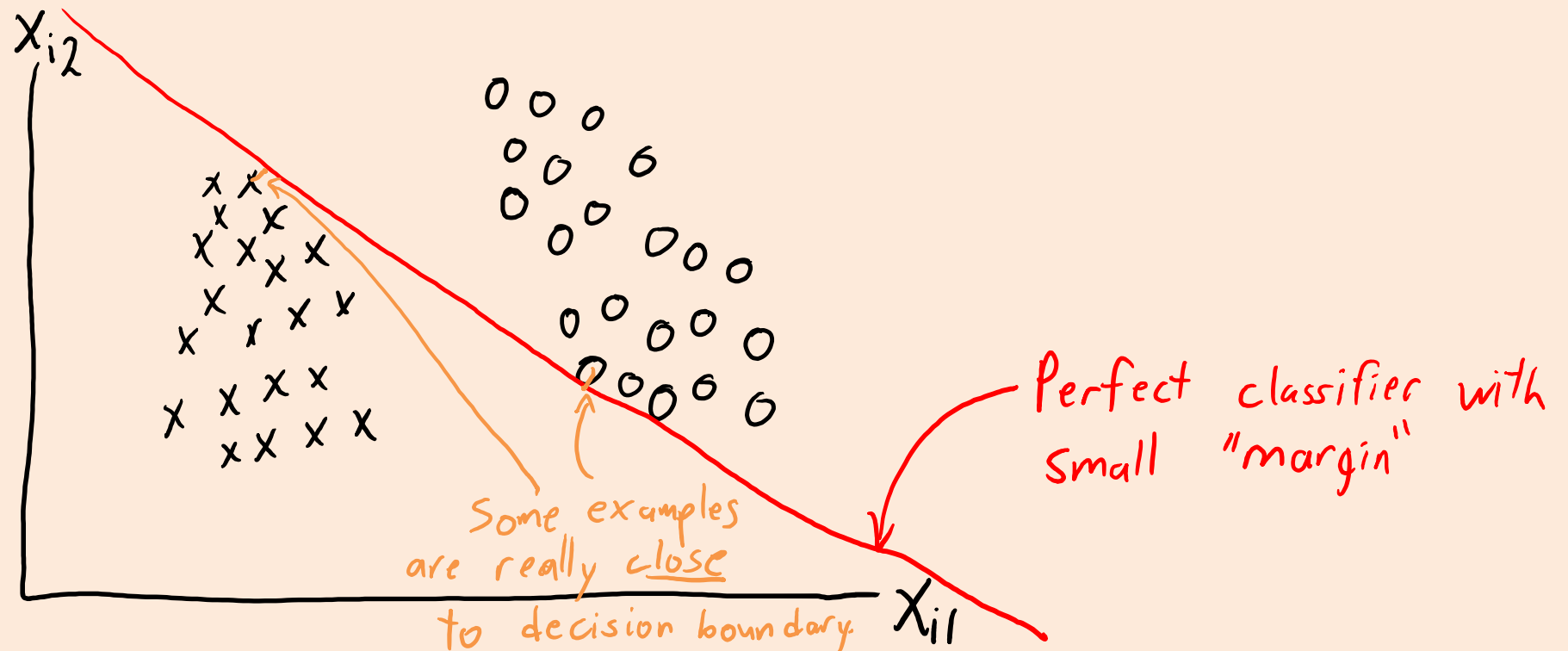
Maximum-Margin Perspective

- Consider a **linearly-separable** dataset.
 - **Perceptron algorithm** finds *some* classifier with zero error.
 - But are all **zero-error classifiers** equally good?



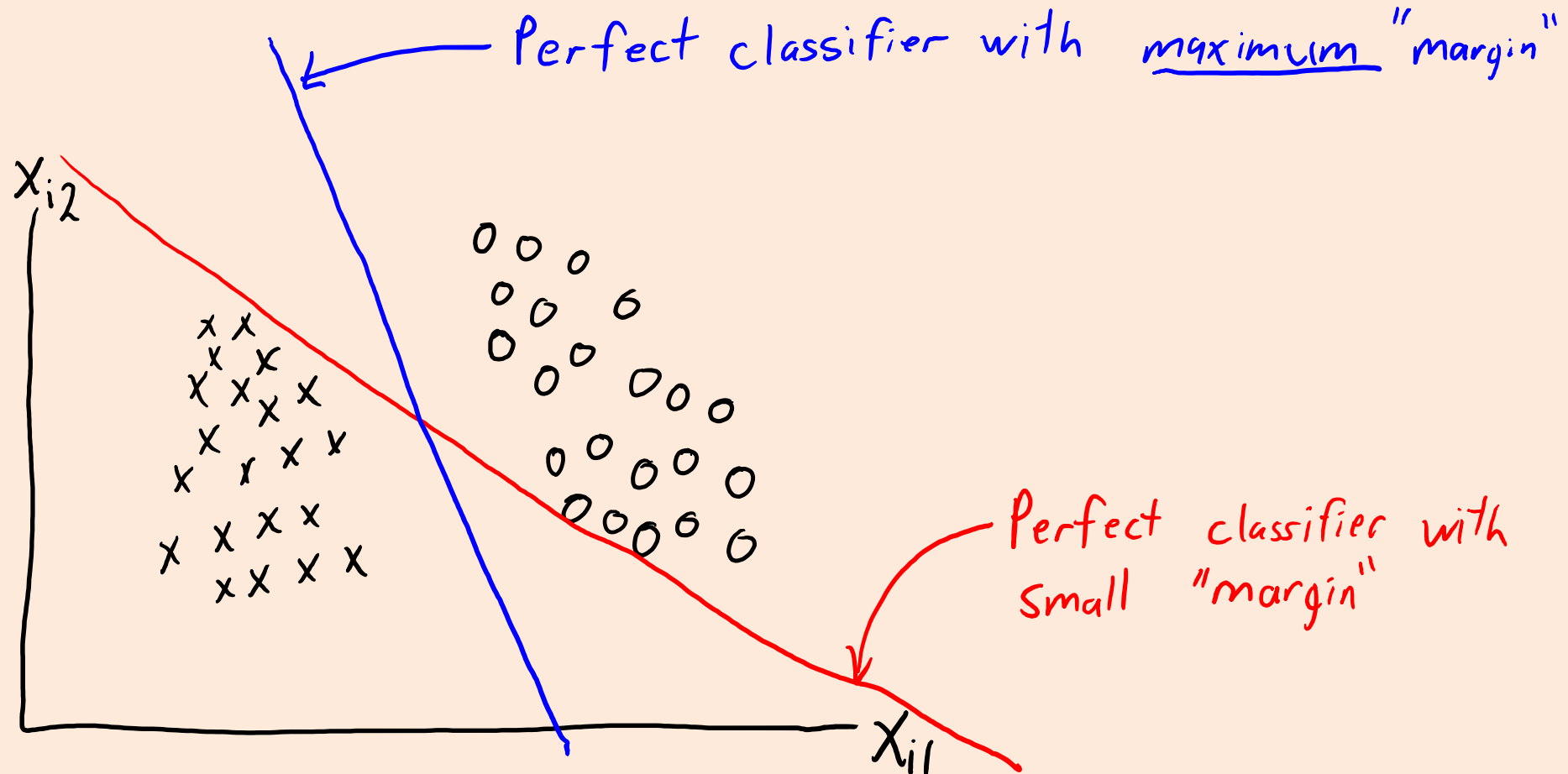
Maximum-Margin Perspective

- Consider a linearly-separable dataset.
 - **Maximum-margin** classifier: choose the farthest from both classes.



Maximum-Margin Perspective

- Consider a linearly-separable dataset.
 - Maximum-margin classifier: choose the farthest from both classes.

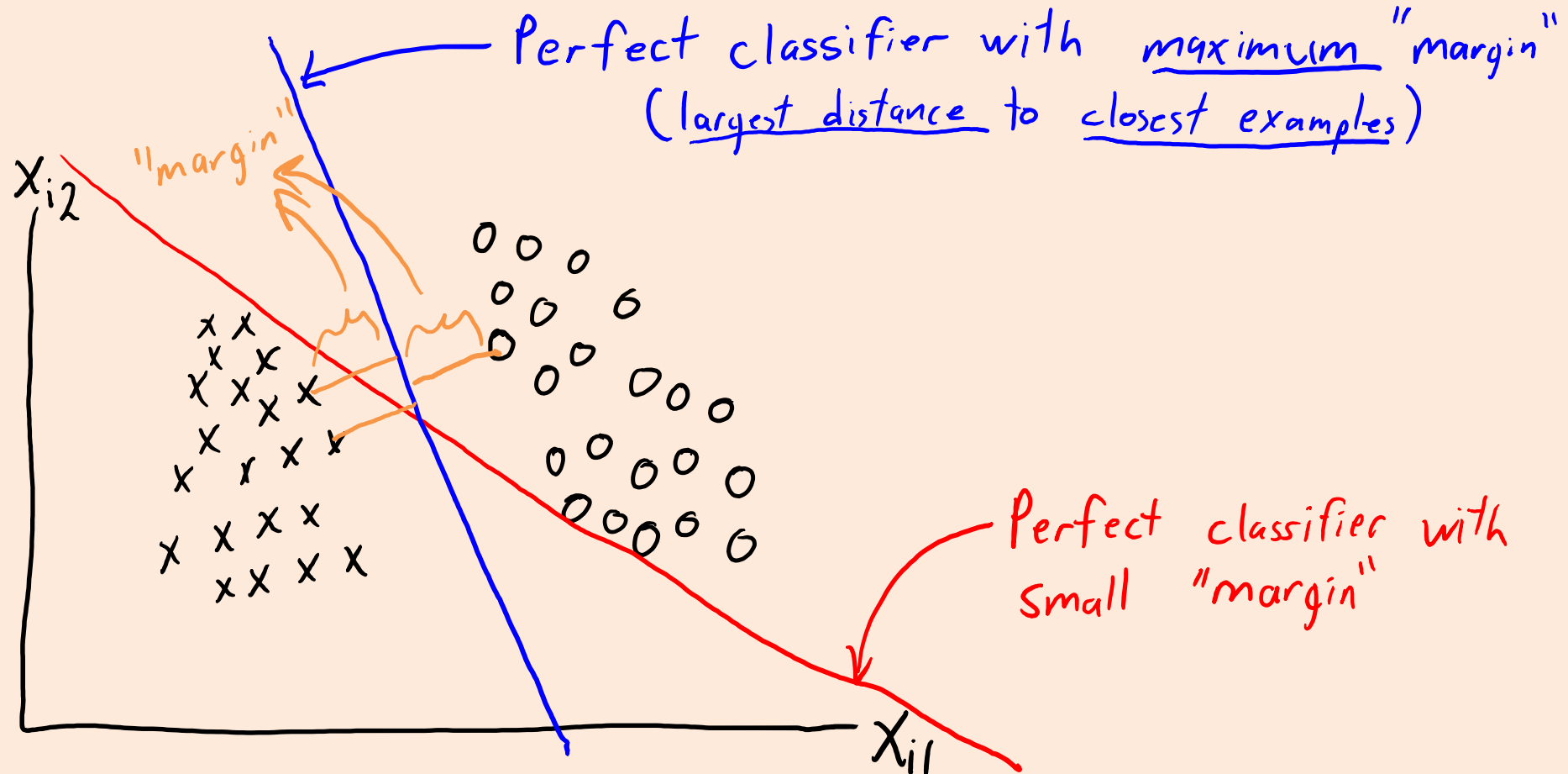


Maximum-Margin Perspective

- Consider a linearly-separable dataset.
 - Maximum-margin classifier: choose the farthest from both classes.

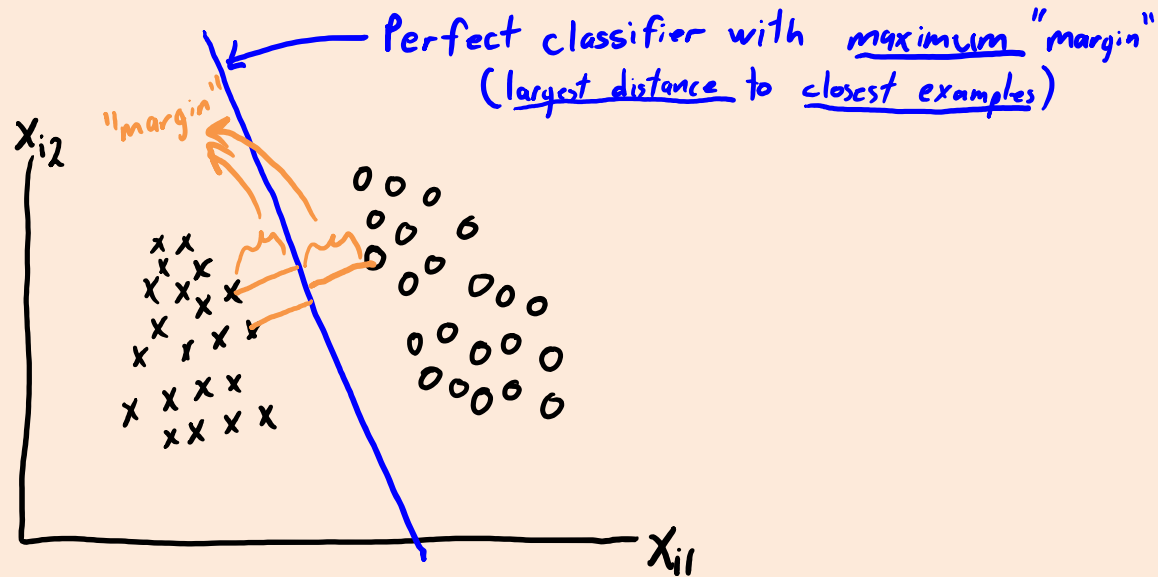
Why maximize margin?

If test data is close to training data, then max margin leaves more "room" before we make an error.



Maximum-Margin Perspective

- For **linearly-separable** data:



- With small-enough $\lambda > 0$, SVMs find the maximum-margin classifier.
 - Need λ small enough that hinge loss is 0 in solution.
 - Origin of the name: the “support vectors” are the points closest to the line (see bonus).
- Recent result: logistic regression also finds maximum-margin classifier.
 - With $\lambda=0$ and if you fit it with gradient descent (not true for many other optimizers).

Next Topic: Multi-Class Linear Classifiers

Multi-Class Linear Classification

- We have been considering **linear models for binary classification**:

$$X = \begin{bmatrix} \\ \\ \\ \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

- E.g., is there a cat in this image or not?



Multi-Class Linear Classification

- Today we will discuss **linear models for multi-class classification**:

$$X = \begin{bmatrix} \\ \\ \\ \\ \\ \end{bmatrix} \quad y = \begin{bmatrix} 27 \\ 16 \\ 8 \\ 7 \\ 21 \\ 5 \end{bmatrix}$$

- For example, classify image as “cat”, “dog”, or “person”.
 - This was natural for methods of Part 1 (decision trees, naïve Bayes, KNN).
 - For linear models, we need some new notation.

“One vs All” Classification

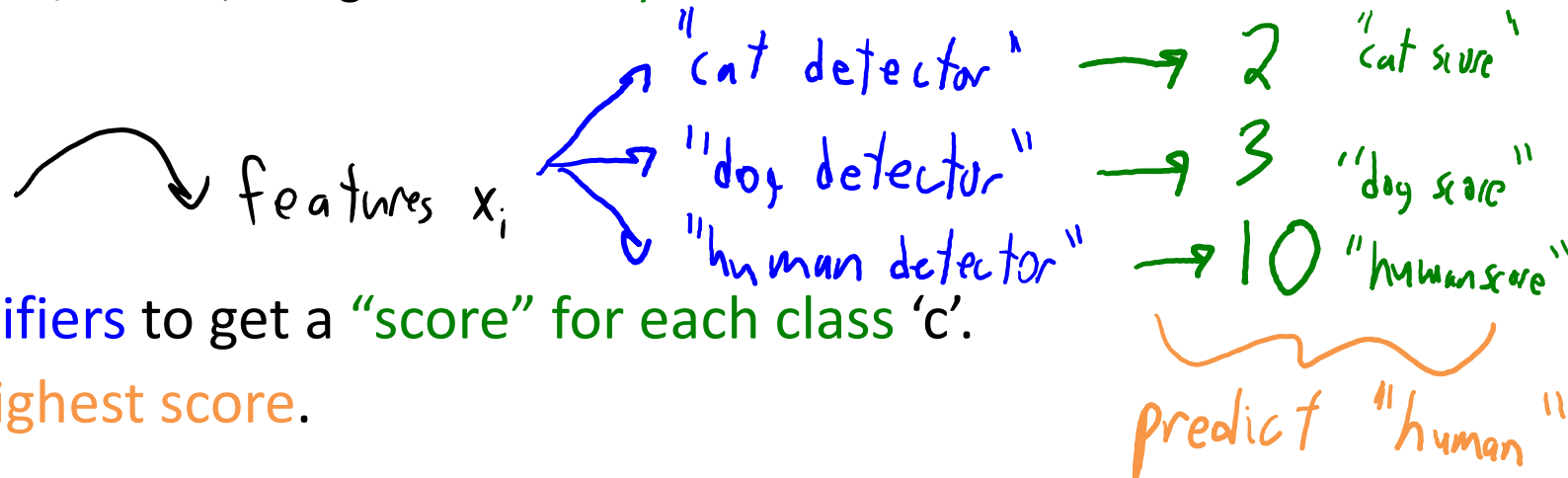
- Suppose you **only know how to do binary classification**:
 - “One vs all” is a way to **turn a binary classifier into a multi-class method**.

- **Training phase:**

- For each class ‘c’, **train binary classifier to predict whether example is a ‘c’**.
 - For example, train a “cat detector”, a “dog detector”, and a “human detector”.
 - If we have ‘ ℓ ’ possible labels/classes, this gives ‘ ℓ ’ **binary classifiers**.

- **Prediction phase:**

- Apply the ‘ ℓ ’ binary classifiers to get a “score” for each class ‘c’.
- Predict the ‘c’ with the highest score.



“One vs All” Linear Classification

- “One vs all” logistic regression for classifying as cat/dog/person.
 - Train a separate classifier for each class.
 - Classifier 1 tries to predict +1 for “cat” images and -1 for “dog” and “person” images.
 - Classifier 2 tries to predict +1 for “dog” images and -1 for “cat” and “person” images.
 - Classifier 3 tries to predict +1 for “person” images and -1 for “cat” and “dog” images.
 - This gives us a weight vector w_c for each class ‘c’:
 - Output $o_{ic} = w_c^T x_i$ tries to have $\text{sign}(o_{ic}) = +1$ for true class ‘c’.
 - And $\text{sign}(o_{ic'}) = -1$ for other classes c' .
 - We’ll use ‘W’ as a matrix with the w_c as rows, and we have $k = \ell$ rows (one per class).

$$W = \begin{bmatrix} \text{---} w_1^T \text{---} \\ \text{---} w_2^T \text{---} \\ \vdots \\ \text{---} w_k^T \text{---} \end{bmatrix} \left. \vphantom{\begin{bmatrix} \text{---} w_1^T \text{---} \\ \text{---} w_2^T \text{---} \\ \vdots \\ \text{---} w_k^T \text{---} \end{bmatrix}} \right\} k$$

$\underbrace{\hspace{10em}}_d$

→ Each row ‘c’ gives weights w_c for a binary logistic regression model to predict class ‘c’.

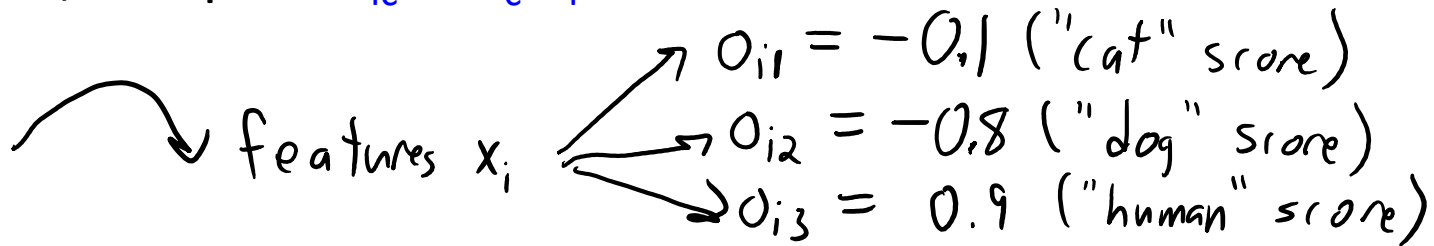
“One vs All” Linear Classification

- “One vs all” logistic regression for classifying as cat/dog/person.
 - Prediction on example x_i given parameters ‘W’ :

$$W = \begin{bmatrix} \text{---} w_1^T \text{---} \\ \text{---} w_2^T \text{---} \\ \vdots \\ \text{---} w_K^T \text{---} \end{bmatrix} \left. \vphantom{\begin{bmatrix} \text{---} w_1^T \text{---} \\ \text{---} w_2^T \text{---} \\ \vdots \\ \text{---} w_K^T \text{---} \end{bmatrix}} \right\} K$$

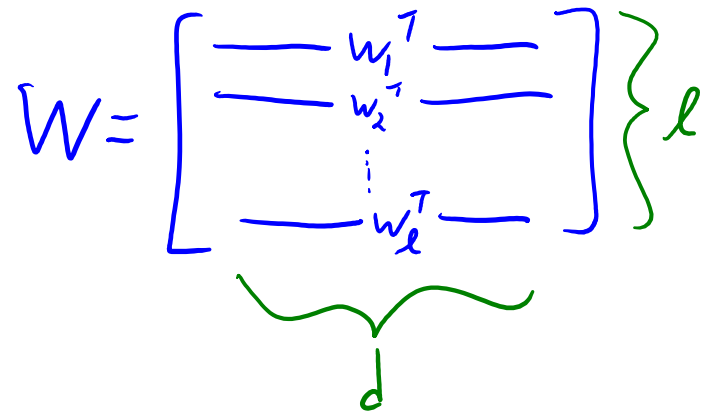
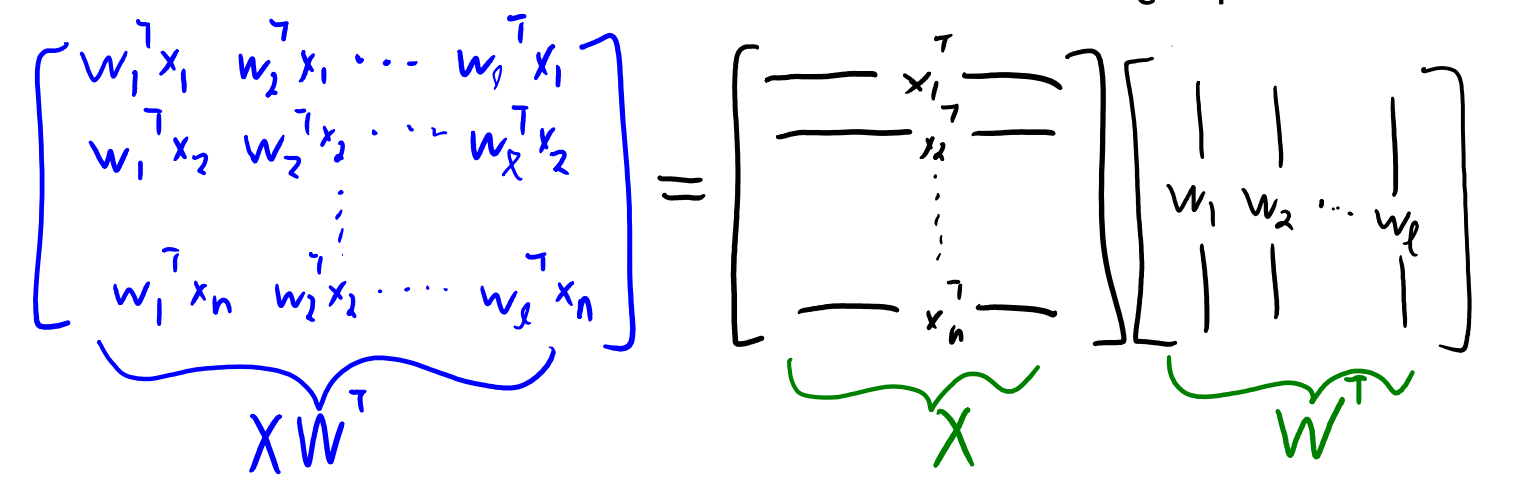
d

- For each class ‘c’, compute $o_{ic} = w_c^T x_i$.



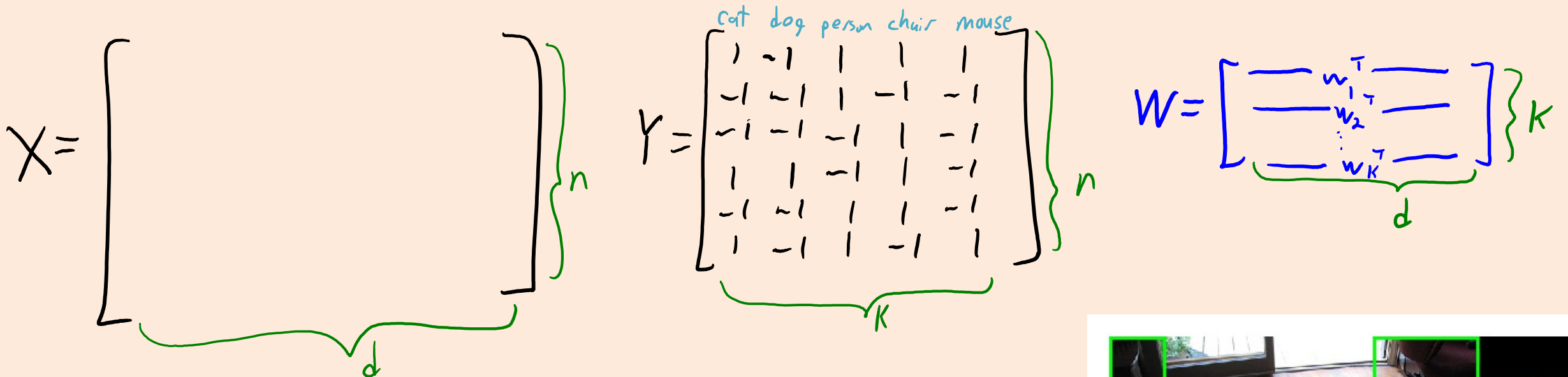
- Ideally, we get $\text{sign}(o_{ic}) = +1$ for one class ‘c’ and $\text{sign}(o_{ic'}) = -1$ for all other classes c’.
- In practice, it **might be +1 for multiple classes or no class**.
- To predict class, we take **maximum value of o_{ic}** (“highest score”).
 - In the example above, predict “human” (0.9 is higher than -0.8 and -0.1).

Multi-Class Linear Prediction in Matrix Notation

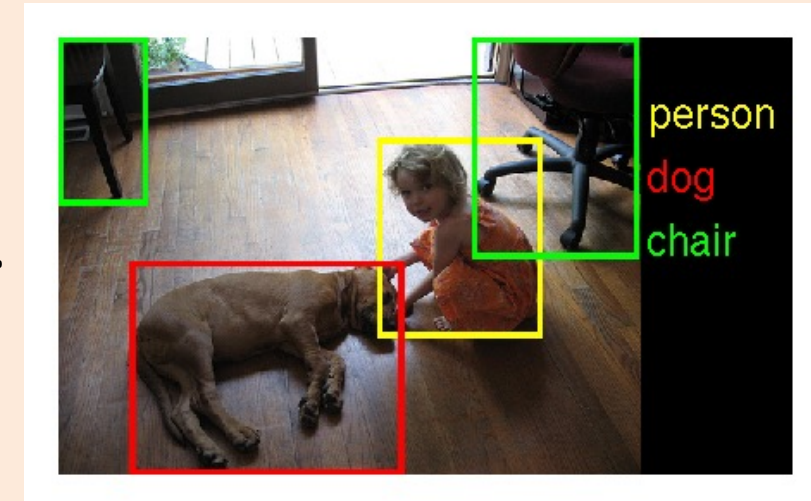
- In multi-class linear classifiers our weights are: $W = \begin{bmatrix} \text{---} w_1^T \text{---} \\ \text{---} w_2^T \text{---} \\ \vdots \\ \text{---} w_\ell^T \text{---} \end{bmatrix}$ $\left. \vphantom{\begin{bmatrix} \text{---} w_1^T \text{---} \\ \text{---} w_2^T \text{---} \\ \vdots \\ \text{---} w_\ell^T \text{---} \end{bmatrix}} \right\} \ell$

- To predict on all training examples, we first compute all $w_c^T x_i$.
 - Or in **matrix notation**: $\begin{bmatrix} w_1^T x_1 & w_2^T x_1 & \dots & w_\ell^T x_1 \\ w_1^T x_2 & w_2^T x_2 & \dots & w_\ell^T x_2 \\ \vdots & \vdots & \ddots & \vdots \\ w_1^T x_n & w_2^T x_n & \dots & w_\ell^T x_n \end{bmatrix} = \begin{bmatrix} \text{---} x_1^T \text{---} \\ \text{---} x_2^T \text{---} \\ \vdots \\ \text{---} x_n^T \text{---} \end{bmatrix} \begin{bmatrix} | & | & \dots & | \\ w_1 & w_2 & \dots & w_\ell \\ | & | & \dots & | \end{bmatrix}$

- So **predictions are maximum column indices of XW^T** (which is 'n' by 'l').

Digression: Multi-Label Classification

- A related problem is **multi-label classification**:



- Which of the ' ℓ ' objects are in this image?
 - There may be more than one "correct" class label.
 - Here we can also fit ' ℓ ' binary classifiers.
 - But we would take all the $\text{sign}(o_{ic})=+1$ as the labels.



Multi-Class Linear Classification (MEMORIZE)

- Notation for multi-class linear classifiers:

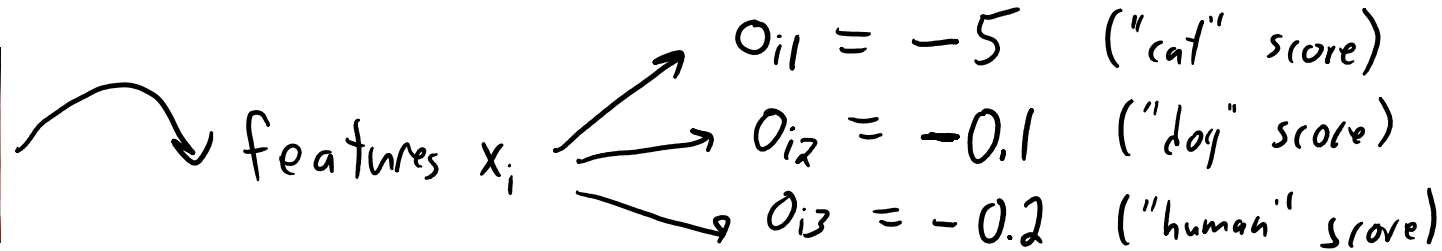
The diagram illustrates the notation for multi-class linear classification. It shows three main components:

- Feature Matrix X :** A large vertical bracket on the left is labeled $X =$. A green curly brace to its right indicates it has n rows. A green curly brace below it indicates it has d columns.
- Target Vector y :** A vertical vector of numbers $\begin{bmatrix} 27 \\ 16 \\ 8 \\ 7 \\ 21 \\ 5 \end{bmatrix}$ is shown. A green curly brace to its right indicates it has n elements. A green curly brace below it indicates it has 1 column.
- Weight Matrix W :** A matrix with l rows and d columns is shown. The rows are labeled $w_1^T, w_2^T, \dots, w_l^T$. A green curly brace to its right indicates it has l rows. A green curly brace below it indicates it has d columns. A red arrow points to the rows with the text: "each row 'c' of W is linear classifier for class 'c'."

- We'll use ' w_{y_i} ' as classifier where $c=y_i$ (weights of correct class).
 - So if $y_i=2$ then $w_{y_i} = w_2$, and $o_{y_i} = o_{ic}$.
- Similar matrix ' W ' used in k-means (here $k=l$, number of classes).

“One vs All” Multi-Class Linear Classification

- Problem: We **didn't train the w_c so that $c=y_i$ would maximize o_{ic}** .
 - Each classifier is **just trying to get the sign right**.



- Here the classifier incorrectly predicts “dog”.
 - “One vs All” **doesn't try to put o_{i2} and o_{i3} on same scale** for decisions like this.
 - We should **try to make o_{i3} positive and o_{i2} negative relative to each other**.
 - The **multi-class hinge losses** and the **multi-class logistic loss** do this.

Multi-Class SVMs

- Can we define a **loss that encourages $c=y_i$ to maximize o_{ic}** ?
 - So when we maximizing over o_{ic} , we **choose correct label y_i** .
- Recall our derivation of the **hinge loss** (SVMs) with **one output o_i** .
 - We **wanted $y_i o_i > 0$** for all 'i' to classify correctly.
 - We avoided **non-degeneracy** by aiming for $y_i o_i \geq 1$.
 - We used the **constraint violation** as our loss: $\max\{0, 1 - y_i o_i\}$.
- We can derive **multi-class SVMs** using the same steps...

Multi-Class SVMs

- Can we define a loss that encourages $c=y_i$ to maximize o_{ic} ?

We want $o_{iy_i} > o_{ic}$ for all 'c' that are not correct label y_i
If we penalize violation of this constraint it's degenerate.

We use $o_{iy_i} \geq o_{ic} + 1$ for all $c \neq y_i$ to avoid strict inequality

Equivalently: $0 \geq 1 - o_{iy_i} + o_{ic}$

- For here, there are two ways to **measure constraint violation**:

"Sum"

$$\sum_{c \neq y_i} \max \{ 0, 1 - o_{iy_i} + o_{ic} \}$$

"Max"

$$\max_{c \neq y_i} \left\{ \max \{ 0, 1 - o_{iy_i} + o_{ic} \} \right\}$$

Multi-Class SVMs

- Can we define a loss that encourages $c=y_i$ to maximize o_{ic} ?

"Sum"

$$\sum_{c \neq y_i} \max \{ 0, 1 - \underbrace{w_{y_i}^T x_i}_{o_{iy_i}} + \underbrace{w_c^T x_i}_{o_{ic}} \}$$

"Max"

$$\max_{c \neq y_i} \left\{ \max \{ 0, 1 - \underbrace{w_{y_i}^T x_i}_{o_{iy_i}} + \underbrace{w_c^T x_i}_{o_{ic}} \} \right\}$$

- For each training example 'i':
 - "Sum" rule penalizes for each 'c' that violates the constraint.
 - "Max" rule penalizes for one 'c' that violates the constraint the most.
 - "Sum" gives a penalty of 'k-1' for $W=0$, "max" gives a penalty of '1'.
- If we add L2-regularization, both are called **multi-class SVMs**:
 - "Max" rule is more popular, "sum" rule usually works better.
 - Both are **convex upper bounds on the 0-1 loss**.

Multi-Class Logistic Regression

- We derived **binary logistic loss** by **smoothing a degenerate 'max'**.
 - A **degenerate constraint** in the multi-class case can be written as:

$$o_{iy_i} \geq \max_c \{o_{ic}\}$$

or

$$0 \geq -o_{iy_i} + \max_c \{o_{ic}\}$$

- We want the right side to be as small as possible.
- Let's **smooth the max with the log-sum-exp**:

$$-o_{iy_i} + \log\left(\sum_{c=1}^{\ell} \exp(o_{ic})\right)$$

- This is no longer degenerate: with $W=0$ this gives a loss of $\log(\ell)$.
- Called the **softmax loss** or **cross-entropy**,
and using this loss is called **multi-class logistic regression**.

Multi-Class Logistic Regression

- We **sum the loss over examples** and **add regularization**:

$$f(W) = \sum_{i=1}^N \left[-w_{y_i}^T x_i + \log \left(\sum_{c=1}^{\ell} \exp(w_c^T x_i) \right) \right] + \frac{\lambda}{2} \sum_{c=1}^{\ell} \sum_{j=1}^d w_{cj}^2$$

Tries to make o_{ic} big for the correct label

Approximates $\max_c \{ o_{ic} \}$ so tries to make o_{ic} small for all labels.

Usual L_2 -regularizer on elements of 'W'

- This **objective is convex** (should be clear for 1st and 3rd terms).
 - It is **differentiable** so you can use gradient descent.
- When $\ell=2$, **equivalent to using binary logistic loss**.
 - Not obvious at the moment.

Digression: Frobenius Norm

- The **Frobenius norm** of a (' ℓ ' by ' d ') matrix ' W ' is defined by:

$$\|W\|_F = \sqrt{\sum_{c=1}^{\ell} \sum_{j=1}^d w_{jc}^2}$$

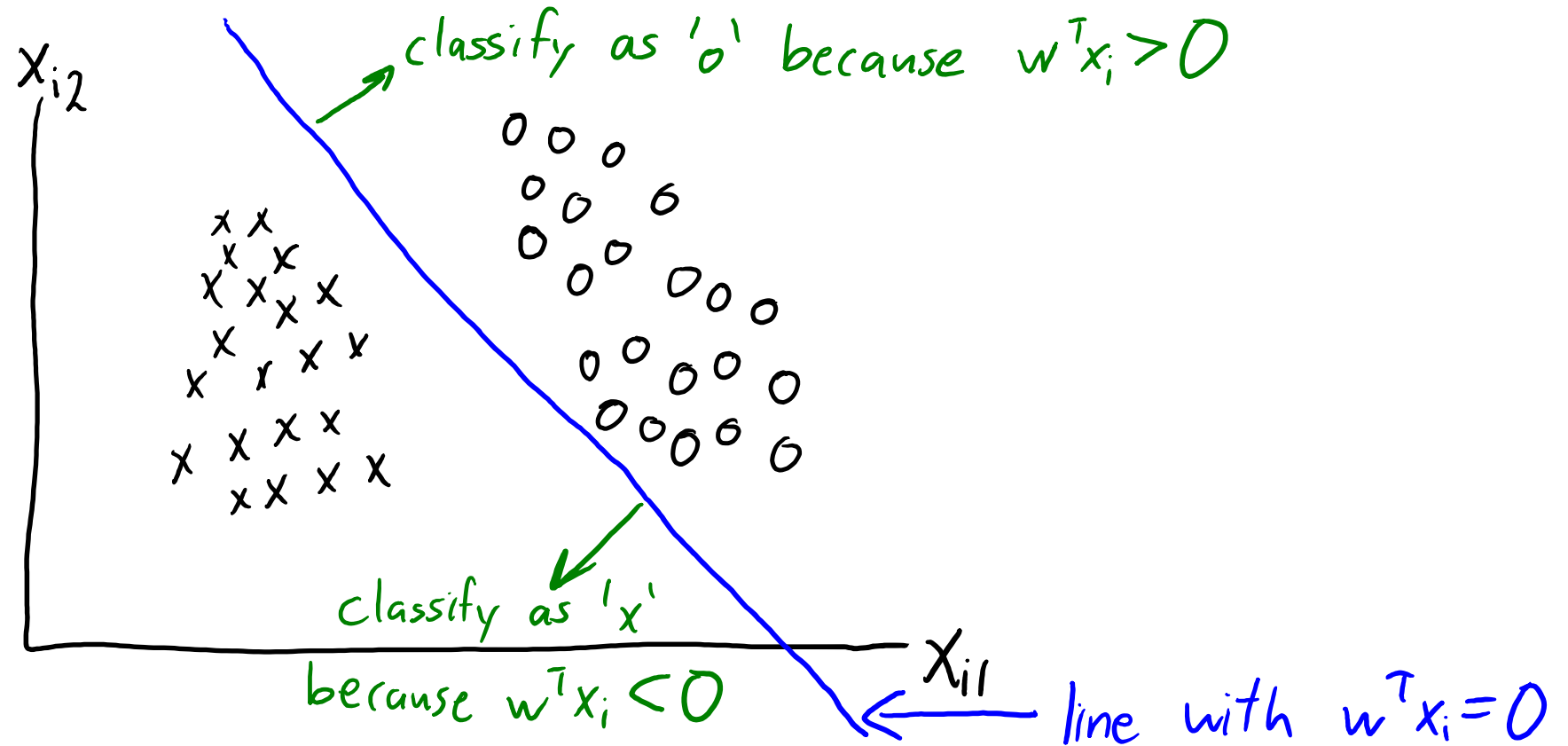
(L_2 -norm if you "stack" elements into one big vector)

- We can use this to write **regularizer in matrix notation**:

$$\begin{aligned} \frac{\lambda}{2} \sum_{c=1}^{\ell} \sum_{j=1}^d w_{cj}^2 &= \frac{\lambda}{2} \sum_{c=1}^{\ell} \|w_c\|^2 && \text{("}L_2\text{-regularizer on each vector")} \\ &= \frac{\lambda}{2} \|W\|_F^2 && \text{("Frobenius-regularizer on matrix")} \end{aligned}$$

Shape of Decision Boundaries

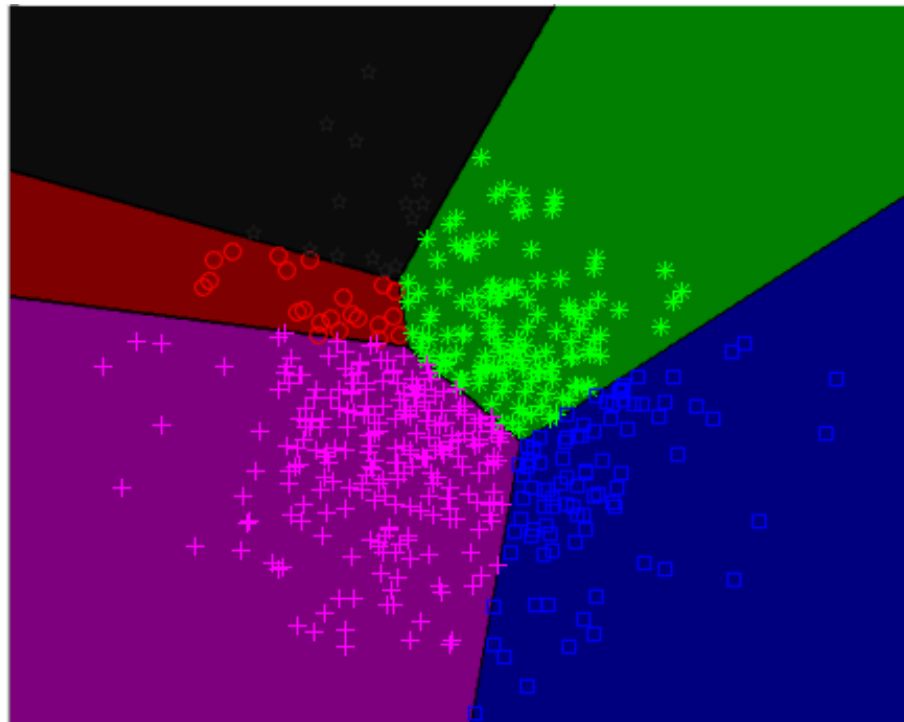
- Recall that a **binary linear classifier** splits space using a hyper-plane:



- Divides x_i space into 2 "half-spaces".

Shape of Decision Boundaries

- **Multi-class linear classifier** is intersection of these “half-spaces”:
 - This divides the space into **convex regions** (like k-means):



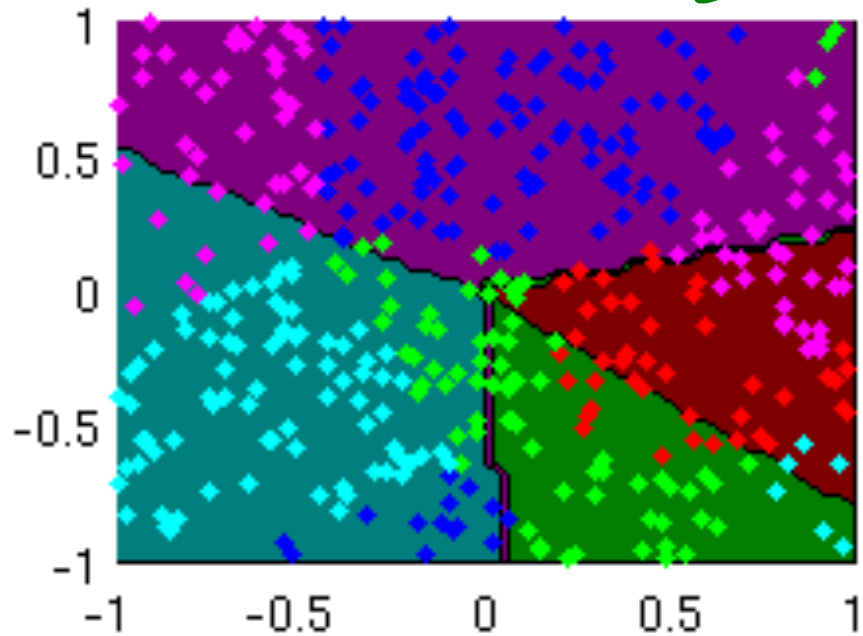
"Blue" region is region where we have:

$$w_{\text{blue}}^T x_i \geq w_{\text{green}}^T x_i$$
$$w_{\text{blue}}^T x_i \geq w_{\text{magenta}}^T x_i$$
$$w_{\text{blue}}^T x_i \geq w_{\text{red}}^T x_i$$
$$w_{\text{blue}}^T x_i \geq w_{\text{black}}^T x_i$$

Shape of Decision Boundaries

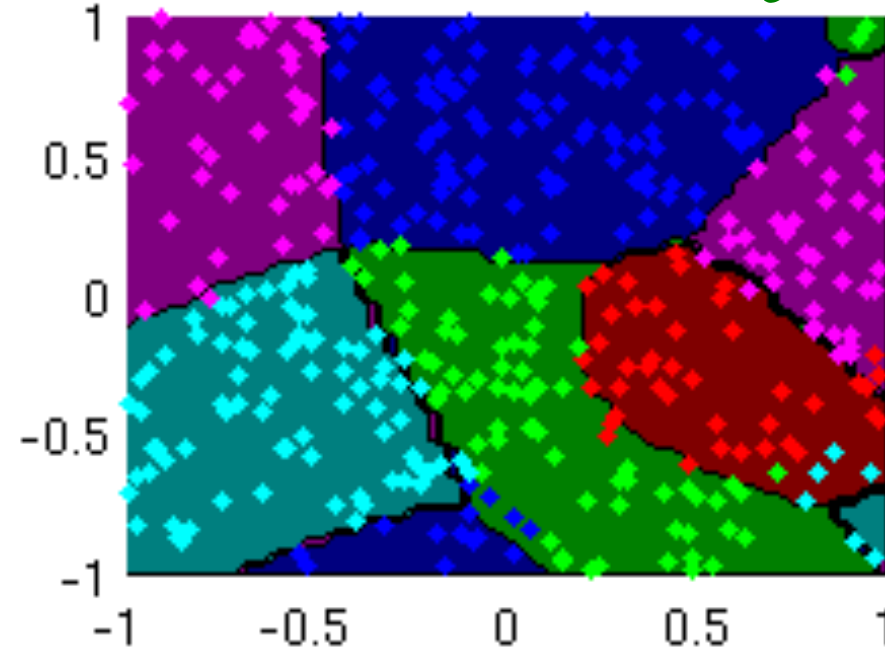
- **Multi-class linear classifier** is intersection of these “half-spaces”:
 - Though regions could be **non-convex with non-linear feature transforms**:

Original Features



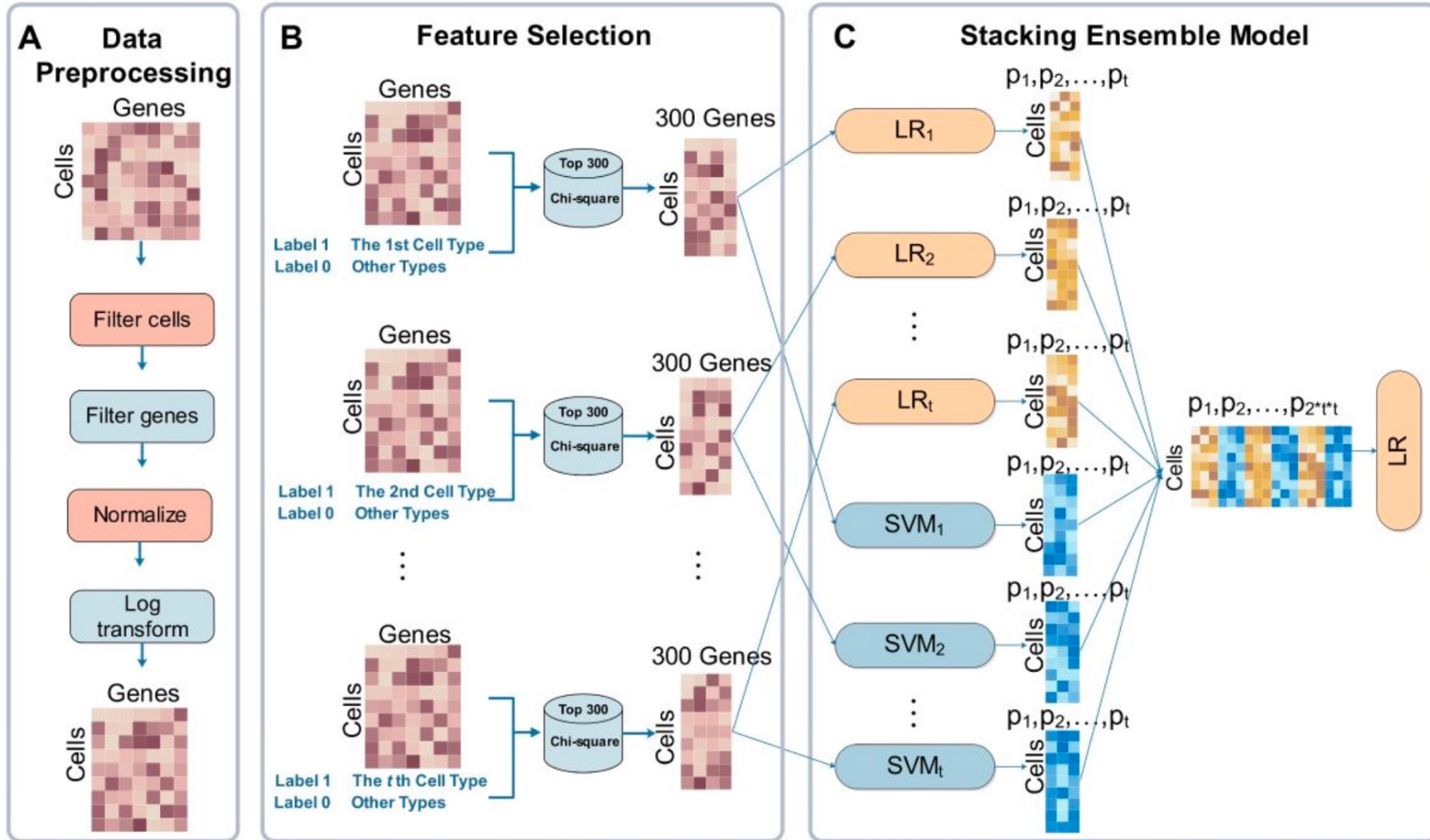
Convex regions (not a good classifier)

Gaussian RBFs as Features



Non-convex regions (much better classifier on this data)

Example Applications



Summary

- **Logistic loss** uses a smooth convex approximation to the 0-1 loss.
- **SVMs and logistic regression are very widely-used.**
 - A lot of ML consulting: “find good features, use L2-regularized logistic/SVM”.
 - Under certain conditions, can be viewed as “**maximizing the margin**”.
 - Both are just **linear** classifiers (a hyperplane dividing into two halfspaces).
- **‘One vs all’** turns a binary classifier into a multi-class classifier.
- **Multi-class SVMs** measure violation of classification constraints.
- **Softmax loss** is a multi-class version of logistic loss.

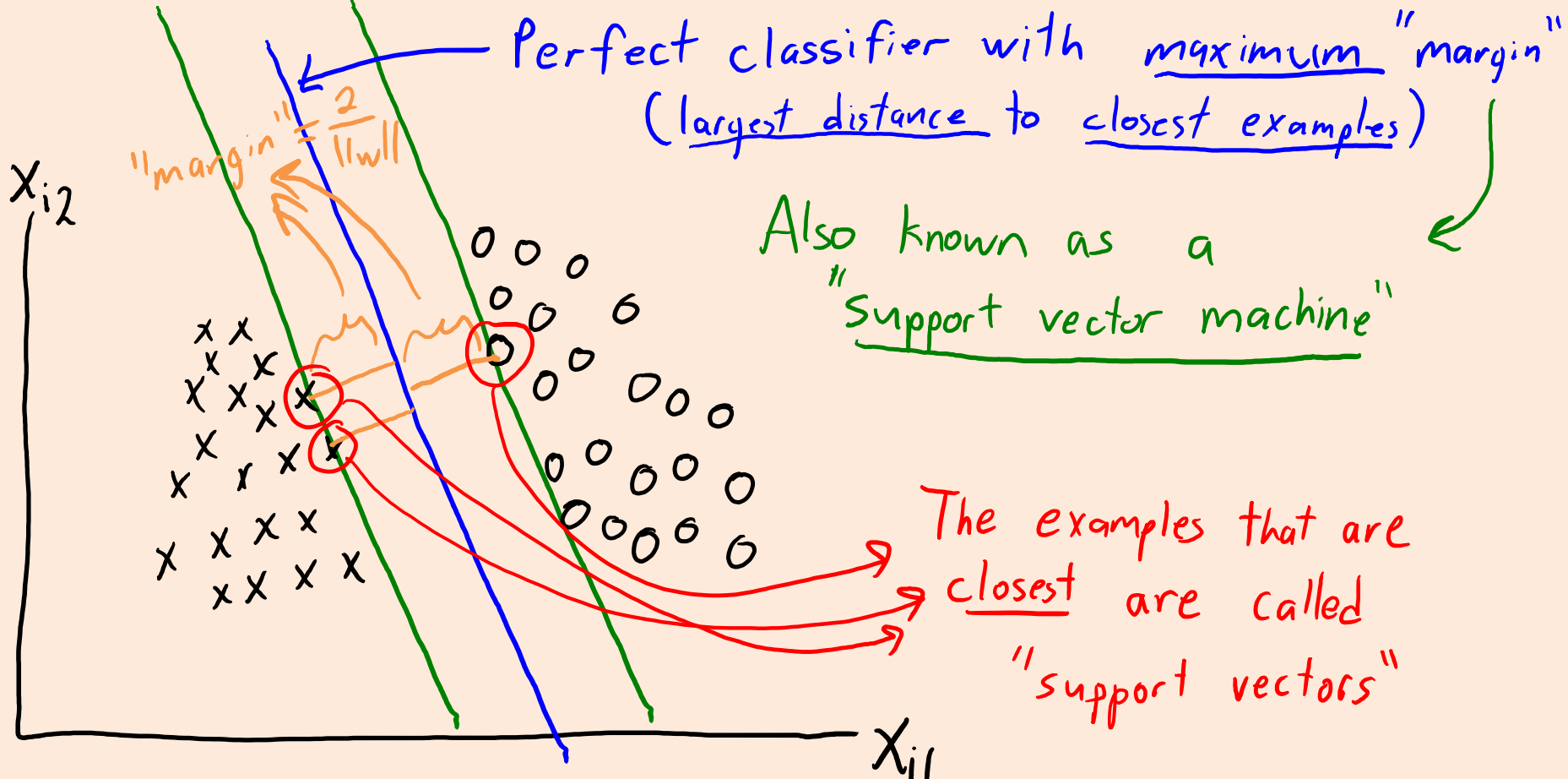
- Next time: what makes good features?

Hinge-Loss Perceptron

- A perceptron-like algorithm for minimizing the hinge loss:
 - Start with any w^0 .
 - Go through examples until you find an example with $y_i w^T x_i > 1$.
 - Set $w^{t+1} = w^t + \frac{1 - y_i (w^t)^T x_i}{x_i^T x_i} y_i x_i$ (minimum change to w_t that satisfies constraint).
- If a classifier with hinge loss of 0 exists, this converges to one.
 - Looks like perceptron, but with a step size added to update (green term).
 - Get perceptron algorithm if you replace green term with '1'.
 - A special case of the “projection onto convex sets” (POCS) algorithm.

Maximum-Margin Classifier

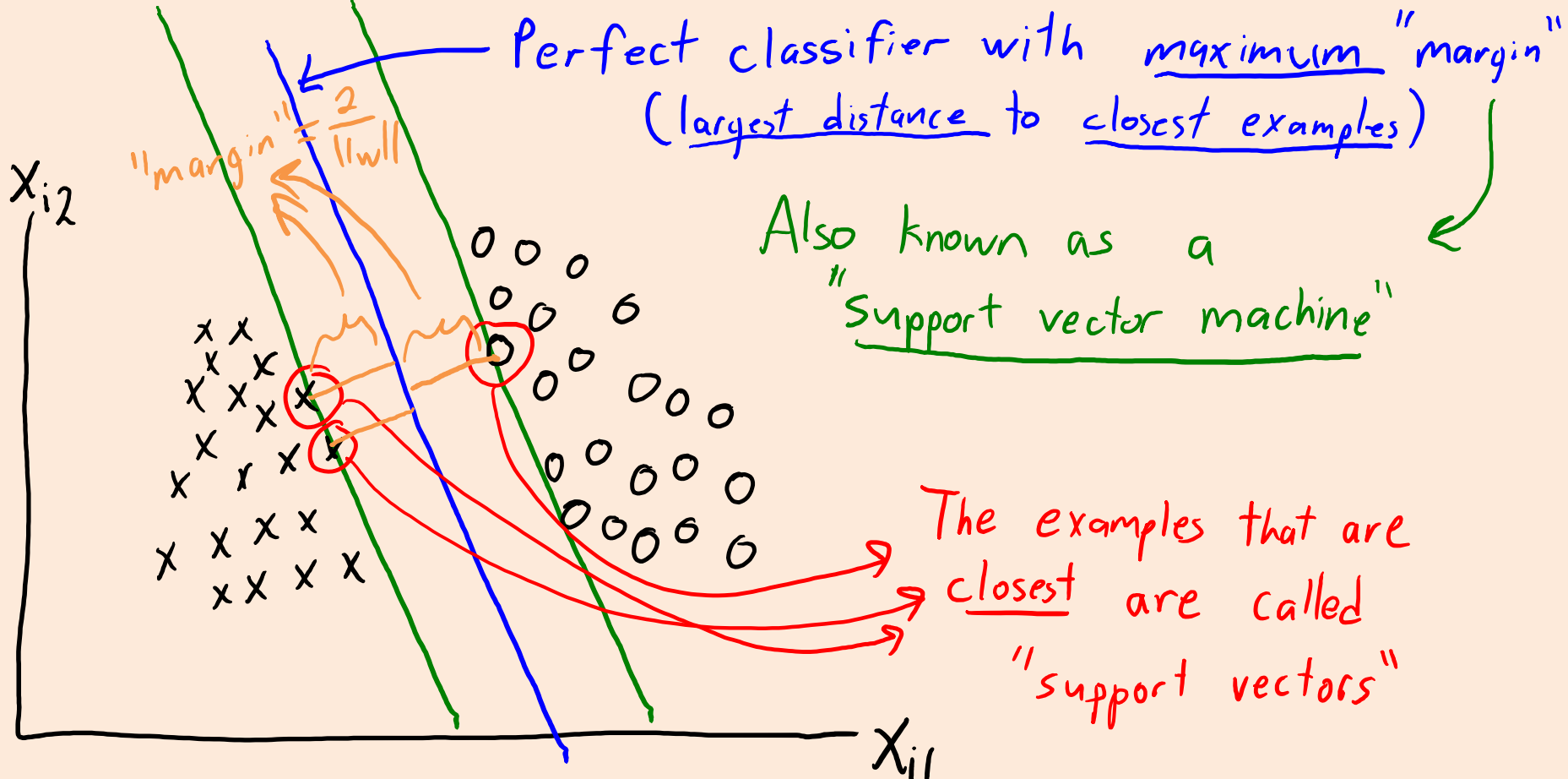
- Consider a linearly-separable dataset.
 - Maximum-margin classifier: choose the farthest from both classes.



Maximum-Margin Classifier

- Consider a linearly-separable dataset.
 - Maximum-margin classifier: choose the farthest from both classes.

Final classifier only depends on support vectors

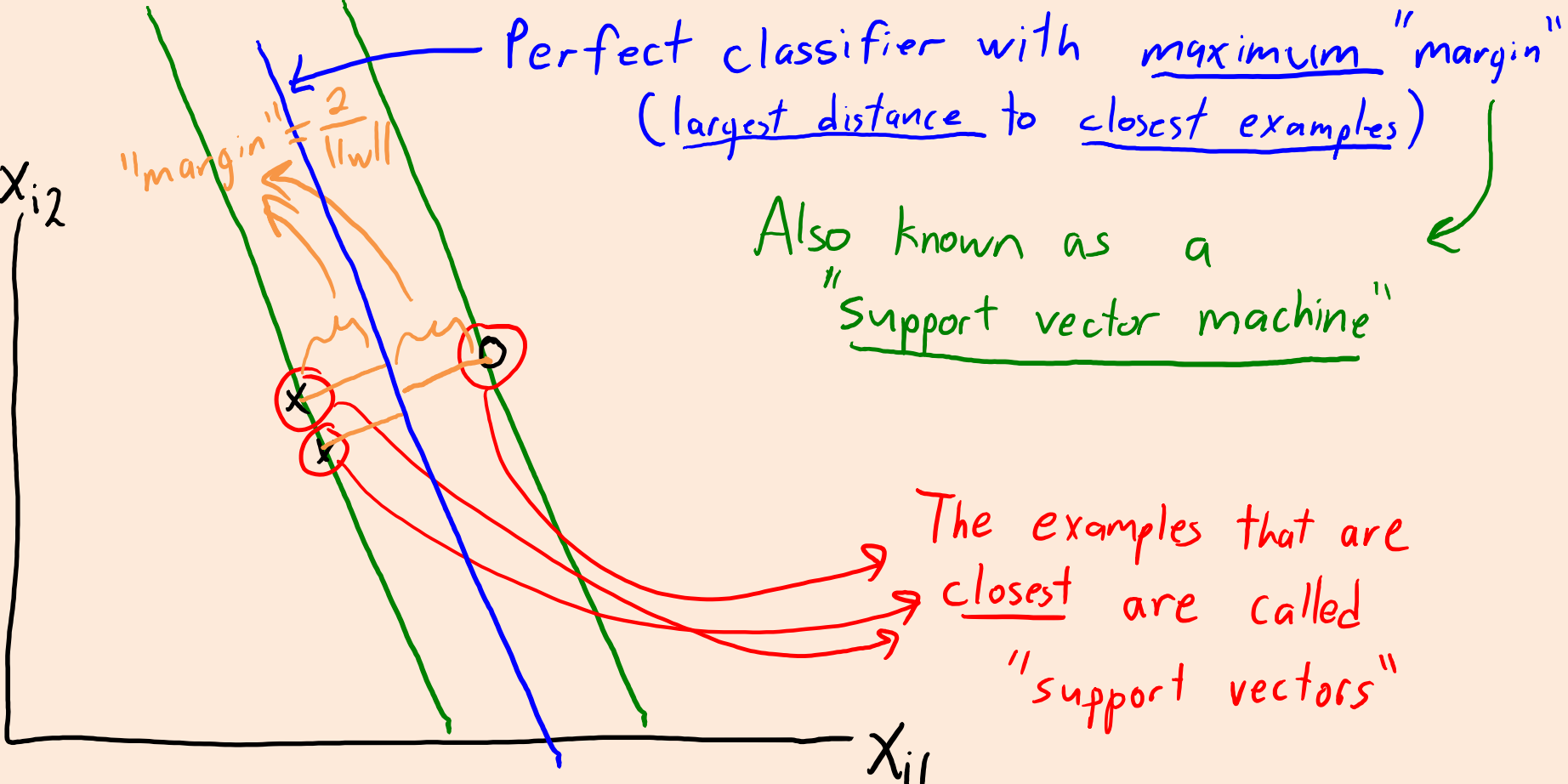


Maximum-Margin Classifier

- Consider a linearly-separable dataset.
 - Maximum-margin classifier: choose the farthest from both classes.

Final classifier only depends on support vectors

You could throw away the other examples and get the same classifier.



Perfect classifier with maximum "margin"
(largest distance to closest examples)

Also known as a "support vector machine"

The examples that are closest are called "support vectors"

"margin" = $\frac{2}{\|w\|}$

x_{i2}

x_{i1}

Support Vector Machines

- For **linearly-separable** data, **SVM** minimizes:

$$f(w) = \frac{1}{2} \|w\|^2 \quad (\text{equivalent to maximizing margin } \frac{2}{\|w\|})$$

- Subject to the constraints that:
(see Wikipedia/textbooks)
- $$\begin{aligned} w^T x_i &\geq 1 && \text{for } y_i = 1 \\ w^T x_i &\leq -1 && \text{for } y_i = -1 \end{aligned} \quad (\text{classify all examples correctly})$$

- But **most data is not linearly separable**.
- For **non-separable data**, try to **minimize violation of constraints**:

If $w^T x_i \leq -1$ and $y_i = -1$ then "violation" should be zero.

If $w^T x_i \geq -1$ and $y_i = -1$ then we "violate constraint" by $1 + w^T x_i$

→ Constraint violation is the hinge loss.

Support Vector Machines

- Try to **maximizing margin** and also **minimizing constraint violation**:

Hinge loss
for example 'i':

$$f(w) = \sum_{i=1}^n \max\{0, 1 - y_i w^T x_i\} + \frac{1}{2} \|w\|^2$$

if's the amount we violate $y_i w^T x_i \geq 1$
"slack"

Original SVM objective:
encourages large margin.

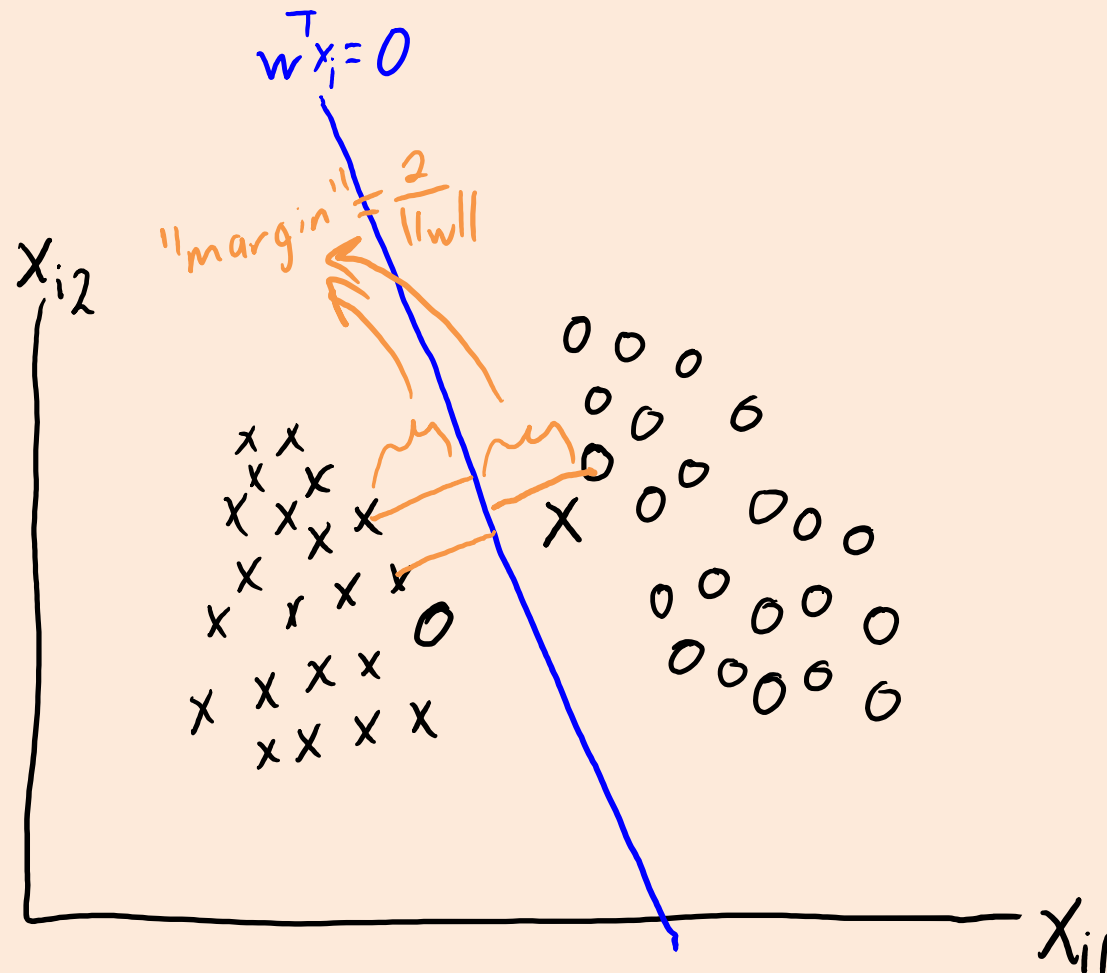
- We typically control margin/violation trade-off with parameter " λ ":

$$f(w) = \sum_{i=1}^n \max\{0, 1 - y_i w^T x_i\} + \frac{\lambda}{2} \|w\|^2$$

- This is the standard SVM formulation (L2-regularized hinge).
 - Some formulations use $\lambda = 1$ and multiply hinge by 'C' (equivalent).

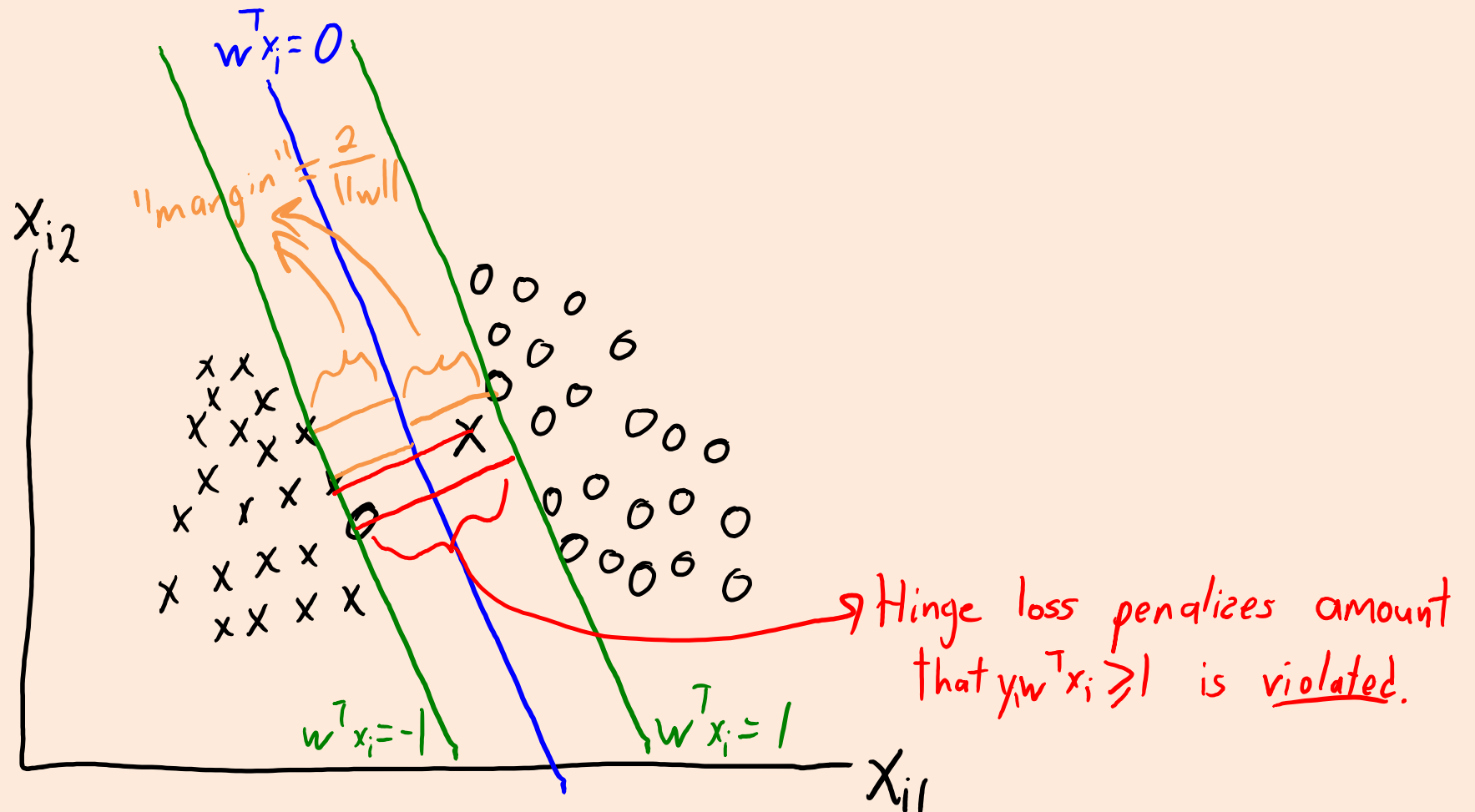
Support Vector Machines for Non-Separable

- Non-separable case:



Support Vector Machines for Non-Separable

- Non-separable case:



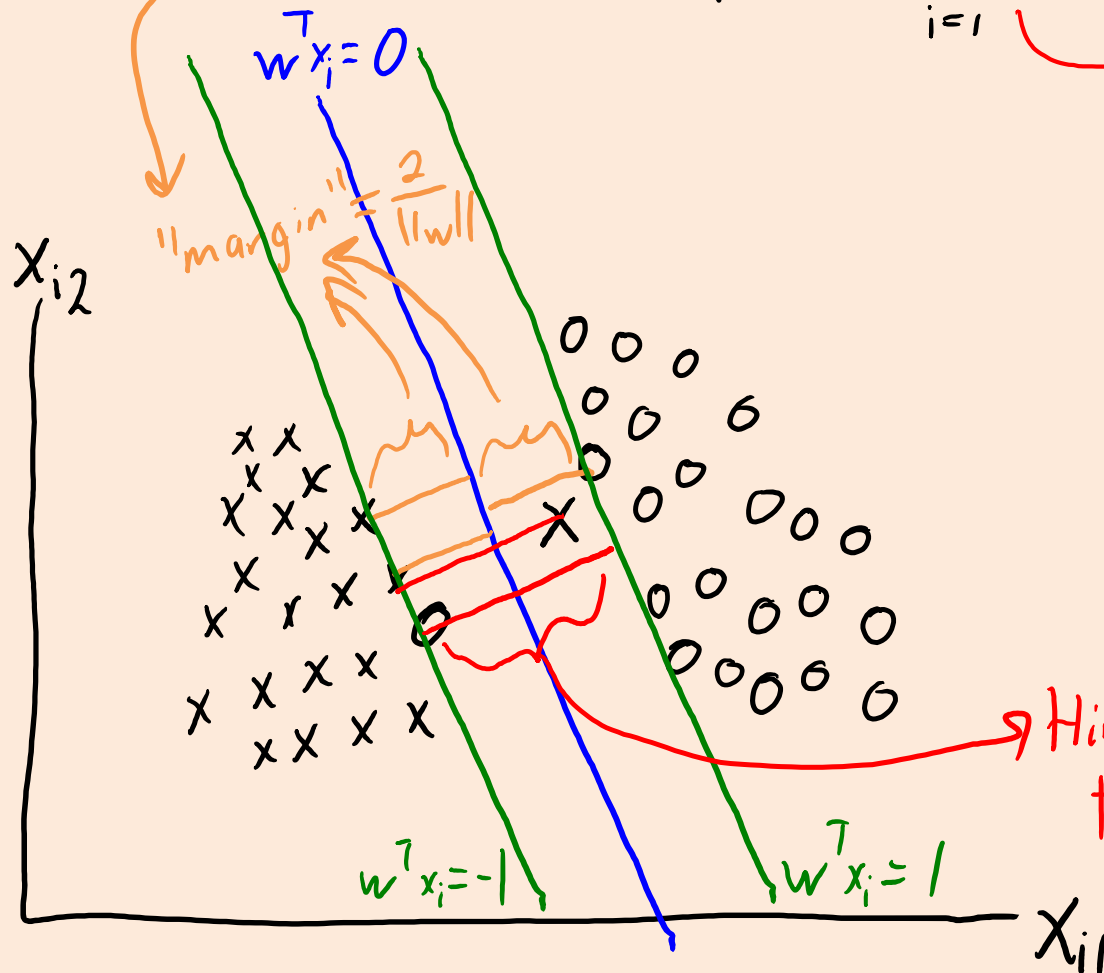
Support Vector Machines for Non-Separable

- Non-separable case:

$$f(w) = \sum_{i=1}^n \max\{0, 1 - y_i w^T x_i\} + \frac{\lambda}{2} \|w\|^2$$

λ controls trade-off between having large margin and classifying examples correctly.

Hinge loss penalizes amount that $y_i w^T x_i \geq 1$ is violated.



Logistic regression can be viewed as smooth approximation to SVMs.

But, no concept of "support vectors" with logistic loss.

Support Vector Machines for Non-Separable

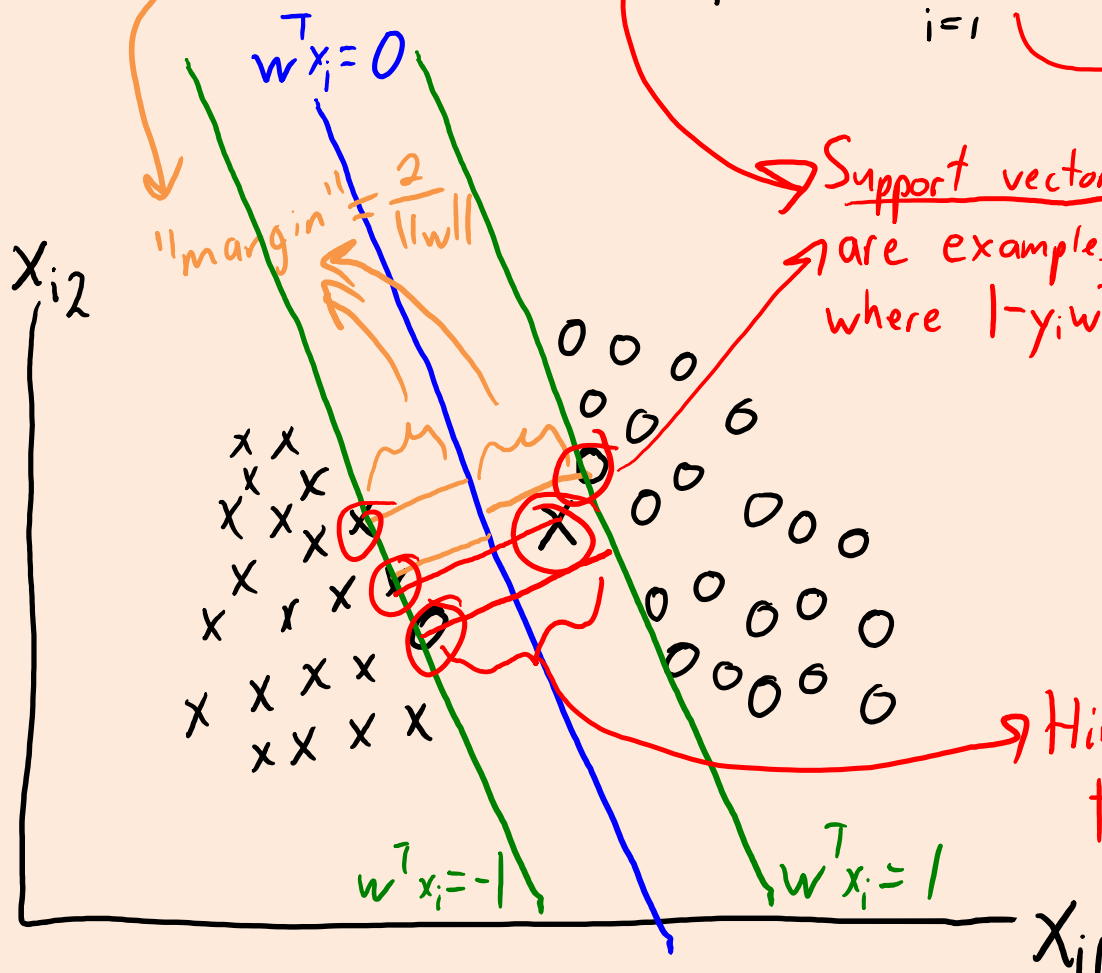
- Non-separable case:

$$f(w) = \sum_{i=1}^n \max\{0, 1 - y_i w^T x_i\} + \frac{\lambda}{2} \|w\|^2$$

Support vectors
are examples 'i'
where $1 - y_i w^T x_i \geq 0$

λ controls trade-off
between having
large margin and
classifying examples
correctly.

Hinge loss penalizes amount
that $y_i w^T x_i \geq 1$ is violated.



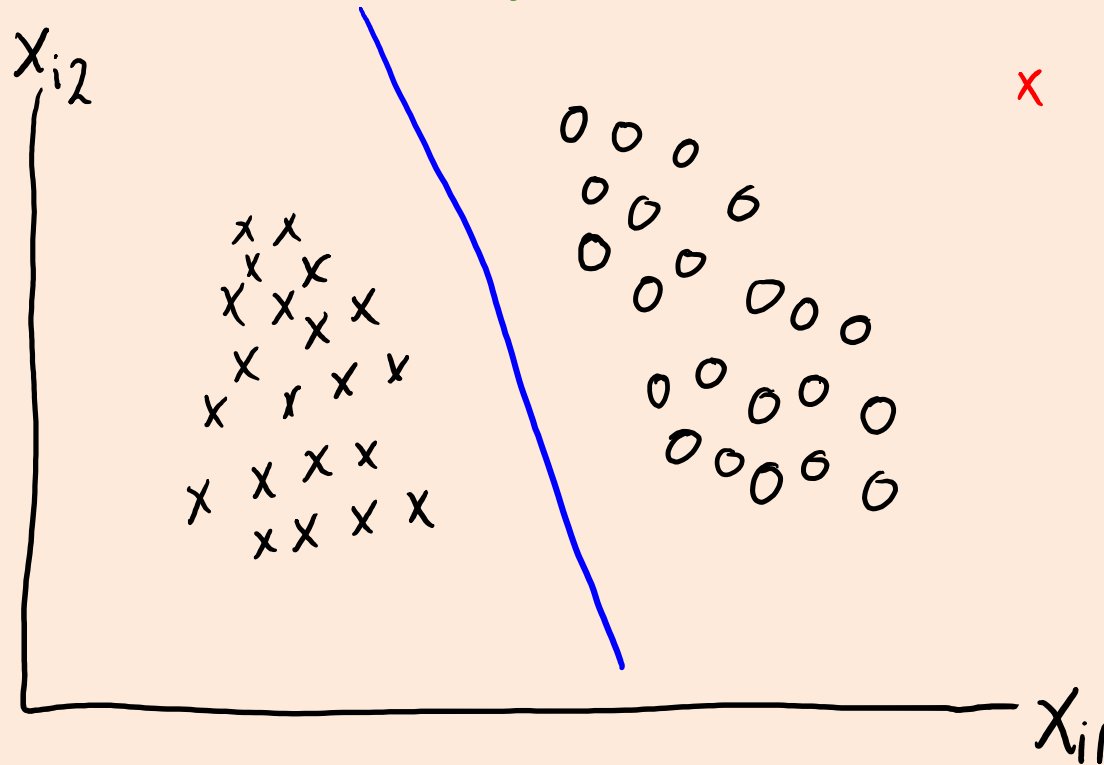
Logistic regression can
be viewed as smooth
approximation to SVMs.
But, no concept of
"support vectors" with
logistic loss.

Discussion of Various Linear Classifiers

- Perceptron vs. logistic vs. SVM:
 - These linear classifiers are all extremely similar. They are basically just variations on reasonable methods to learn a classifier that uses the rule $\hat{y}_i = \text{sign}(w^T x_i)$. (The online vs. offline issue is a red herring, you can train logistic/SVMs online using stochastic gradient and you can write a linear program that will give you a minimizer of the perceptron objective).
 - If you want to explore the small differences, these are some of the usual arguments:
 - The perceptron has largely been replaced by logistic/SVM, except in certain subfields like theory (it is easy to prove things about perceptrons) and natural language processing (mostly historical reasons). Perceptrons have the potential disadvantages of non-regularized models (non-uniqueness and potential non-existence of the solution, potential high sensitivity to small changes in the data, and non-robustness to irrelevant features). However, perceptrons do not interact well with regularization: if you add L2-regularization and the dataset is linearly-separable, then the solution only exists as a limit and it is actually $w=0$ (although it may still work in practice).
 - A usual criticism of logistic regression by people that favour SVMs is that, if the data is linearly separable, then the solution only exists as a limit as some elements w go to plus or minus ∞ . However, this argument disappears if you add regularization. A second argument traditionally made by SVM people is that you can't kernelize logistic regression, but this is now known to be incorrect (we'll cover a general kernelization strategy for L2-regularized linear classifiers in one of the next two classes).
 - The remaining differences between logistic and SVMs is that logistic regression is smooth while SVMs have support vectors. This means that the logistic regression training problem is easier from an optimization perspective (we'll get to this next class). But if you have very few support vectors, you can only take advantage of this with SVMs (or perceptrons), and this is especially important if you are using kernels.
- Regarding other linear predictors for binary classification, there are a few more:
 - Probit regression uses the Gaussian CDF in place of the logistic sigmoid function. This has very similar properties to logistic regression, but it's harder to generalize to the multi-class case (while probit regression is better if you are using a "Bayesian" estimator). You could actually use any CDF as your sigmoid function, and if there is some asymmetry between the classes using an extreme value distribution is sometimes advocated in statistics.
 - In neural networks, they sometimes use tanh in place of the logistic sigmoid function, and the reason to do this is to get values into the interval $[-1,1]$ instead of $[0,1]$.
 - If you want to keep support vectors but get a smooth optimization problem, you can square the hinge loss (making it once but not twice differentiable), and this is called smooth SVMs. Alternately, you could replace the non-differentiable kink with a small smooth part, and this is called Huberized SVMs.
 - Finally, some people actually just apply least squares to classification problems. If you use a flexible enough basis/kernel, then the 'bad' errors may not actually be that harmful.

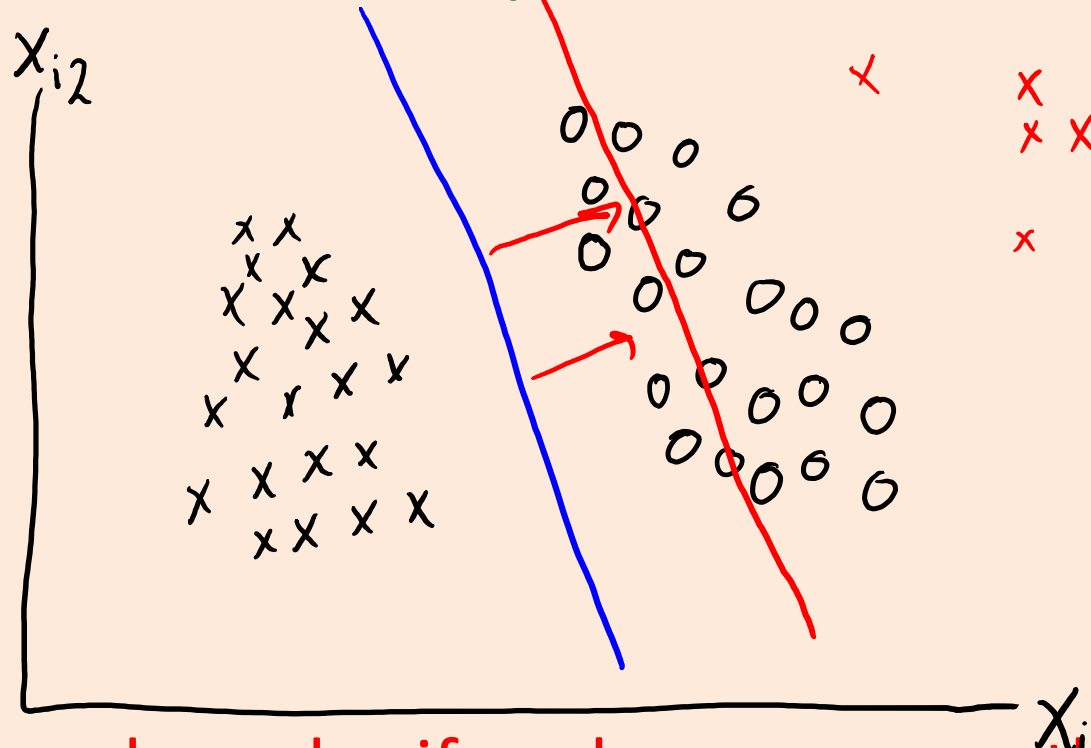
Robustness and Convex Approximations

- Because the hinge/logistic grow like absolute value for mistakes, they tend **not to be affected by a small number of outliers**.



Robustness and Convex Approximations

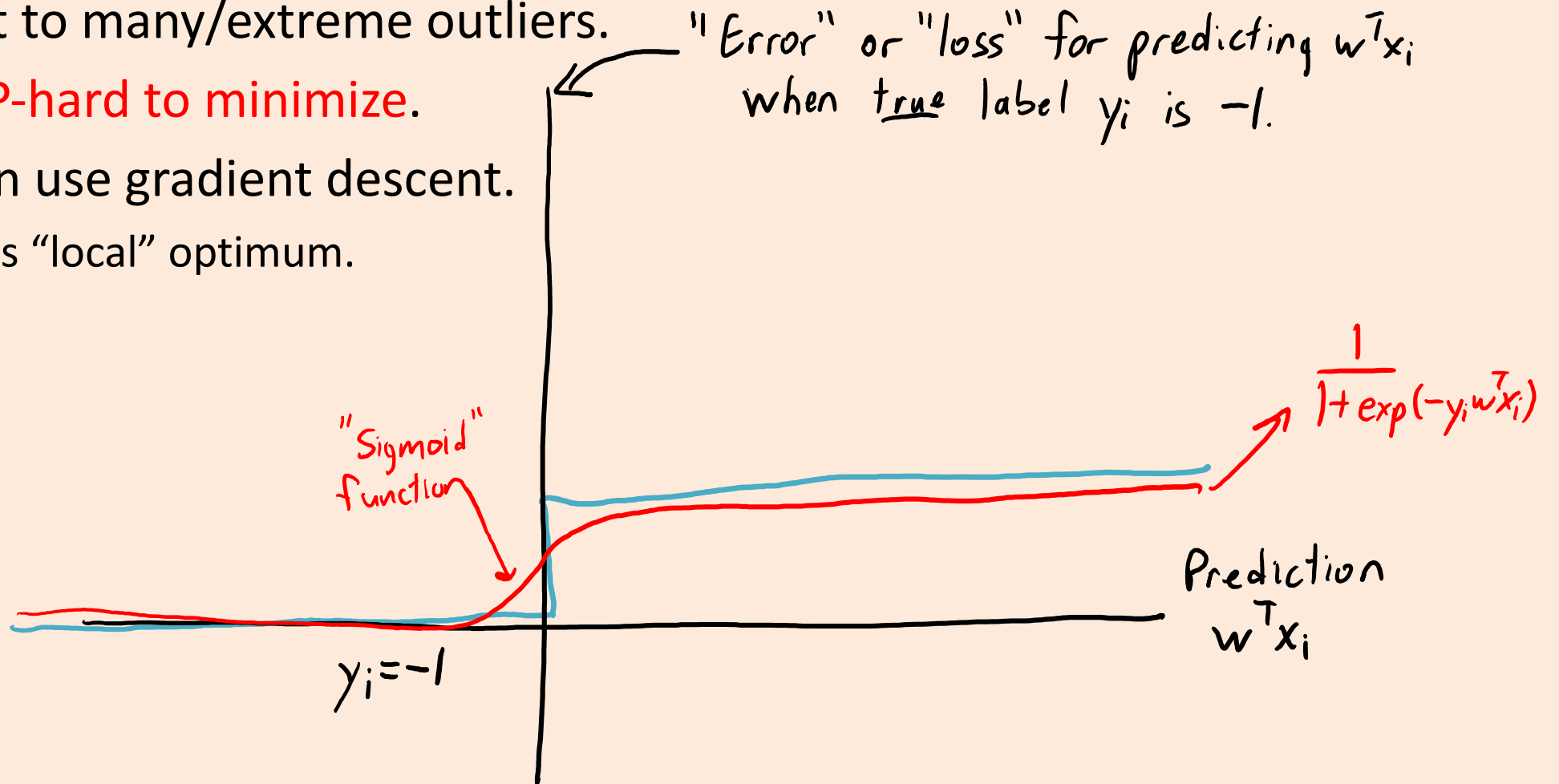
- Because the hinge/logistic grow like absolute value for mistakes, they tend **not to be affected by a small number of outliers.**



- But **performance degrades if we have many outliers.**

Non-Convex 0-1 Approximations

- There exists some **smooth non-convex 0-1 approximations**.
 - Robust to many/extreme outliers.
 - Still **NP-hard to minimize**.
 - But can use gradient descent.
 - Finds “local” optimum.



“Robust” Logistic Regression

- A recent idea: add a “fudge factor” v_i for each example.

$$f(w, v) = \sum_{i=1}^n \log(1 + \exp(-y_i w^T x_i + v_i))$$

- If $w^T x_i$ gets the sign wrong, we can “correct” the mis-classification by modifying v_i .
 - This makes the training error lower but doesn’t directly help with test data, because we won’t have the v_i for test data.
 - But having the v_i means the ‘ w ’ parameters don’t need to focus as much on outliers (they can make $|v_i|$ big if $\text{sign}(w^T x_i)$ is very wrong).

“Robust” Logistic Regression

- A recent idea: add a “fudge factor” v_i for each example.

$$f(w, v) = \sum_{i=1}^n \log(1 + \exp(-y_i w^T x_i + v_i))$$

- If $w^T x_i$ gets the sign wrong, we can “correct” the mis-classification by modifying v_i .
- A problem is that we can ignore the ‘w’ and get a tiny training error by just updating the v_i variables.
- But we want most v_i to be zero, so “robust logistic regression” puts an L1-regularizer on the v_i values:

$$f(w, v) = \sum_{i=1}^n \log(1 + \exp(-y_i w^T x_i + v_i)) + \lambda \|v\|_1$$

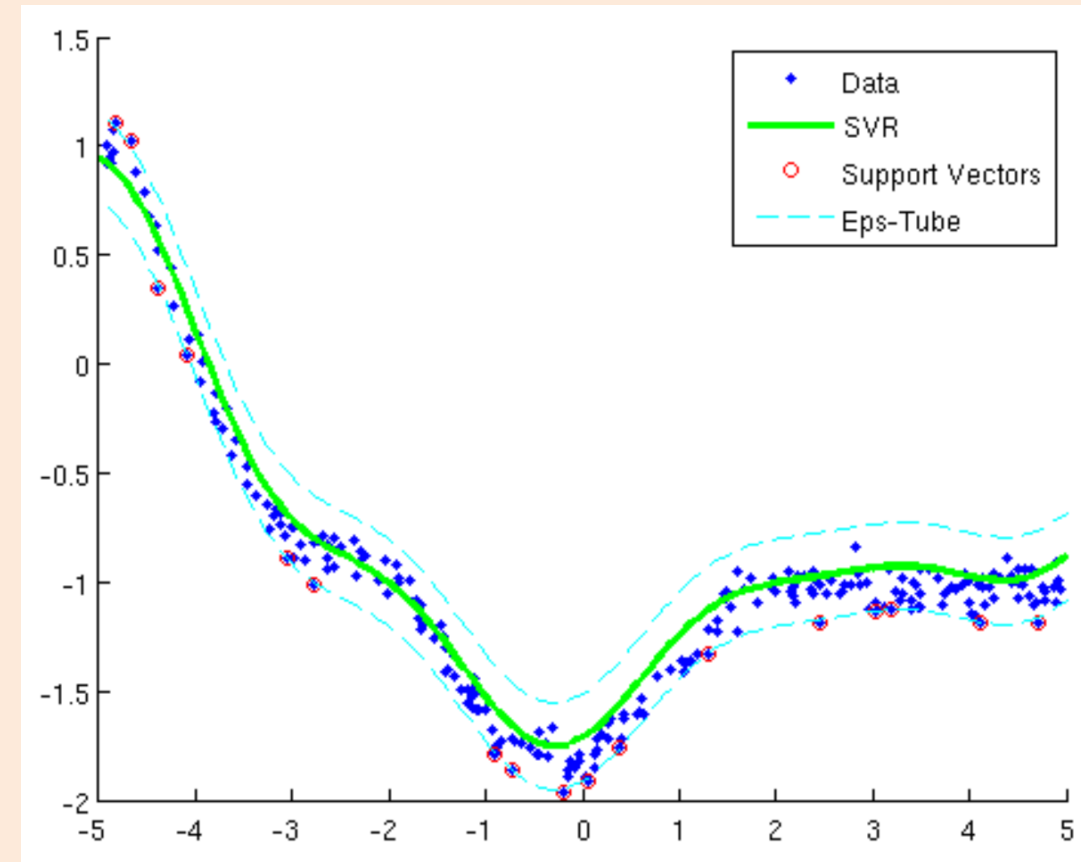
- You would probably also want to regularize the ‘w’ with different λ .

Support Vector Regression

- Support vector regression objective (with hyper-parameter ϵ):

$$f(w) = \sum_{i=1}^n \max\{0, |w^T x_i - y_i| - \epsilon\} + \frac{\lambda}{2} \|w\|^2$$

- Looks like L2-regularized robust regression with the L1-loss.
- But have **loss of 0** if \hat{y}_i within ϵ of \tilde{y}_i .
 - So doesn't try to fit data exactly.
 - This can help fight overfitting.
- Support vectors are points with loss > 0 .
 - Points outside the “epsilon-tube”.
- Example with Gaussian-RBFs as features:



1-Class SVMs

- 1-class SVMs for outlier detection.

$$f(w, w_0) = \sum_{i=1}^N [\max\{0, w_0 - w^T x_i\} - w_0] + \frac{\lambda}{2} \|w\|_2^2$$

– Variables are ‘w’ (vector) and ‘w₀’ (scalar).

– Only trains on “inliers”.

- Tries to make $w^T x_i$ bigger than w_0 for inliers.
- At test time: says “outlier” if $w^T x_i < w_0$.
- Usually used with RBFs.

– The above is one possible 1-class formulation, but there are many more.

