# CPSC 340:
# Machine Learning and Data Mining

Support Vector Machines

# Admin

- Sunday Feb 19: Assignment 3 due
- Thursday Feb 23: Assignment 3 solutions posted
- Wednesday March 1: midterm
  - Covers assignments 1-3 and lectures 1-16
  - Closed book, 1 double-sided sheet of notes allowed
  - Starts at 1pm sharp, ends at 1:55pm
- This Friday (Feb 17) we'll save some time for review / Q&A
  - This will be about recent difficult topics, NOT the midterm
    - Next tutorials will focus on the midterm
  - This is in response to 61% of you responding "I am so lost that I don't even know what questions to ask"
  - If you're comfortable with the material you can skip / leave early

# "Part 3" Review

- Focus of Part 3 is linear models:
  - Supervised learning where prediction is linear combination of features:

$$y_i = w_1 x_{i1} + w_2 x_{i2} + \cdots + w_d x_{id}$$
$$= w^T x_i$$

- Change of basis: replace features $x_i$ with $z_i$:
  - Add a bias variable (feature that is always one).
  - Polynomial basis.
  - Radial basis functions (non-parametric basis).
- Regression:
  - Target $y_i$ is numerical.
  - Testing whether (yhat == $y_i$) doesn't make sense.

# Part 3 Review

- Alternate error functions for regression:
  - Squared error: $\frac{1}{2}\sum_{i=1}^{n}(w^T x_i - y_i)^2$ or $\frac{1}{2}\|Xw - y\|^2$
    - Can find optimal 'w' by solving linear system.
  - $L_1$-norm and $L_\infty$-norm errors:

    $$\|Xw - y\|_1 \qquad \|Xw - y\|_\infty$$

    - More/less robust to outliers.
- L2-regularization:
  - Adding a penalty on the L2-norm of 'w' to decrease overfitting:

$$f(w) = \|Xw - y\|_1 + \frac{\lambda}{2}\|w\|^2$$

# Part 3 Review

- Gradient descent:
  - Can we used to find a local minimum of a smooth function.
- $L_1$-norm and $L_\infty$-norm errors are convex but non-smooth:
  - But we can smooth them using Huber and log-sum-exp functions.
- Convex functions:
  - Special functions where all local minima are global minima.
  - Simple rules for showing that a function is convex.

# Last Time: Classification using Regression

- Binary classification using sign of linear models:

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

  $y_i \in \{+1, -1\}$

- Problems with existing loss functions:
  - If $y_i = +1$ and $w^T x_i = +100$, then squared error $(w^T x_i - y_i)^2$ is huge.
  - Hard to minimize training error ("0-1 loss") with respect to 'w'.
- Motivates convex approximations to 0-1 loss:
  - Logistic loss (logistic regression): $\sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i)) + \frac{\lambda}{2}\|w\|^2$
  - Hinge loss (support vector machine): $\sum_{i=1}^{n} \max\{0, 1 - y_i w^T x_i\} + \frac{\lambda}{2}\|w\|^2$

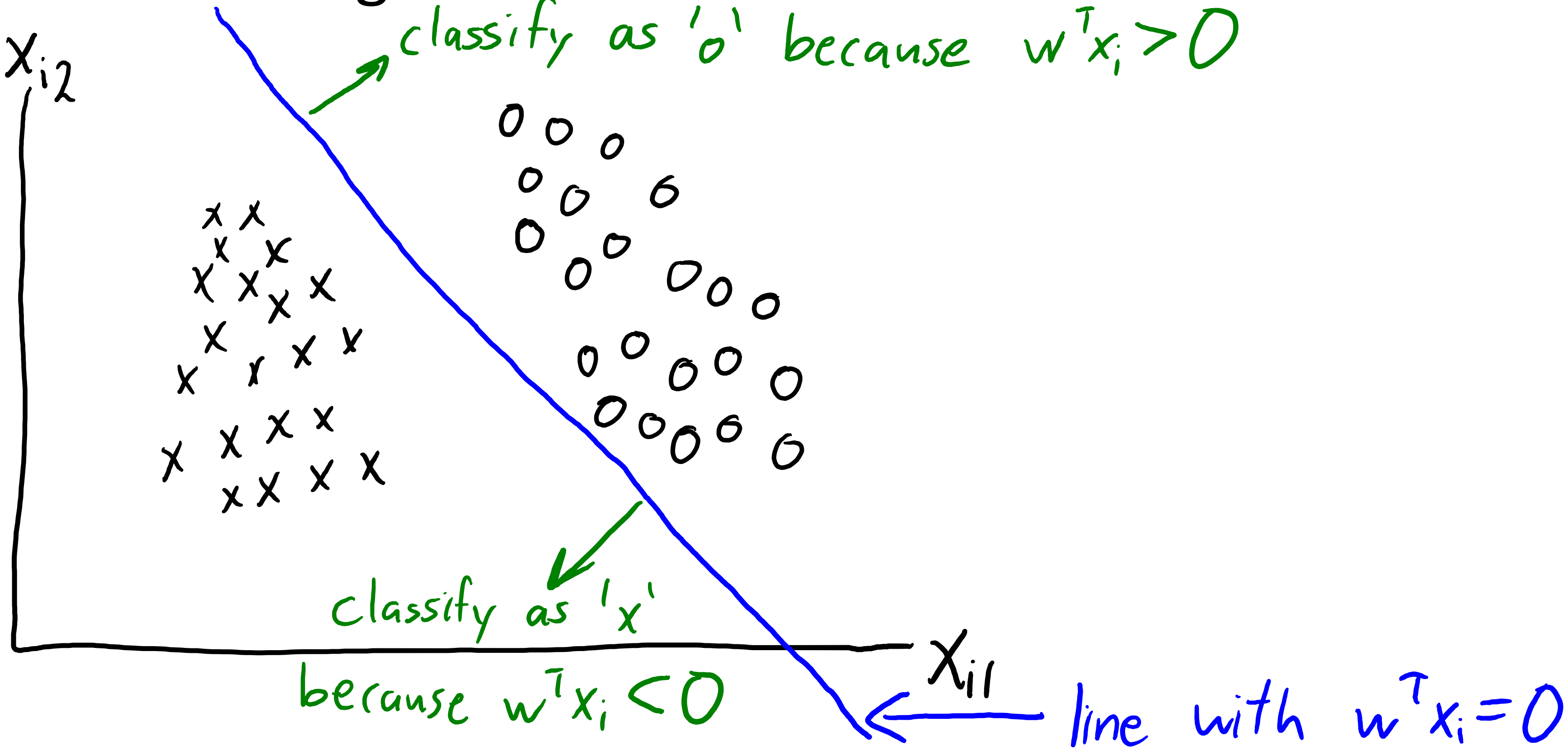
# Last Time: Classification using Regression

- Can minimize smooth/convex logistic loss using gradient descent.
  - There are also efficient methods for support vector machines (SVMs).
- Logistic regression and SVMs are used EVERYWHERE!
  - Fast training and testing, weights $w_j$ are easy to understand.
  - With high-dimensional features and regularization, often good test error.
    - Otherwise, often good test error with RBF basis and regularization.

- Some random questions you might be asking:
  - Can we use a polynomial basis with more than 1 feature?
  - Why didn't we do the "textbook" derivation of logistic/SVM?
  - How do we train on all of Gmail?
  - Did we miss feature selection?

# 2D View of Linear Classifiers

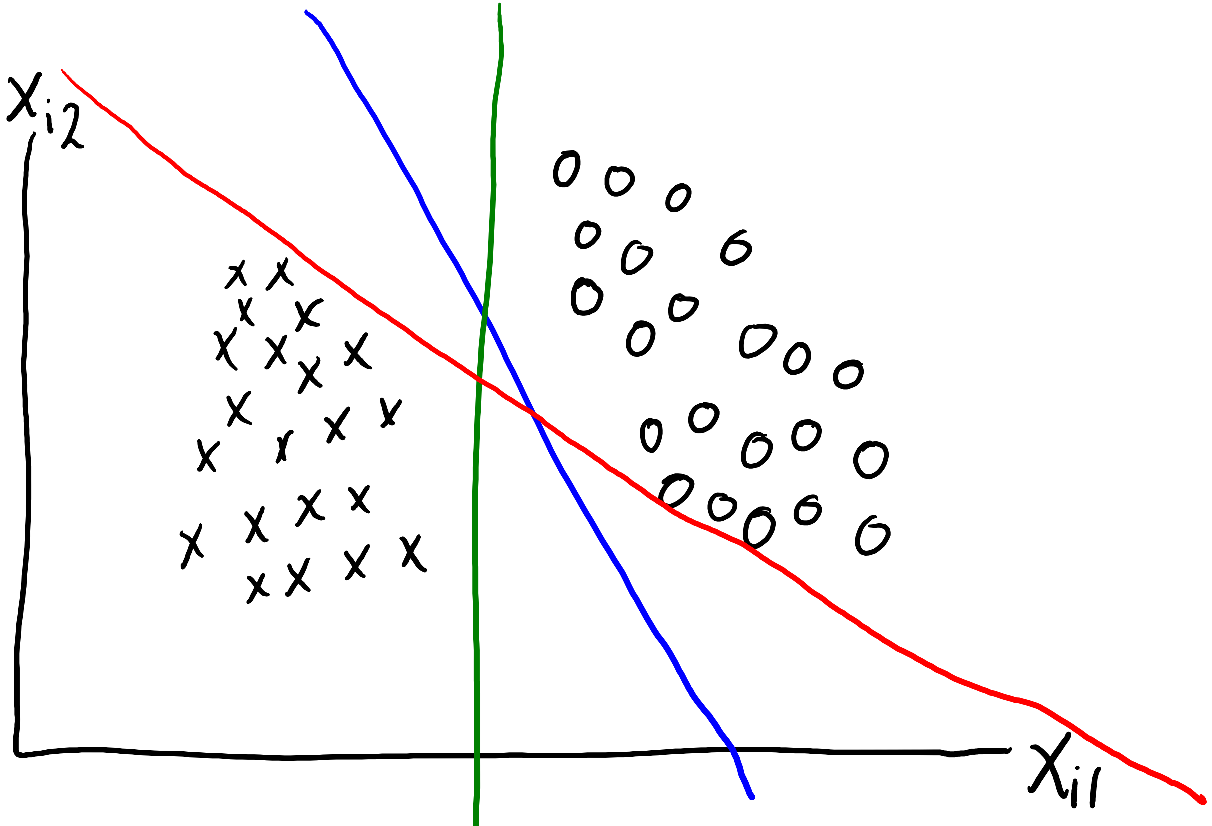
- 2D Visualization of linear regression for classification:




- “Linearly separable”: a perfect linear classifier exists.

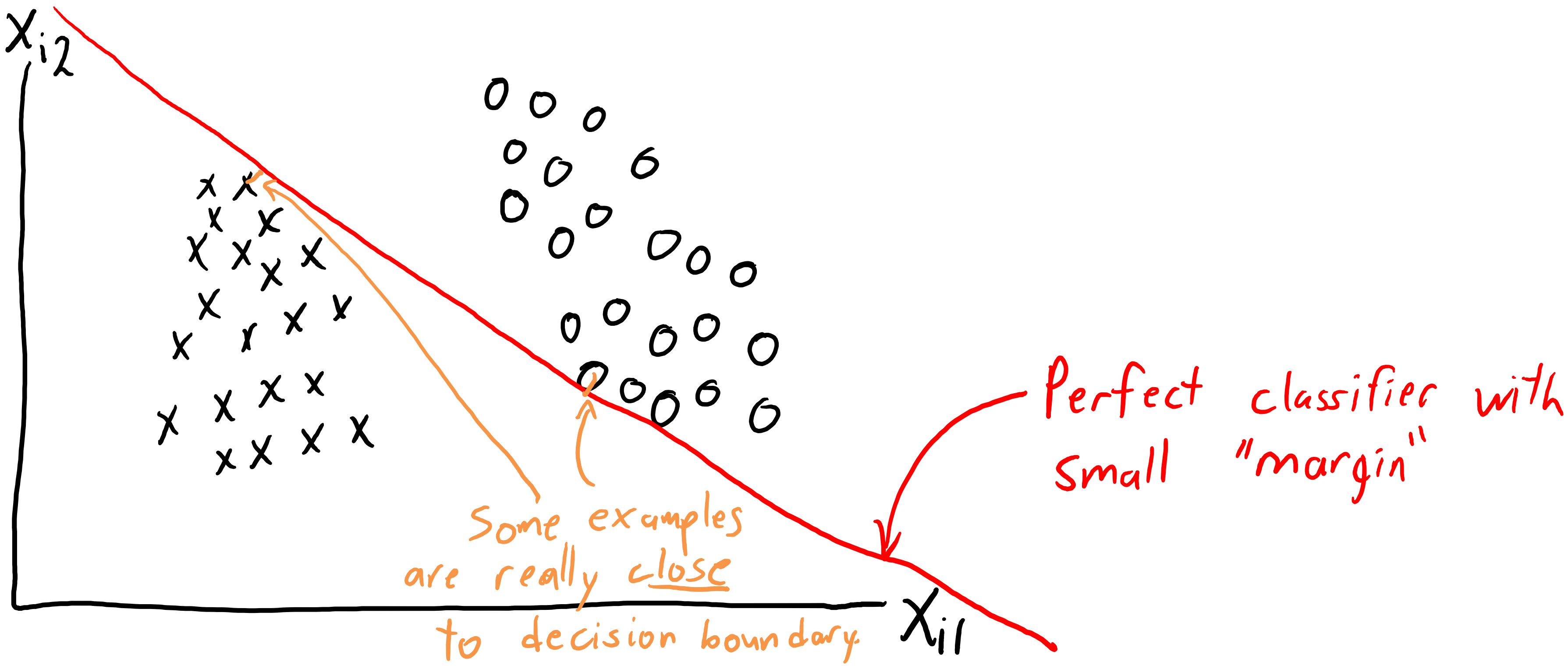
# Maximum-Margin Classifier

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



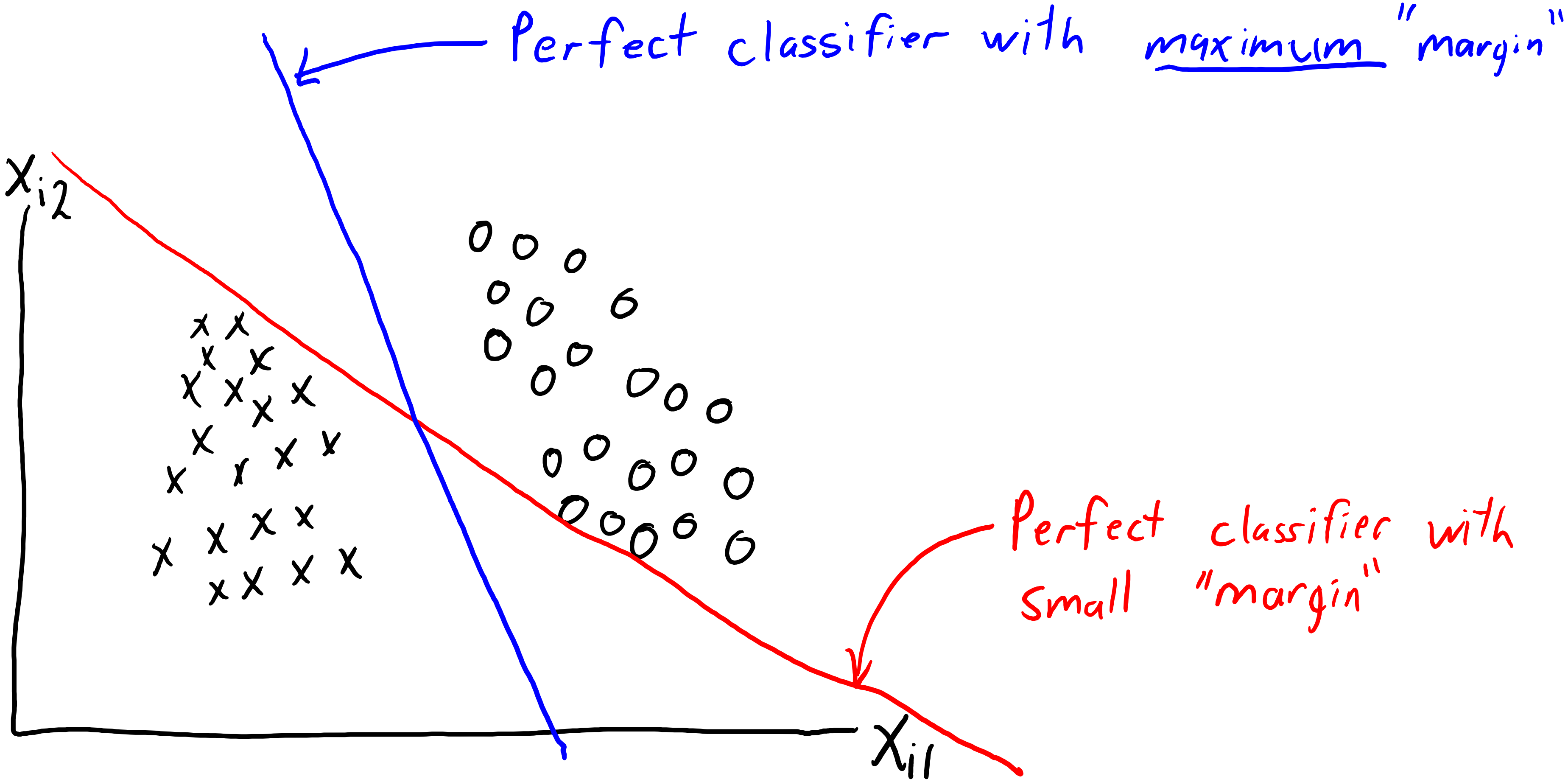
# Maximum-Margin Classifier

- 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.



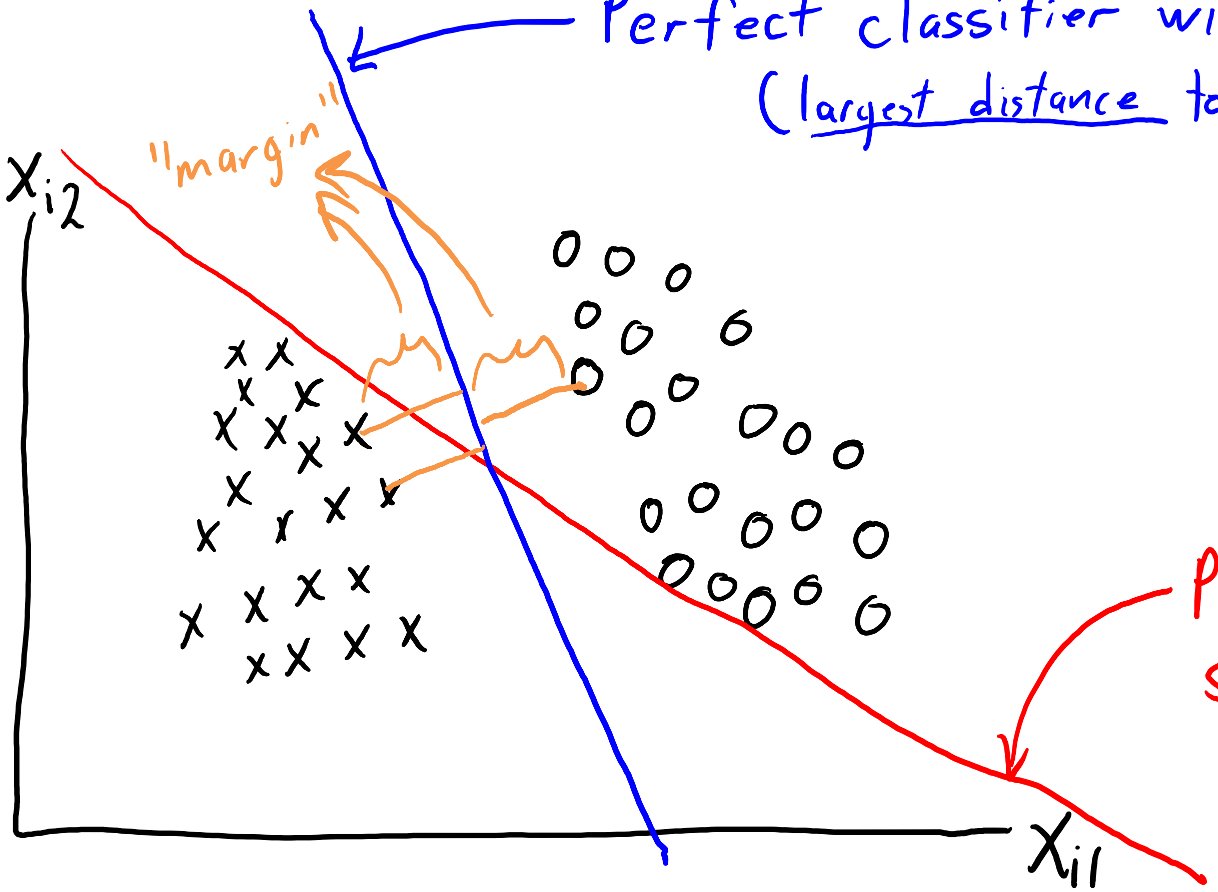
# Maximum-Margin Classifier

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

Why maximize margin?

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

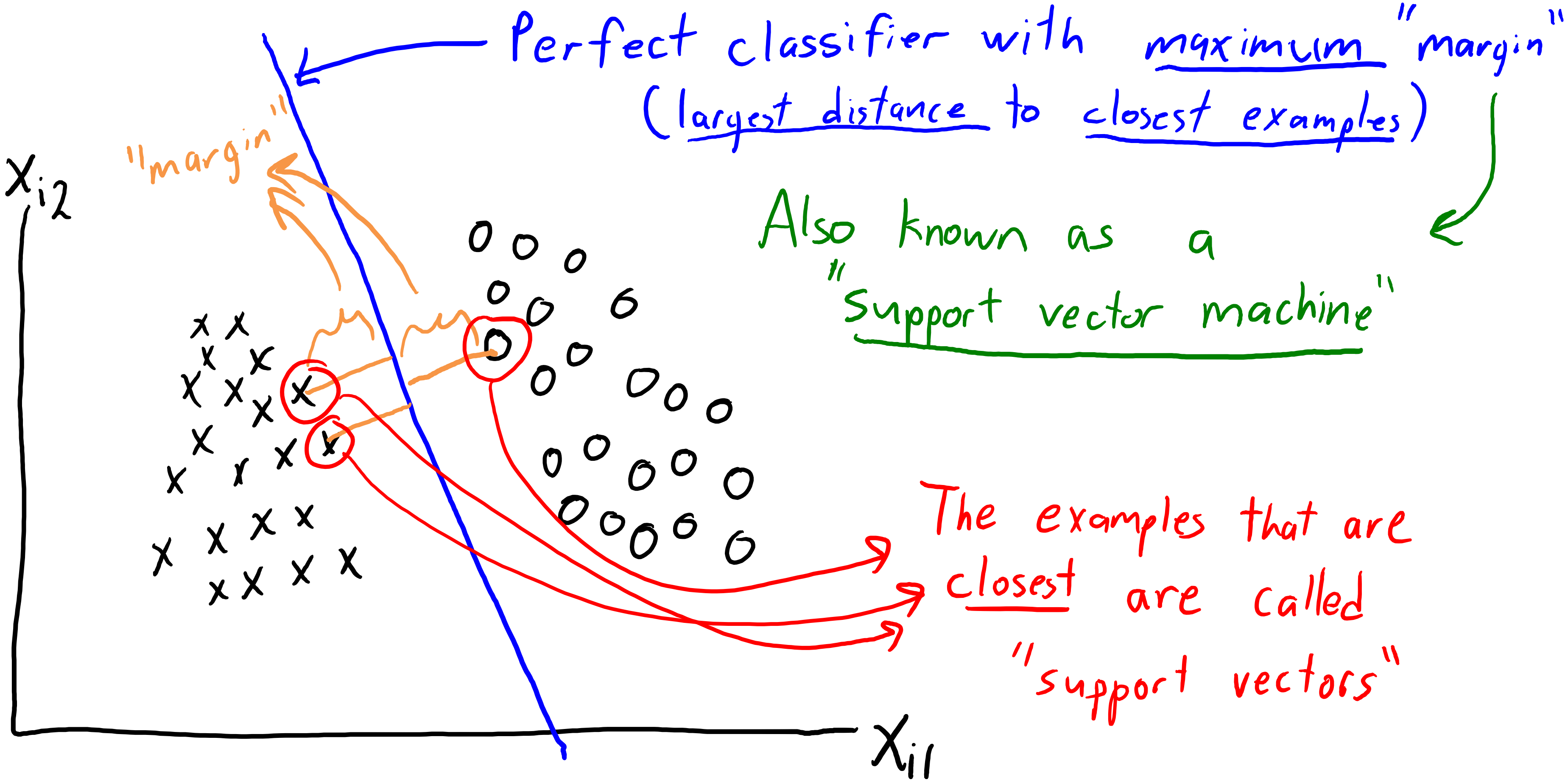
Perfect classifier with maximum "margin" (largest distance to closest examples)




Perfect classifier with small "margin"

# Maximum-Margin Classifier

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



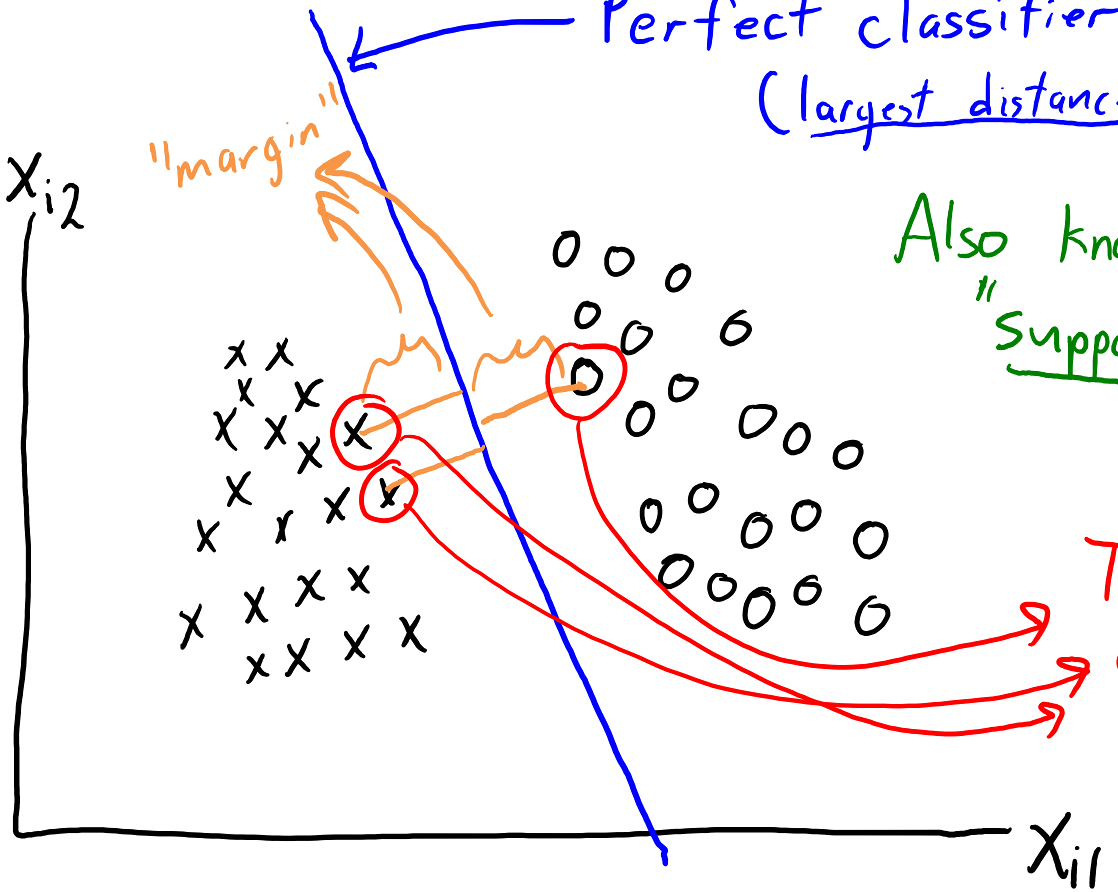
# Maximum-Margin Classifier

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

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

Also known as a "support vector machine"

Final classifier only depends on support vectors




The examples that are closest are called "support vectors"

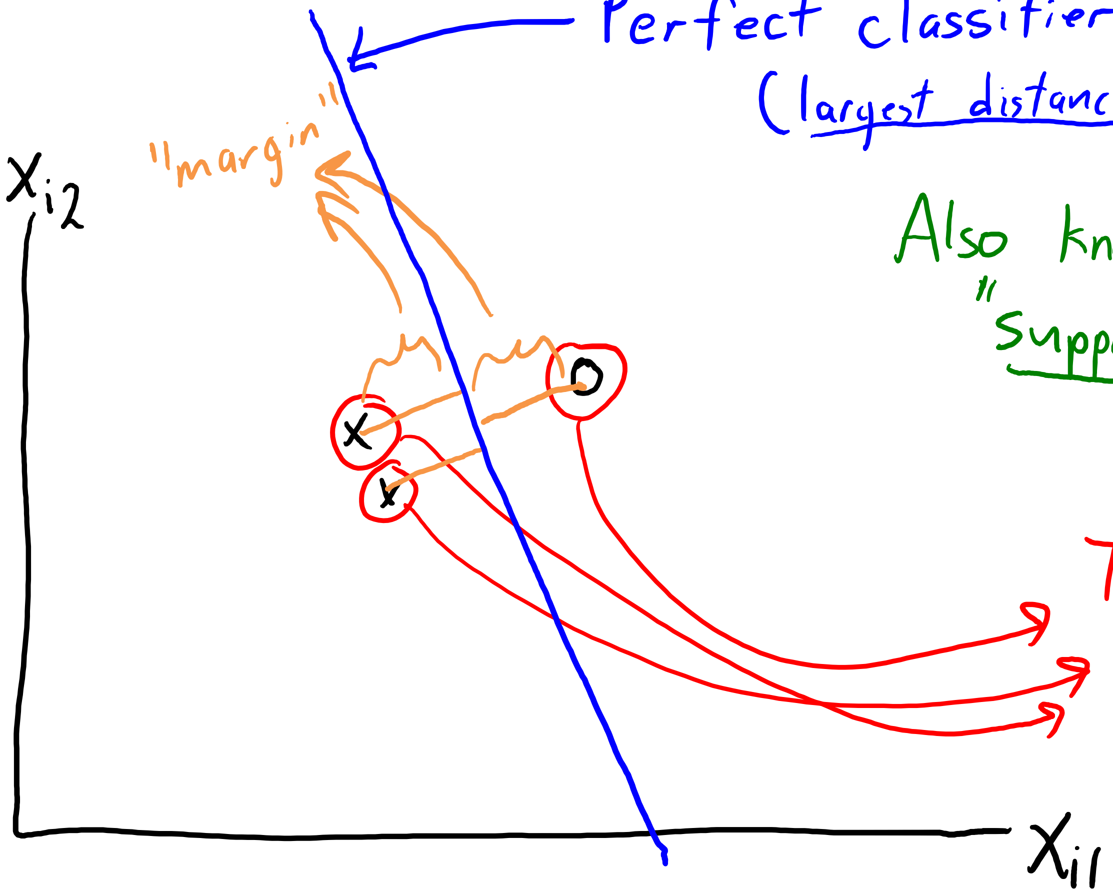
# Maximum-Margin Classifier

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

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

Also known as a "support vector machine"

Final classifier only depends on support vectors




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

The examples that are closest are called "support vectors"

# Support Vector Machines

- For linearly-separable data, support vector machine (SVM) minimizes: $f(w) = \frac{1}{2}\|w\|^2$ (← this sounds insane, but see next slide)
  - Subject to the constraints that:
    $$\left.\begin{array}{ll} w^T x_i \geq 1 & \text{for } y_i = 1 \\ w^T x_i \leq -1 & \text{for } y_i = -1 \end{array}\right\}$$
    Or we can write both cases as $y_i w^T x_i \geq 1$
- We played this trick ($y_i w^T x_i$) with the logistic loss as well:

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

  - You can think of it as "mirroring" the loss (or constraints) across the y-axis for positive vs. negative examples

# (bonus slide) from margin to ||w||

- The statement on the previous slide is very non-obvious. Starting from:

$$\max_w \gamma = \max_w \min_i \frac{w^\top x_i}{||w||} = \max_w \frac{1}{||w||} \min_i w^\top x_i$$

  - Where $\gamma$ is the "margin" or "distance to the closest point"
  - The part on the right-hand side is just geometry
    - It's the formula for the distance from a point to a plane
    - For now we assume the decision boundary passes through the origin (but argument extends to an intercept)
- The next step is to notice that the choice of *w* is non-unique
  - Because it is invariant to scaling (it's just representing a *direction*)
  - So we insist that $\min_i w^\top x_i = 1$ to pin down the scaling. This leaves the objective

$$\arg\max_w \frac{1}{||w||} = \arg\min_w ||w|| = \arg\min_w ||w||^2$$

  - Which is what we had on the previous slide
  - Note that the dependence of w* on the {$x_i$} has been moved into a constraint, but…
  - We get this equality constraint "for free" by using the inequality constraints on the previous slide
    - \`w\` wants to be small so if we use +1/-1 then we'll have equality for the closest point(s)
    - This is a bit subtle

# Support Vector Machines

- For non-separable data, try to minimize violation of constraints:

We want $y_i w^T x_i \geq 1$

or equivalently $0 \geq 1 - y_i w^T x_i$

Since we can't satisfy this for all 'i', let's add "slack" $\beta_i \geq 0$ to each constraint:

$\beta_i \geq 1 - y_i w^T x_i$

- For non-separable data, we usually define SVMs as minimum of:

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

Hinge loss for example 'i':

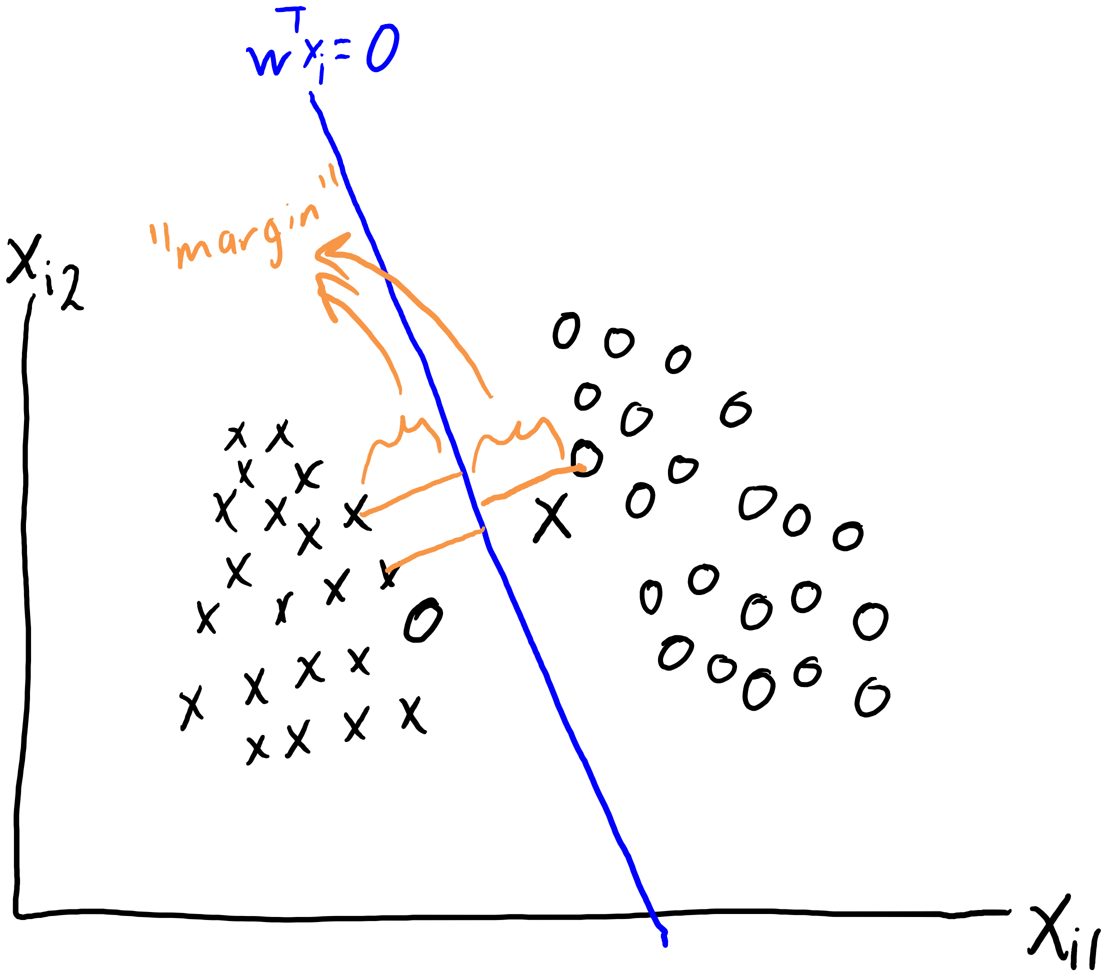
it's the amount we violate $y_i w^T x_i \geq 1$

"slack"

Original SVM objective: encourages large margin.

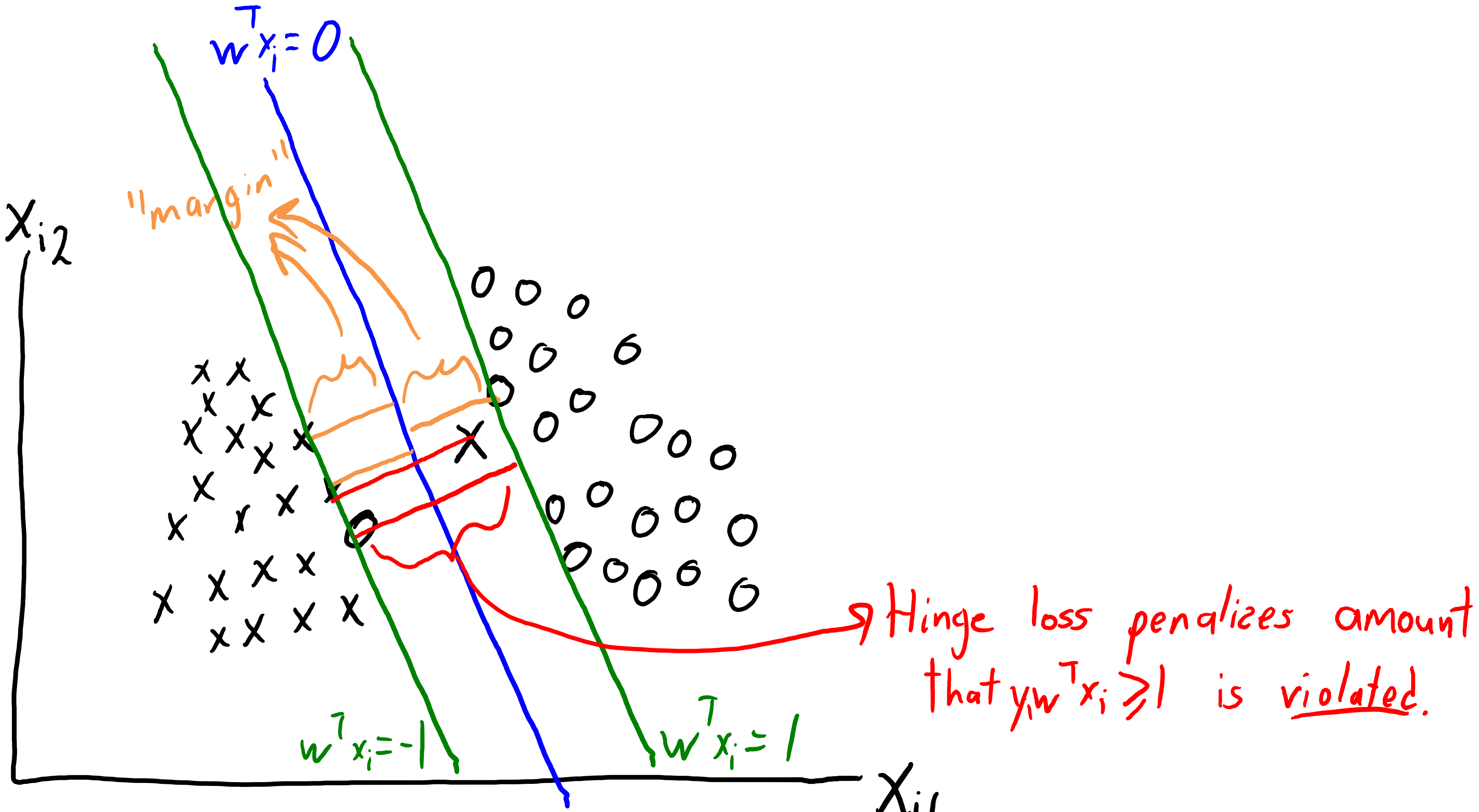
# Support Vector Machines for Non-Separable

- Non-separable case:



# Support Vector Machines for Non-Separable

- Non-separable case:



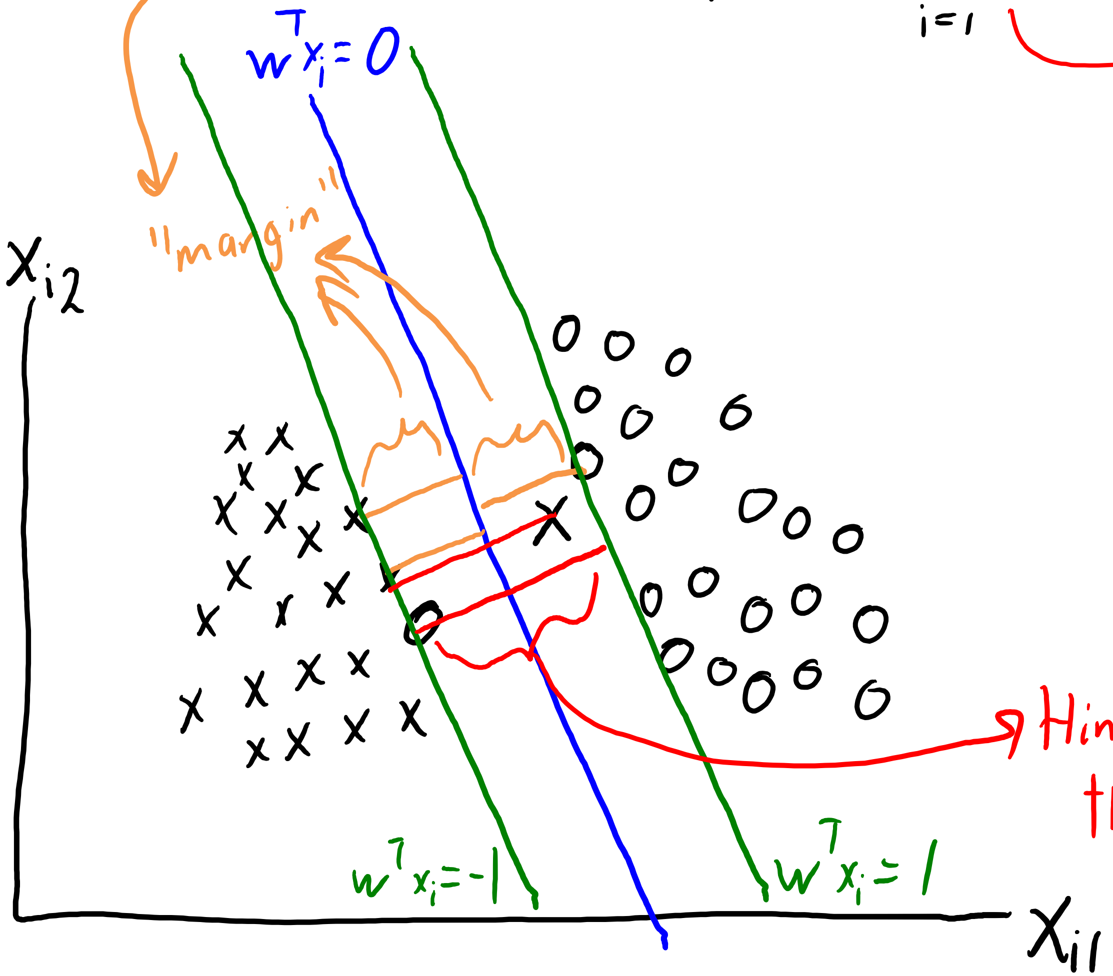
# Support Vector Machines for Non-Separable

- Non-separable case:

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

Logistic regression can be viewed as smooth approximation to SVMs.

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




$\lambda$ 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.

# Support Vector Machines for Non-Separable

- Non-separable case:

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

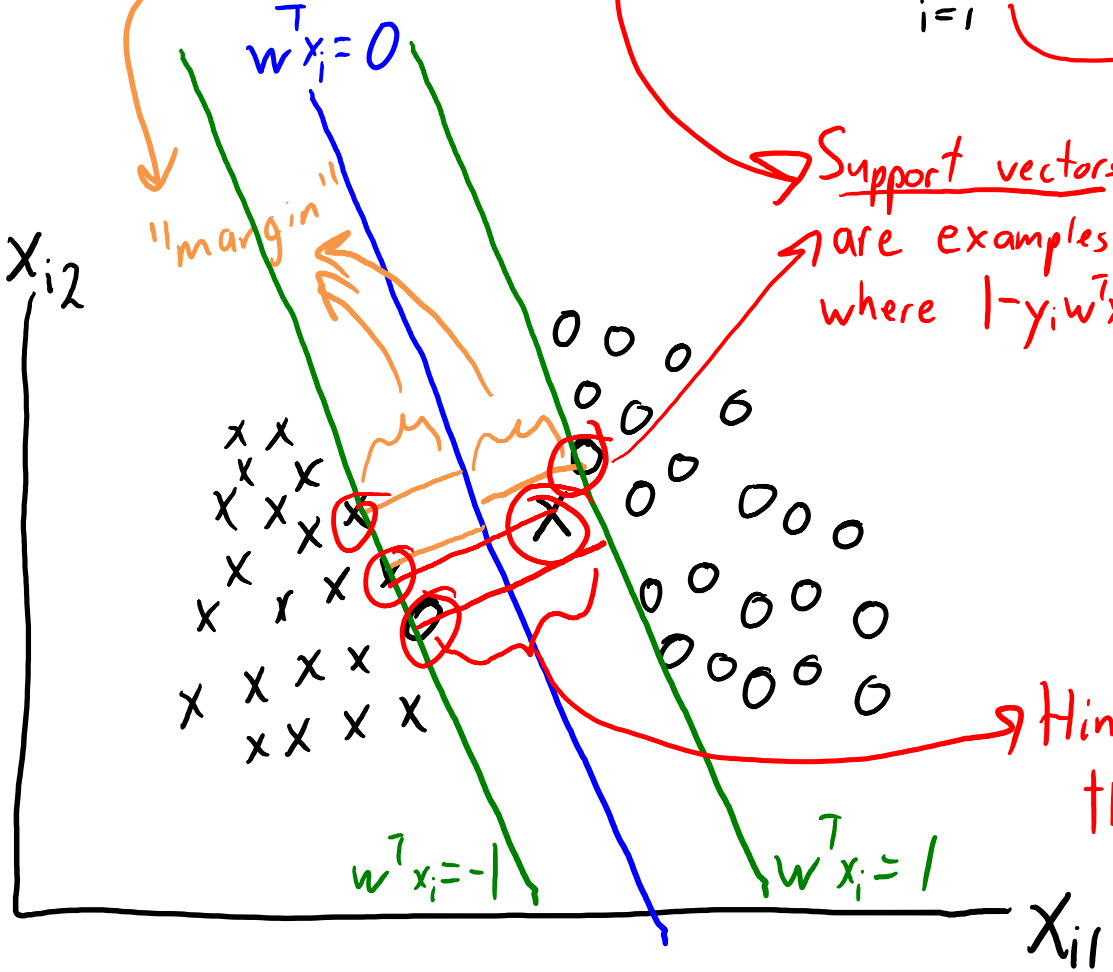
Support vectors are examples 'i' where $1 - y_i w^T x_i \geq 0$

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

$\lambda$ controls trade-off between having large margin and classifying examples correctly.

Logistic regression can be viewed as smooth approximation to SVMs.

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



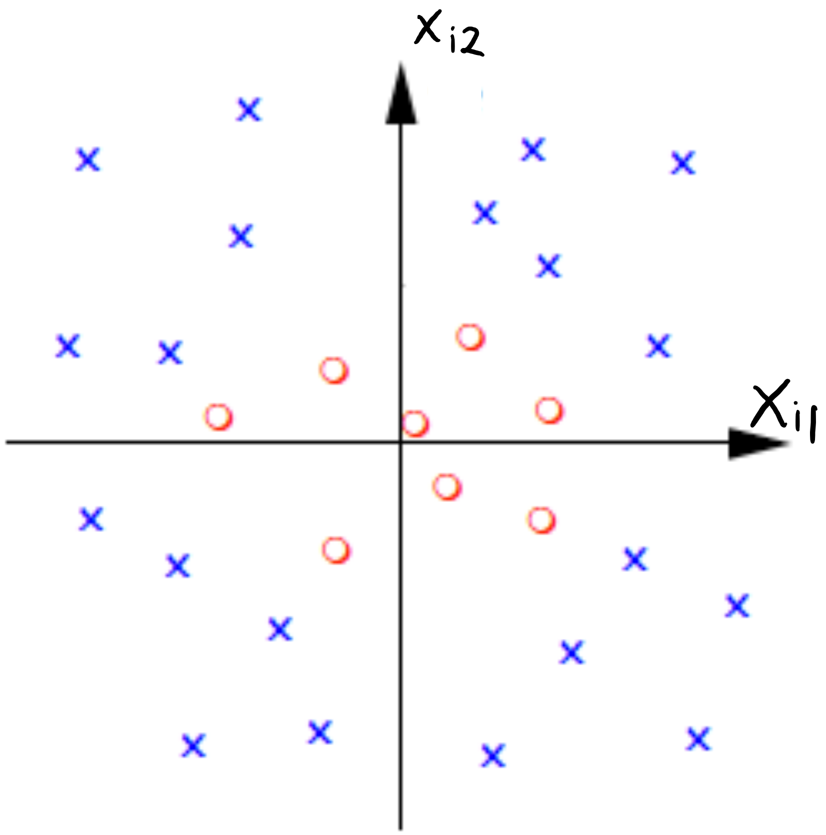

This is precisely because the loss is not flat (so there is still a "signal") from these examples.

# Support Vectors

- When you fit an SVM some training examples will be “support vectors”
- The support vectors have a particular interpretation in optimization theory (duality), but this is out of scope
- But for now you can think of them as…
  - The boundary would change if a moved/removed this point
  - Or equivalently, this point contributes a non-zero gradient
- Note that this is not true for any points with ordinary least squares or logistic regression
  - It’s considered an appealing property of SVMs.
- Note: this doesn’t mean SVMs are perfectly robust to outliers
  - If a point is misclassified it will be a support vector even if far away from the boundary (think about hinge loss)

# Support Vector Machines for Non-Separable

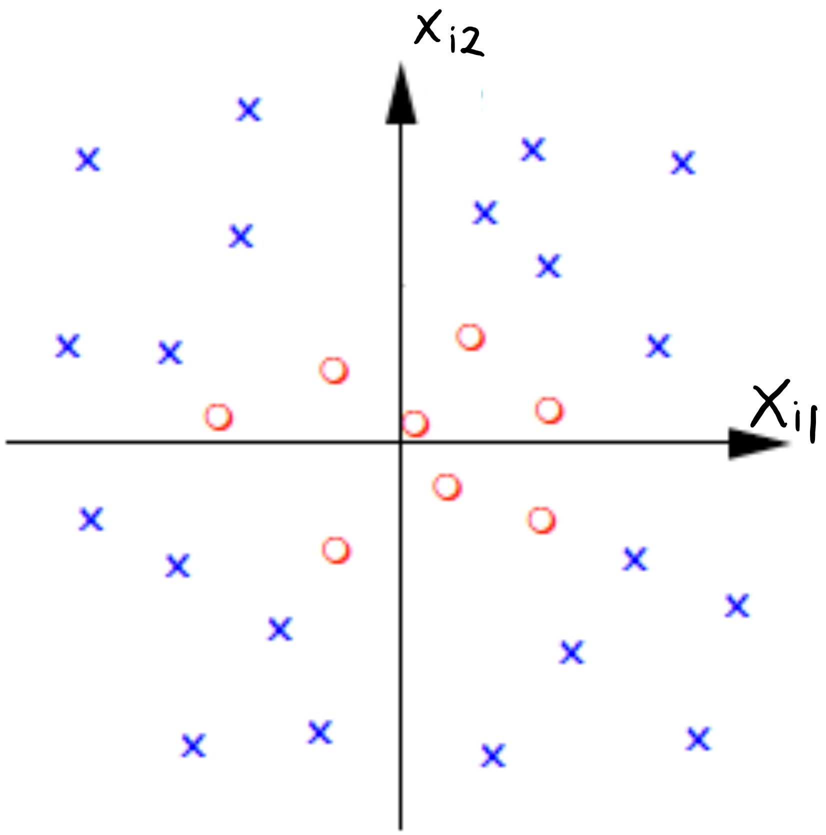
- What about data that is not even close to separable?

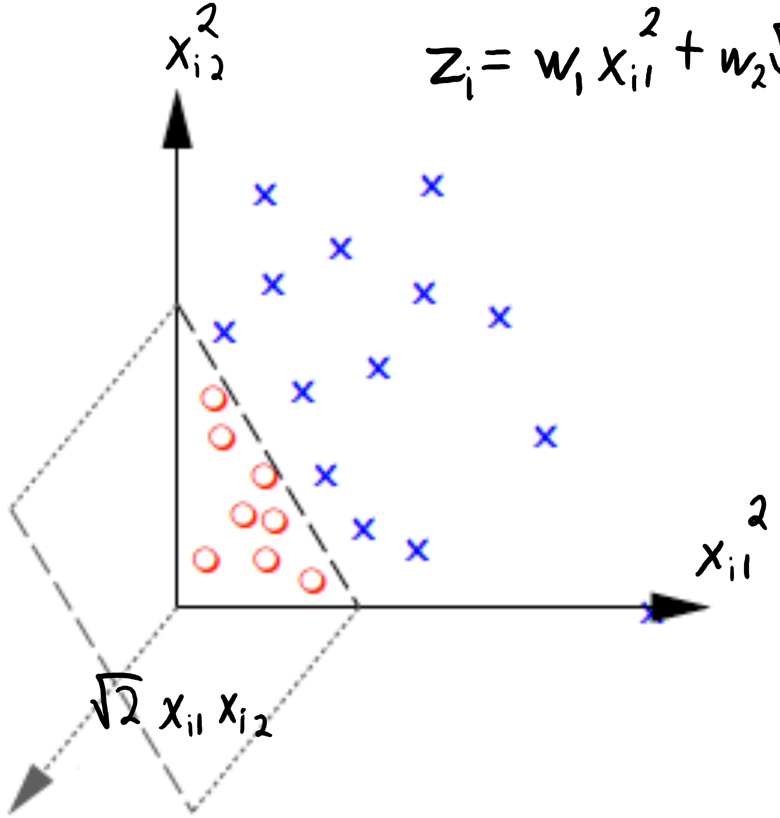


# Support Vector Machines for Non-Separable

- What about non-linear decision boundaries?
  - Recall our pal change of basis (change of features)

$$z_i = w_1 x_{i1}^2 + w_2 \sqrt{2} x_{i1} x_{i2} + w_3 x_{i2}^2$$



# Multi-Dimensional Polynomial Basis

- Recall fitting polynomials when we only have 1 feature:

$$y_i = w_0 + w_1 x_i + w_2 x_i^2$$

- We can fit these models using a change of basis:

$$X = \begin{bmatrix} 0.2 \\ -0.5 \\ 1 \\ 4 \end{bmatrix} \qquad Z = \begin{bmatrix} 1 & 0.2 & (0.2)^2 \\ 1 & -0.5 & (-0.5)^2 \\ 1 & 1 & (1)^2 \\ 1 & 4 & (4)^2 \end{bmatrix}$$

- How can we do this when we have a lot of features?

# Multi-Dimensional Polynomial Basis

- Approach 1: use polynomial basis for each variable.

$$X = \begin{bmatrix} 0.2 & 0.3 \\ 1 & 0.5 \\ -0.5 & -0.1 \end{bmatrix} \longrightarrow Z = \begin{bmatrix} 1 & 0.2 & (0.2)^2 & 0.3 & (0.3)^2 \\ 1 & 1 & (1)^2 & 0.5 & (0.5)^2 \\ 1 & 0.5 & (0.5)^2 & -0.1 & (-0.1)^2 \end{bmatrix}$$

bias; quadratic function of $x_{i1}$; quadratic function of $x_{i2}$

- But this is restrictive:
  - We should allow terms like '$x_{i1}x_{i2}$' that depend on feature interaction.
  - But number of terms in $X_{poly}$ is huge:
    - Degree-5 polynomial basis has $O(d^5)$ terms:

$$x_{i1}^5, x_{i1}^4 x_{i2}, x_{i1}^4 x_{i3}, \dots, x_{i1}^4 x_{id}, x_{i1}^3 x_{i2}^2, x_{i1}^3 x_{i3}^2, \dots, x_{i1}^3 x_{id}^2, \dots, x_{i2}^5, x_{i2}^4 x_{i3}, \dots \dots, x_{id}^5$$

- If reasonable 'n', we can do this efficiently using the kernel trick.

# Equivalent Form of Ridge Regression

- Recall L2-regularized least squares objective with basis matrix 'Z':

$$f(w) = \frac{1}{2}\|Zw - y\|^2 + \frac{\lambda}{2}\|w\|^2$$

- We showed that the solution is given by:

$$w = (\underbrace{Z^TZ + \lambda I}_{d \times d})^{-1}(Z^Ty)$$

- Using a "matrix inversion lemma" we can re-write this as:

$$w = Z^T(\underbrace{ZZ^T + \lambda I}_{n \times n})^{-1}y$$

- This is faster if n << d:
  - $Z^TZ$ is d-by-d while $ZZ^T$ is n-by-n.

# Predictions using Equivalent Form

- Given test data $\hat{X}$, predict $\hat{y}$ by forming and $\hat{Z}$ using:

$$\hat{y} = \hat{Z}w \leftarrow \boxed{w = Z^T(ZZ^T + \lambda I)^{-1}y}$$

$$= \underbrace{\hat{Z}Z^T}_{\hat{K}}(\underbrace{ZZ^T}_{K} + \lambda I)^{-1}y$$

$$= \hat{K}(K + \lambda I)^{-1}y$$

- Key observation behind kernel trick:
  - Predictions $\hat{y}$ only depend on features through K and $\hat{K}$.
  - If we have function that computes K and $\hat{K}$, we don't need the features.

# Gram Matrix

- The Gram matrix 'K' is defined by:

$$K = ZZ^T = \underbrace{\begin{bmatrix} — z_1 — \\ — z_2 — \\ \vdots \\ — z_n — \end{bmatrix}}_{Z} \underbrace{\begin{bmatrix} | & | & & | \\ z_1 & z_2 & \cdots & z_n \\ | & | & & | \end{bmatrix}}_{Z^T}$$

$$= \begin{bmatrix} z_1^T z_1 & z_1^T z_2 & \cdots & z_1^T z_n \\ z_2^T z_1 & z_2^T z_2 & \cdots & z_2^T z_n \\ \vdots & & \ddots & \\ z_n^T z_1 & z_n^T z_2 & \cdots & z_n^T z_n \end{bmatrix}$$

- 'K' contains the inner products between all training examples.

# Gram Matrix

- The Gram matrix ‘K’ is defined by:

$$K = ZZ^T = \begin{bmatrix} z_1^T z_1 & z_1^T z_2 & \cdots & z_1^T z_n \\ z_2^T z_1 & z_2^T z_2 & \cdots & z_2^T z_n \\ \vdots & & \ddots & \\ z_n^T z_1 & z_n^T z_2 & \cdots & z_n^T z_n \end{bmatrix}$$

- ‘K’ contains the inner products between all training examples.
- ‘$\hat{K}$’ contains the inner products between training and test examples.
- Kernel trick:
  - I want to use a basis $z_i$ that is too huge to store.
  - But I only need $z_i$ to compute $K = ZZ^T$ and $\hat{K} = \hat{Z}Z^T$.
  - I can use this basis if I have a kernel function that computes $k(x_i,x_j) = z_i^T z_j$.

# Example: polynomial kernel

- Consider two examples $x_i$ and $x_j$ for a 2-dimensional dataset:

$$x_i = (x_{i1}, x_{i2}) \qquad x_j = (x_{j1}, x_{j2})$$

- And consider a particular degree-2 basis:

$$z_i = (x_{i1}^2, \sqrt{2}\, x_{i1} x_{i2}, x_{i2}^2) \qquad z_j = (x_{j1}^2, \sqrt{2}\, x_{j1} x_{j2}, x_{j2}^2)$$

- We can compute inner product $z_i^T z_j$ without forming $z_i$ and $z_j$:

$$
\begin{aligned}
z_i^T z_j &= x_{i1}^2 x_{j1}^2 + (\sqrt{2}\, x_{i1} x_{i2})(\sqrt{2}\, x_{j1} x_{j2}) + x_{i2}^2 x_{j2}^2 \\
&= x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{i2} x_{j1} x_{j2} + x_{i1}^2 x_{i2}^2 \\
&= (\underbrace{x_{i1} x_{j1} + x_{i2} x_{j2}}_{x_i^T x_j})^2 \qquad \text{"completing the square"} \\
&= (x_i^T x_j)^2 \quad \leftarrow \text{No } \underline{\text{need}} \text{ for } z_i \text{ to compute } z_i^T z_j
\end{aligned}
$$

# Summary

- Support vector machines maximize margin to nearest data points.
- High-dimensional bases allows us to separate non-separable data.
- Kernel trick allows us to use high-dimensional bases efficiently.

- Next time:
  - A few more slides on SVMs/kernels, and then review.