Introduction to Support Vector Machines

THE KNOWLEDGE HOUSE

Agenda - Schedule

1. Kahoot
2. Pre-Class Review
3. Support Vector Machines
4. Break
5. Kernel Trick

***Kernel**: most essential or most vital part of some idea*

Agenda - Goals

- ...

Kahoot

Kahoot!

Week 6 Kahoot - Naive Bayes & kNN

Pre-Class Work Review

Pre-Class Work Review

Let’s go over the following material as a class:

- **ISLP Chapter 9 - 9.1**
- **ISLP Chapter 8 - 8.1.1 - 8.1.4; 8.2.1 - 8.2.3**

What questions do you have about this week's material?

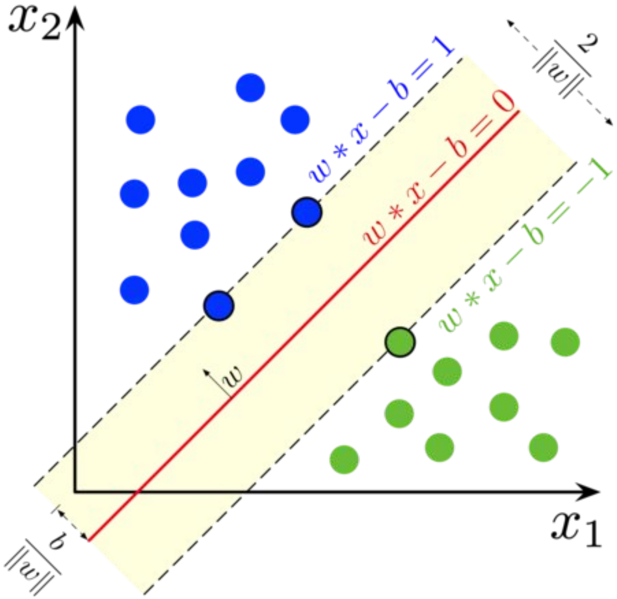
Support Vector Machines (SVMs)

Support Vector Machines

We’ve discussed the concept of a “decision boundary” in last weeks kNN lecture.

This boundary was a **consequence** of our classifier, as opposed to a direct creation.

However SVM’s take this idea and run with it: **how can we find the best separation (hyperplane) between data classes?**



An Aside - Hyperplanes

First off, let’s define what a hyperplane is.

Let’s break down the word:

hyper-plane

- **Hyper:** Think back to the term *“hyperparameter.”* How did our hyperparameters relate to our regular parameters?

- **Plane:** What is a plane?

GPT Prompt “Make me an image of a hyper hamster. He's running at the speed of sound with a carrot in its mouth. Also it's fuzzy.”

This is why it's important to nail terminology down. A word used in one context will be reused again.

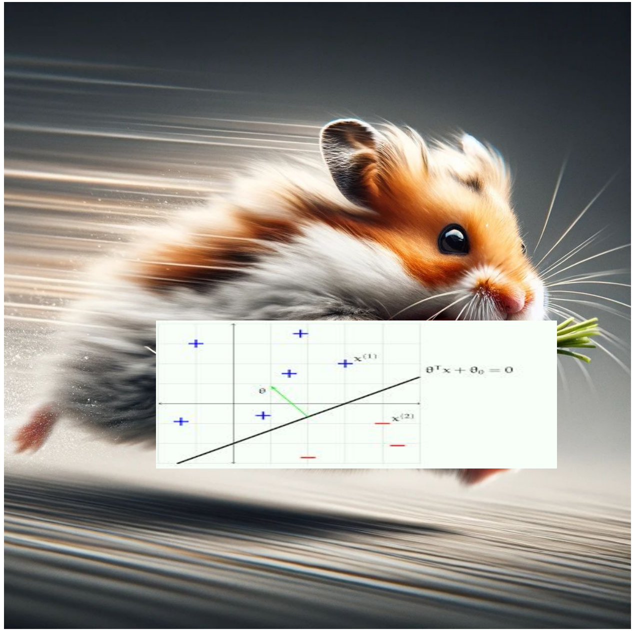
An Aside - Hyperplanes

First off, let's define what a hyperplane is.

Let's break down the word:

hyper-plane

- **Hyper:** latin for *above;* denotes concepts that are extensions into higher dimensions or complex forms
- **Plane:** a "flat" dimensional surface



GPT Prompt "Make me an image of a hyper hamster. He's running at the speed of sound with a carrot in its mouth. Also it's fuzzy."

An Aside - Hyperplanes

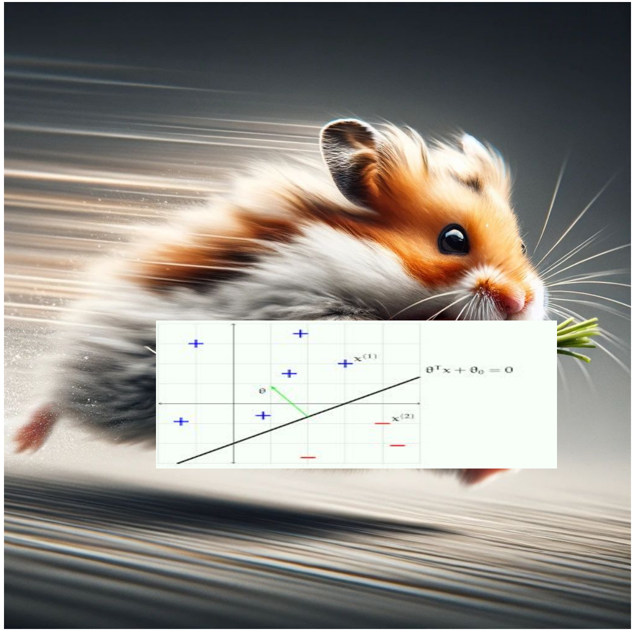
First off, let’s define what a hyperplane is.

hyperplane

Definition:

> A **flat, affine** (allows for parallel relationships), **n-dimensional subspace**

This is why it's important to nail terminology down. A word used in one context will be reused again.



GPT Prompt “Make me an image of a hyper hamster. He's running at the speed of sound with a carrot in its mouth. Also it's fuzzy.”

An Aside - Hyperplanes

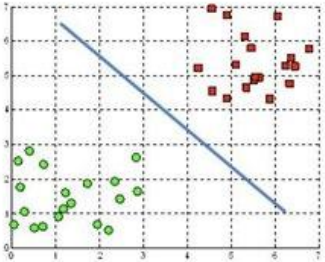
We will increasingly rely on mathematical terminology to discuss machine learning concepts.

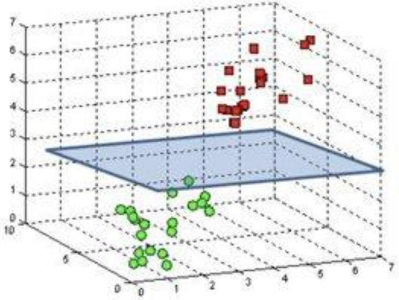
“A hyperplane is a set described by a single scalar product equality.”

betas are coefficients   **HINT HINT**

x are the predictor values of a specific sample

Each sample will either be >0 (**above** the hyperplane) or <0 (**below** the hyperplane) given the hyperplane equation.





$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0$$

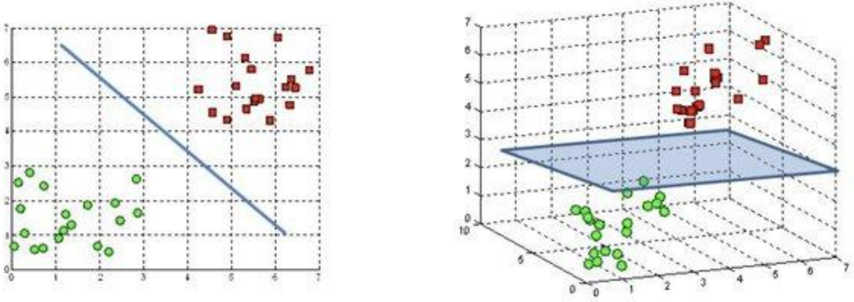
An Aside - Hyperplanes

We will increasingly rely on mathematical terminology to discuss machine learning concepts.

“A hyperplane is a set described by a single scalar product equality.”

betas are coefficients   HINT HINT

x are the dimensions of the search-space



$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0$$

Hyperplane Dot Product

*Whenever you see x with a subscript, just consider that to be another dimension.

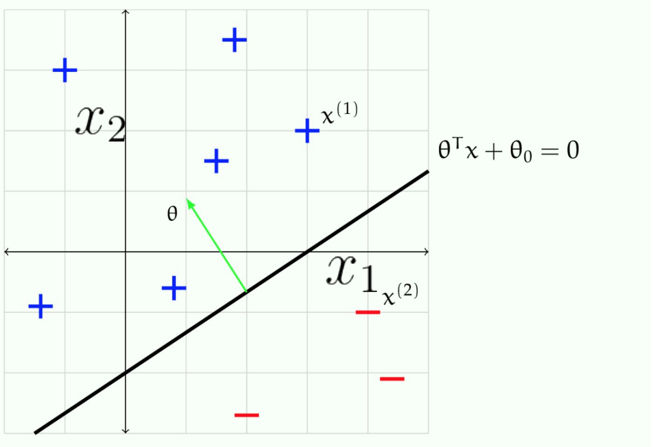
In this example x1 = x axis; x2 = y axis

This hyperplane can be defined as the following **dot product** when considering only two dimensions

Our Beta's are a **vector of weights**

Our X is a **vector of predictors**

We're missing one last component of this formula…



$$\begin{vmatrix}\beta_1 \\ \beta_2\end{vmatrix} * |x_1 x_2| \longrightarrow \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

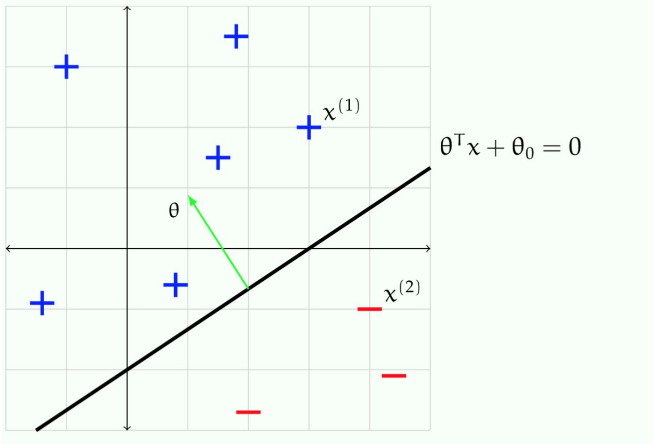
Hyperplane Dot Product

This hyperplane can be defined as the following **dot product**.

Our Beta's are a **vector of weights**

Our X is a **vector of predictors**

We're missing one last component of this formula…

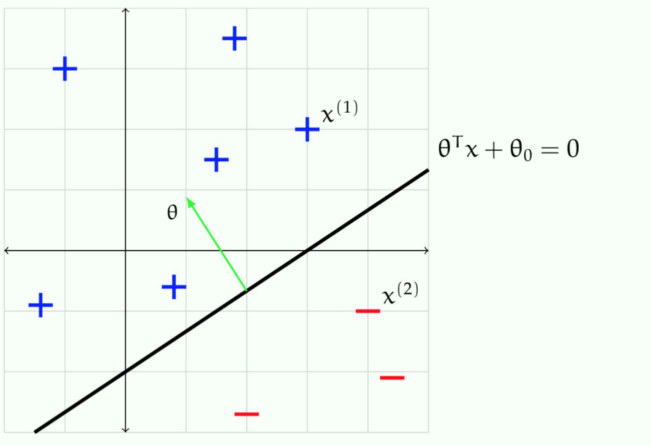


the "bias" or intercept

$$\beta_0 + \begin{vmatrix}\beta_1 \\ \beta_2\end{vmatrix} * |x_1 x_2| \longrightarrow \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

Hyperplane Dot Product

Also it appears that we "tilted" our beta vector on its side. What is this called again in linear algebra?

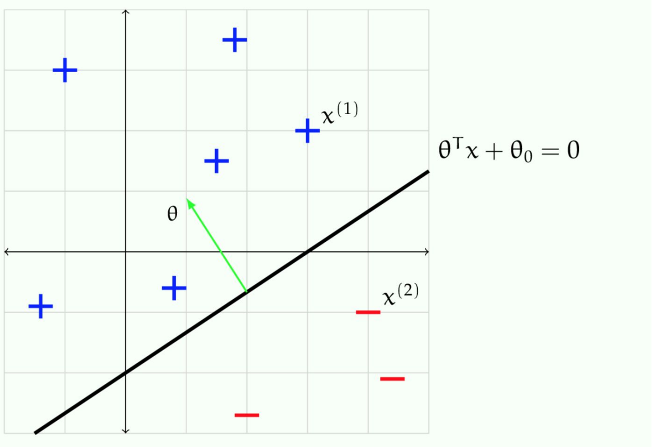


$$\beta_0 + \begin{vmatrix} \beta_1 \\ \beta_2 \end{vmatrix} * |x_1 x_2| \longrightarrow \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

Hyperplane Dot Product

Also it appears that we "tilted" our beta vector on its side. What is this called again in linear algebra?

Transpose. We summarize this equation via the following:



We will be relying on this common language ("*lingua franca*") of notation going forward.

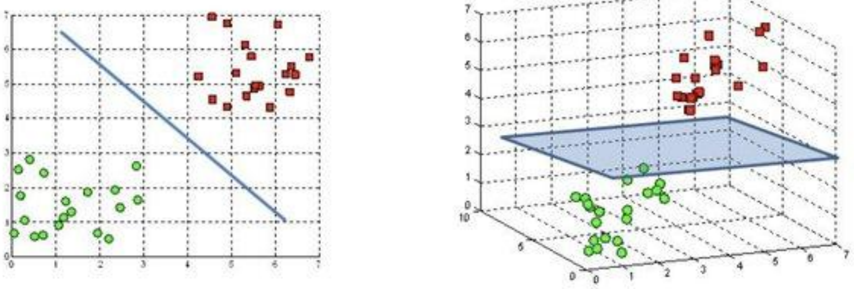
$$\beta_0 + \beta^T X \longrightarrow \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

An Aside - Hyperplanes

Geometrically, this results in a “*A subspace whose dimension is one less of the ambient space*.”

We will see how this concept of “more dimensions” **is important to SVM’s**.

- *In a 2D space, a hyperplane is a line*
- *In a 3D space, a hyperplane is a plane*
- *In a 4D space, a hyperplane is a cube (huh???)*



$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0$$

Support Vector Machines - Hyperplanes

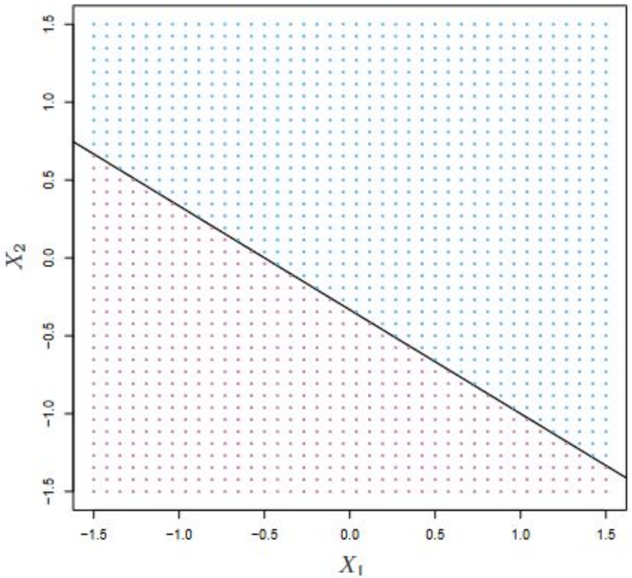
It turns out that for a sample who's X values (predictors) result **in a value greater than 0**, we state that we are above the hyperplane.

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p > 0.$$

For a sample whose X values give us a **value less than 0**, we are below the hyperplane.

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p < 0,$$

2 different spaces...**what kind of classification problem is the hyperplane ideal for?**



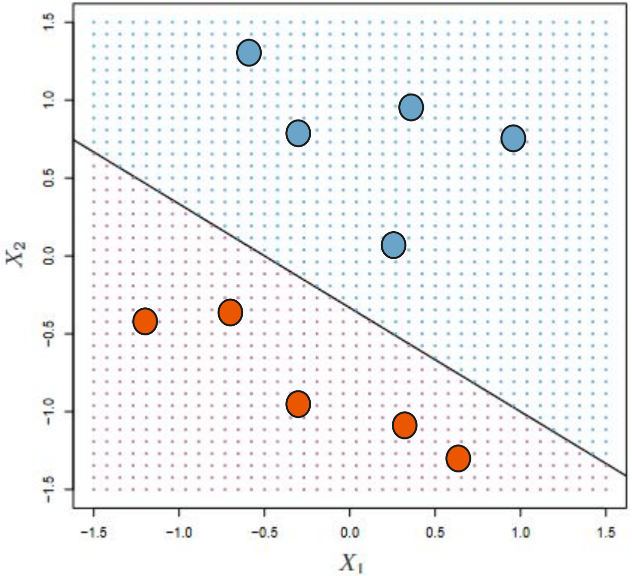
Support Vector Machines - Hyperplanes

Binary classification!

Suppose we have a set of test samples that we must predict to determine their class.

Which parts of the equation must we *learn* in order to find the best separating hyperplane?

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0$$



*What assumptions are we making about the separability of our data?

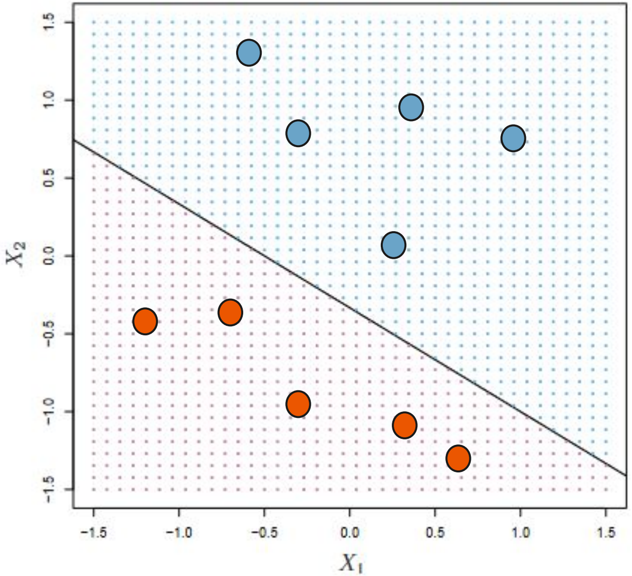
Support Vector Machines - Hyperplanes

The beta's!

We discover which beta's will cause us to calculate values > 0 **(above the hyperplane)** for "correct" classes.

At the same time, we also look for beta's which will cause the equation to be < 0 **(below the hyperplane)** for "incorrect" classes.

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0$$



Support Vector Machines (SVM)

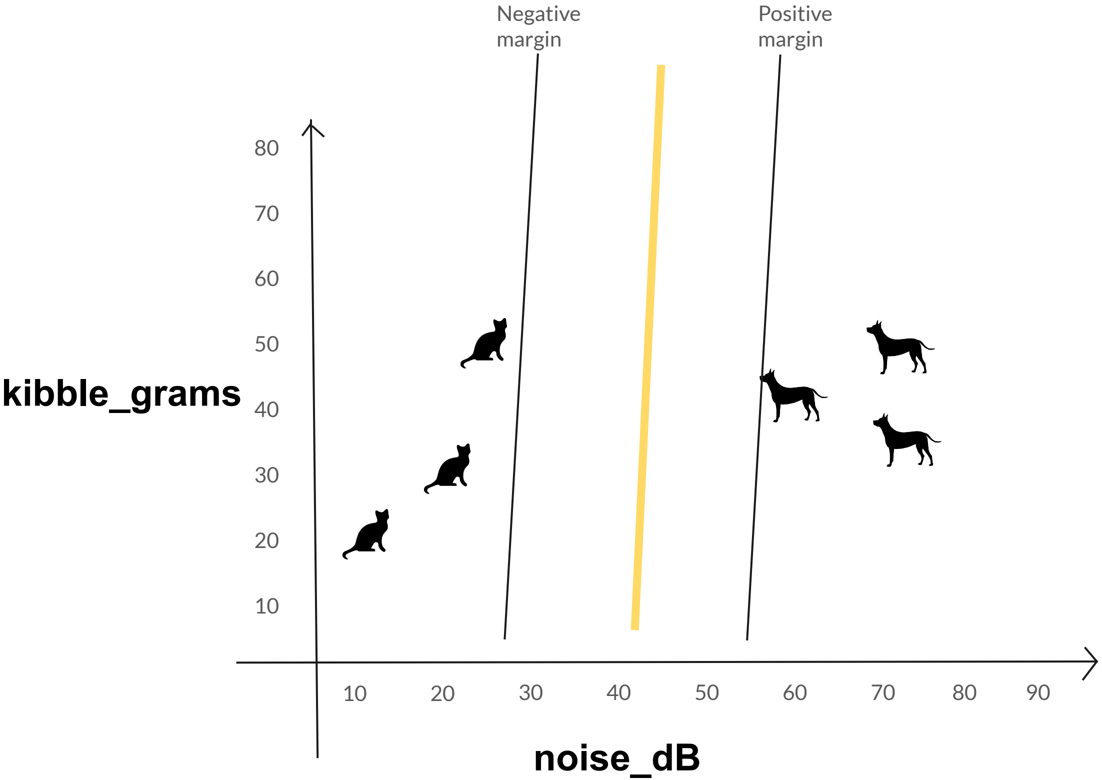
The SVM **provides a solution** to calculating the best possible weights for our hyperplane given a set of training data.

SVM is a **supervised learning binary classifier**. Much like logistic regression, we learn these coefficients through optimization.

Therefore, we do not need to “carry around” **a dataset to form predictions.** Instead, (much like logistic regression) we learn optimal coefficients (*weights*) for our given training predictors.

The concept of an SVM relies on us building up to a couple of concepts:

- *Maximal margin classifiers*
- *Support Vector Classifier*
- *Kernel Trick*



*In the words of Josh Starmer (StatQuest), if you don't like cats and dogs, think of fraud vs not-fraud; spam vs not-spam; a basketball upset vs not a basketball upset.

The first concept we will tackle is a "maximal margin."

$$\max_{\beta_0,\beta_1,\ldots,\beta_p} \quad M$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq M \quad \text{for all } i = 1, \ldots, n.$$

$$y_1, \ldots, y_n \in \{-1, 1\}$$

The first concept we will discuss is the **maximal margin classifier**. This is summarized in the following formula.

We have many possible hyperplanes to choose from. We want to choose the hyperplane that maximizes the distance from our points. **Imagine our hyperplane really wants us to respect personal space.**

$$\max_{\beta_0,\beta_1,\ldots,\beta_p} \quad M$$

Find the betas that maximize the width M (**margin**)

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq M \quad \text{for all } i = 1, \ldots, n.$$

$$y_1, \ldots, y_n \in \{-1, 1\}$$

Let’s break down this formula.

$$\max_{\beta_0,\beta_1,\ldots,\beta_p} M$$

Find the betas that maximize the width M (**margin**)

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$

Subject to the constraint that the squared-sum of betas is equal to 1

$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq M \quad \text{for all } i = 1, \ldots, n.$$

$$y_1, \ldots, y_n \in \{-1, 1\}$$

$$\max_{\beta_0,\beta_1,\ldots,\beta_p} M$$

Find the betas that maximize the width M (**margin**)

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1,$$

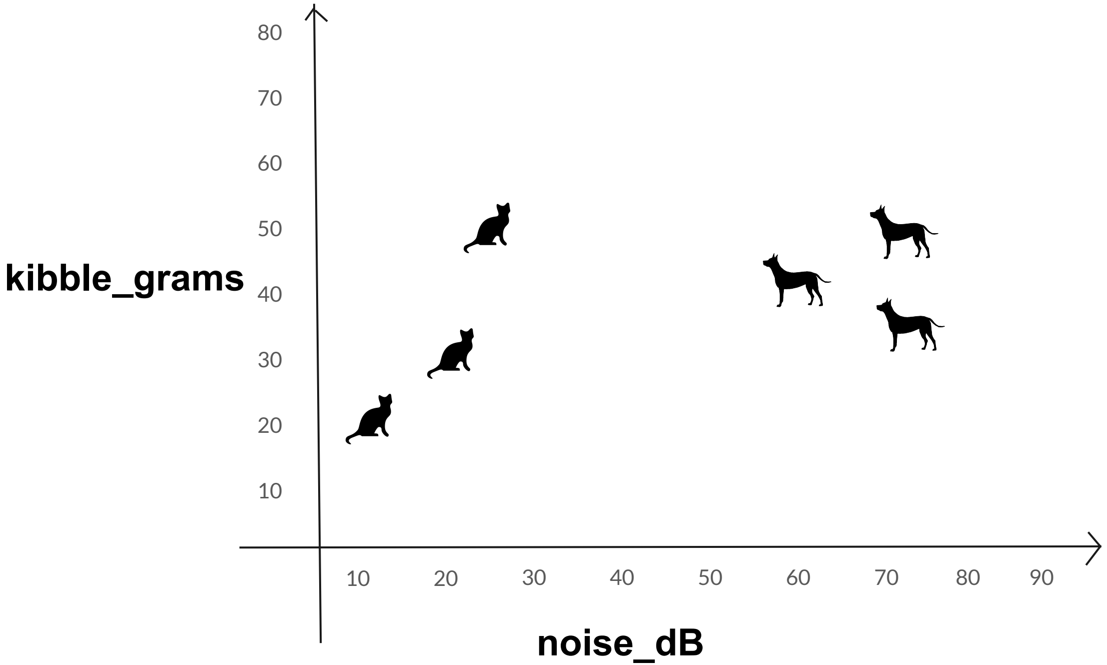
Subject to the constraint that the squared-sum of betas is equal to 1

$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq M \quad \text{for all } i = 1, \ldots, n.$$

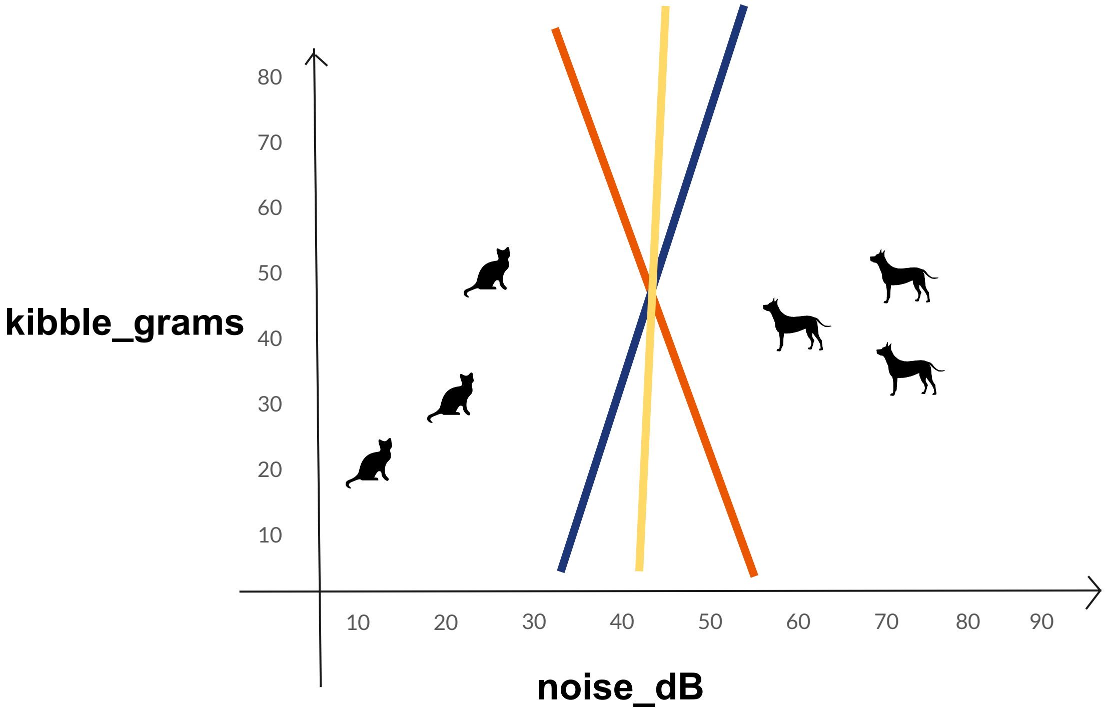
$$y_1, \ldots, y_n \in \{-1, 1\}$$

Such that all correctly classified data-points are spaced greater than or equal to the **maximal margin** of our hyperplane

Where y is either 1 for positive cases (1) or-1 for negative cases (0)

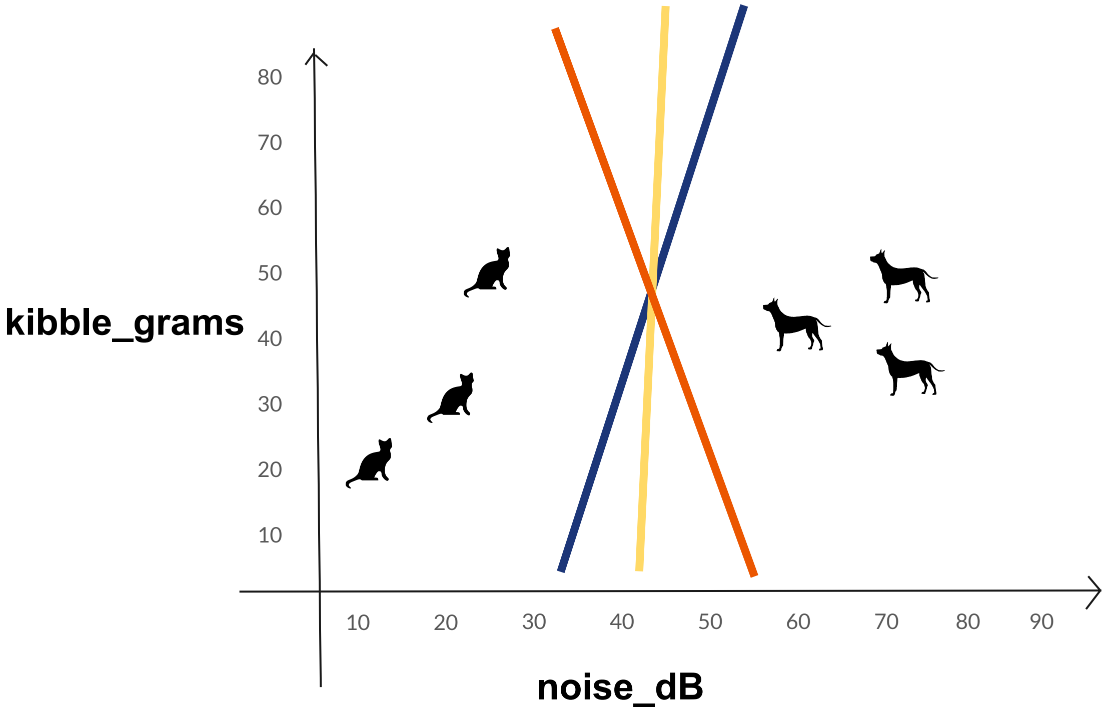


Let’s bring back our dogs and cats. I remove hamsters since we’re only looking at **binary classifications**.



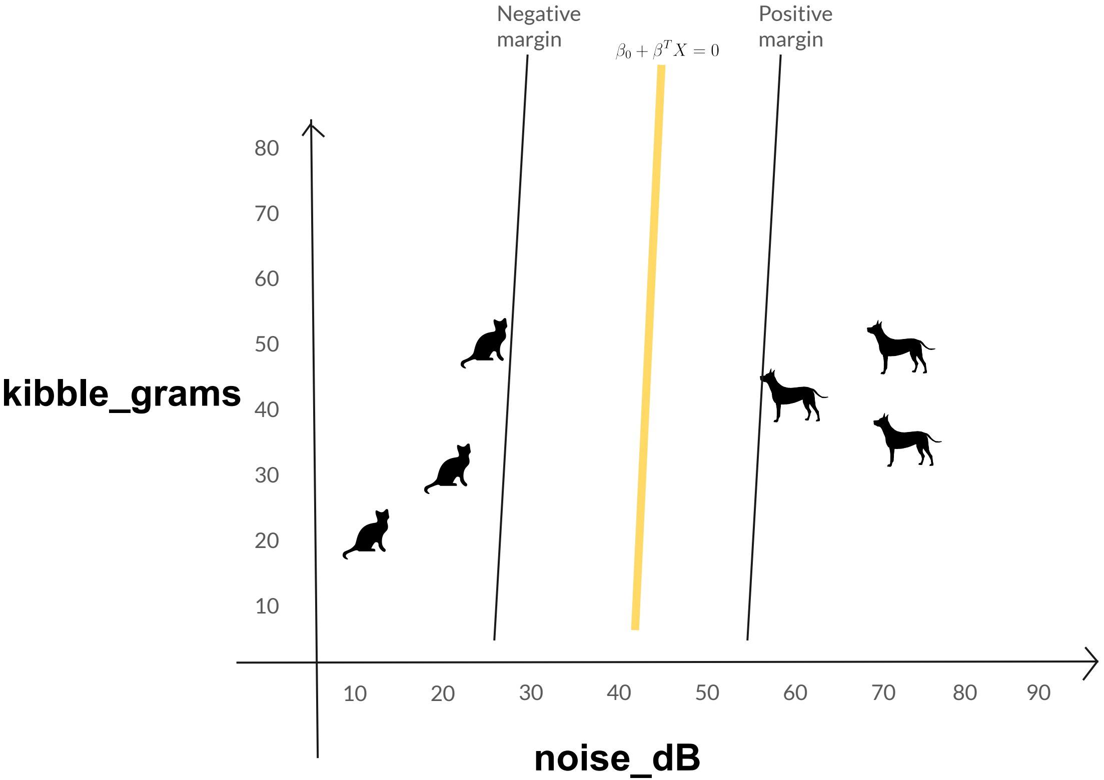
*a line (or a “flat affine 1-dimensional subspace”) separates our classes

There are many possible hyperplanes that could separate our cases (*we’ll say dogs are the positive case, sorry cat people*)



Margin: the **minimal distance for each training observation to a separating hyperplane**.

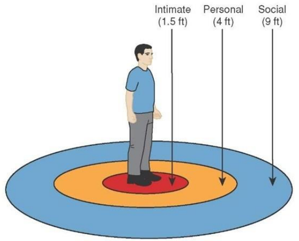
We want the hyperplane that has **maximal margins** (the one that respects personal space of both classes the most). Which hyperplane is that? (feel free to eyeball it).



*the phrase “maximum of a minimal” distance could be trippy, so just keep this in mind.

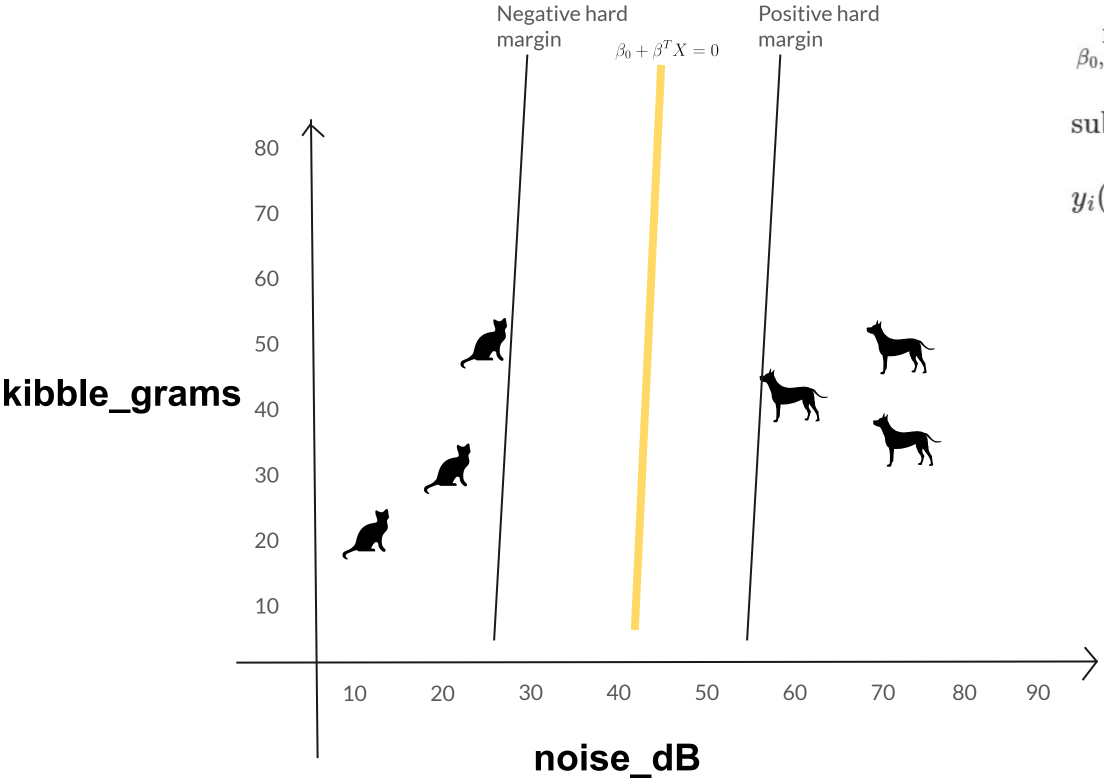
Each hyperplane **has its own margin** (minimum distance to training points).

We want the hyperplane with the largest margin.



Margin: the **minimal distance for each training observation to a separating hyperplane**.

Seems to be the yellow hyperplane. Notice how we draw the **largest possible margin**.



$$\max_{\beta_0,\beta_1,\ldots,\beta_p} \quad M$$

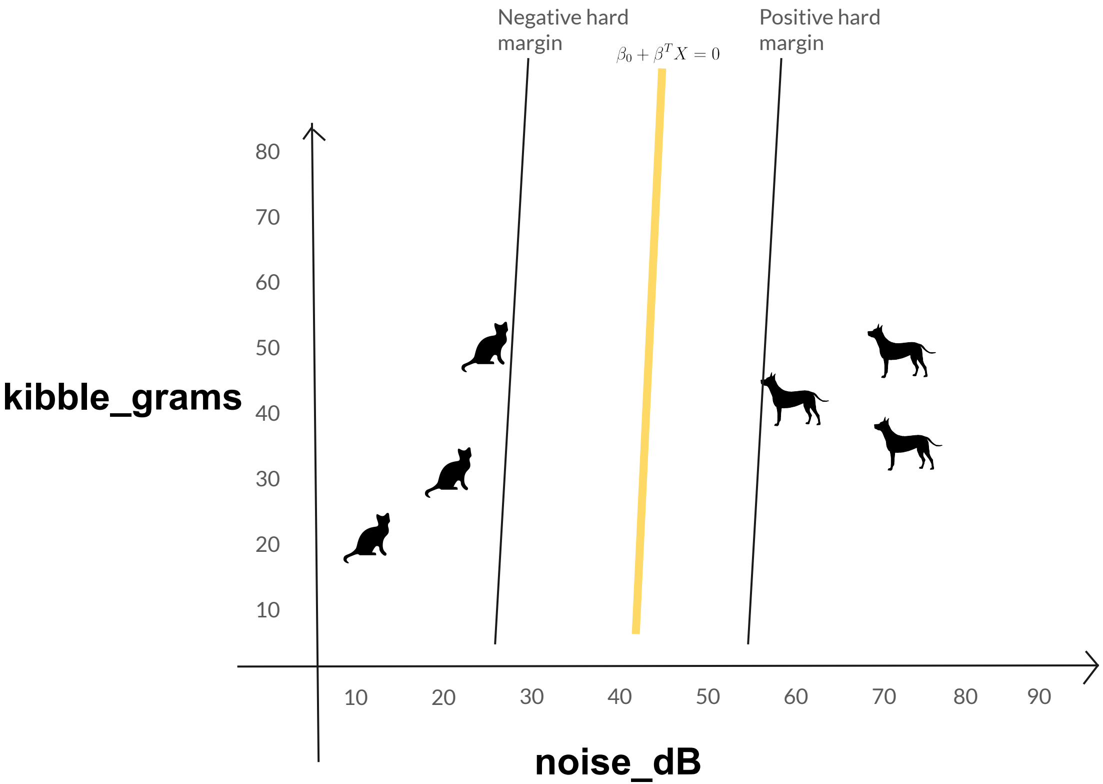
$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq M \quad \text{for all } i = 1,\ldots,n.$$

function = *the margin*
constraints = *all data points are correct*

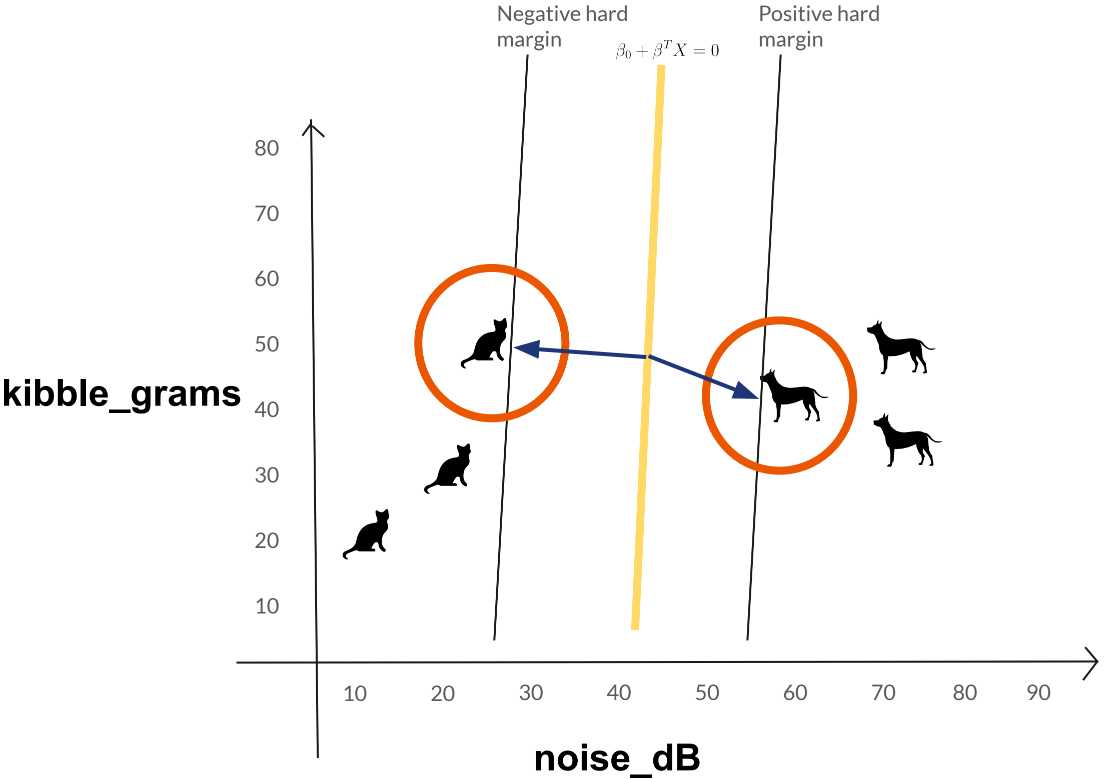
You may be wondering, how are the coefficients for the maximal margin calculated? **Well, prepare yourself.**

We transform this set of constraints into a function that can be **minimized**.



Before we go further into the math, we must also formalize the idea of a **support vector**.

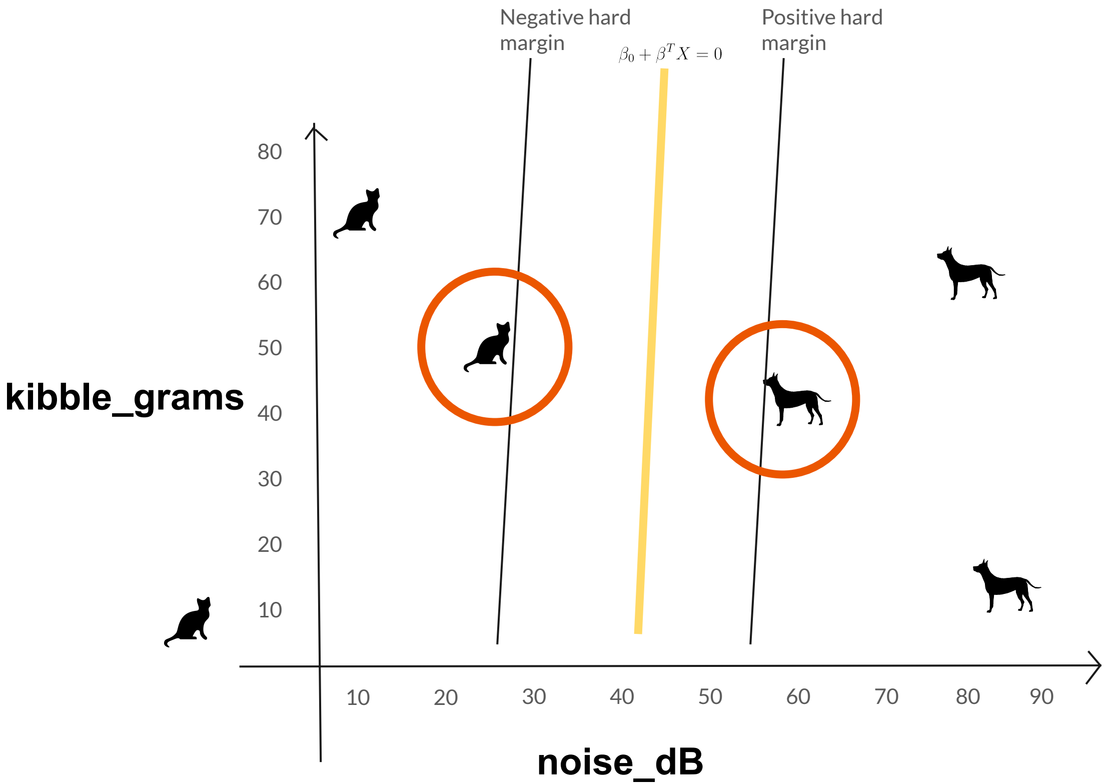
A support vector is a **data point** that just touches the **boundary of our maximal margin**. Can anyone identify our support vectors here?



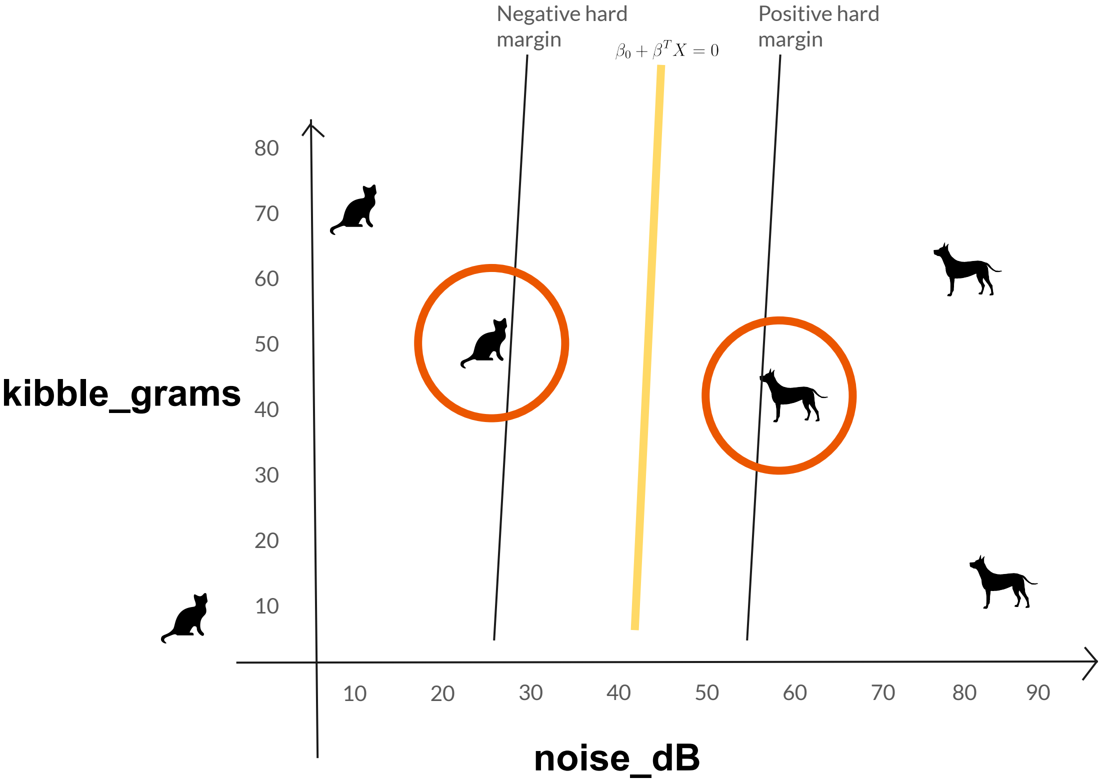
Keep this in mind!

A vector is simply a 1D matrix that expresses magnitude and direction.

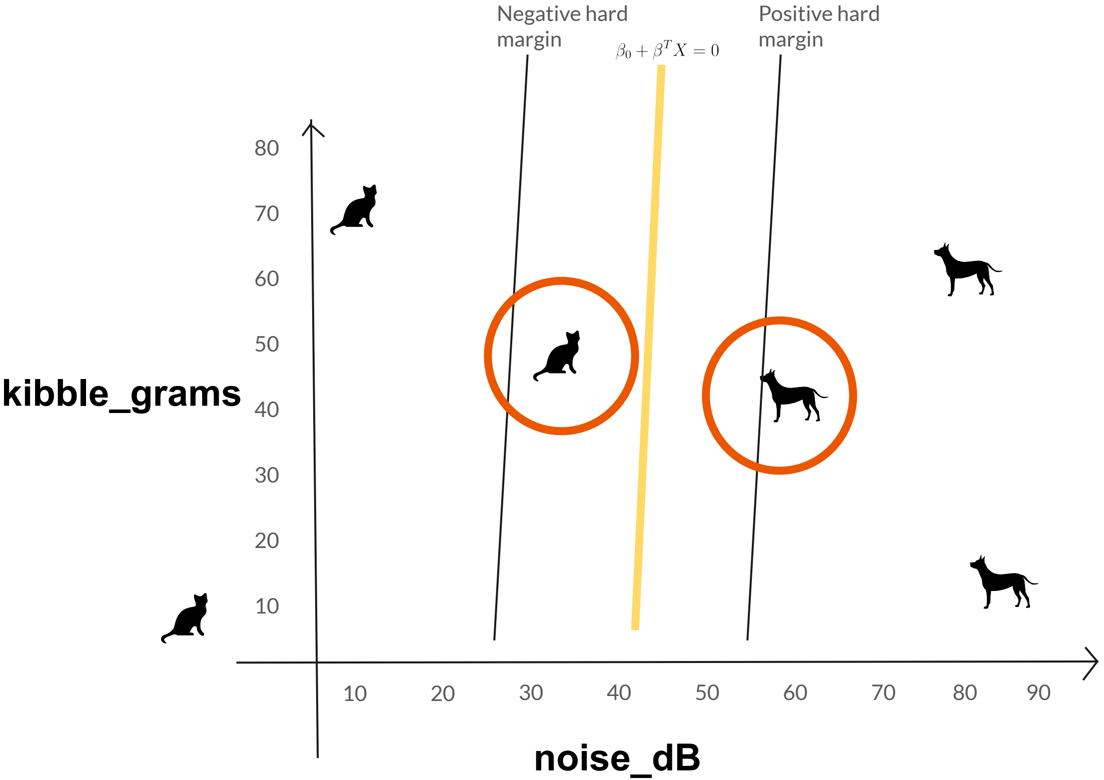
More formally, these are only the “tips” of the support vectors. Vectors are quantities with magnitude and direction. However we’ll omit this fact for now and continue with the understanding that this dog and this cat are our support vectors.



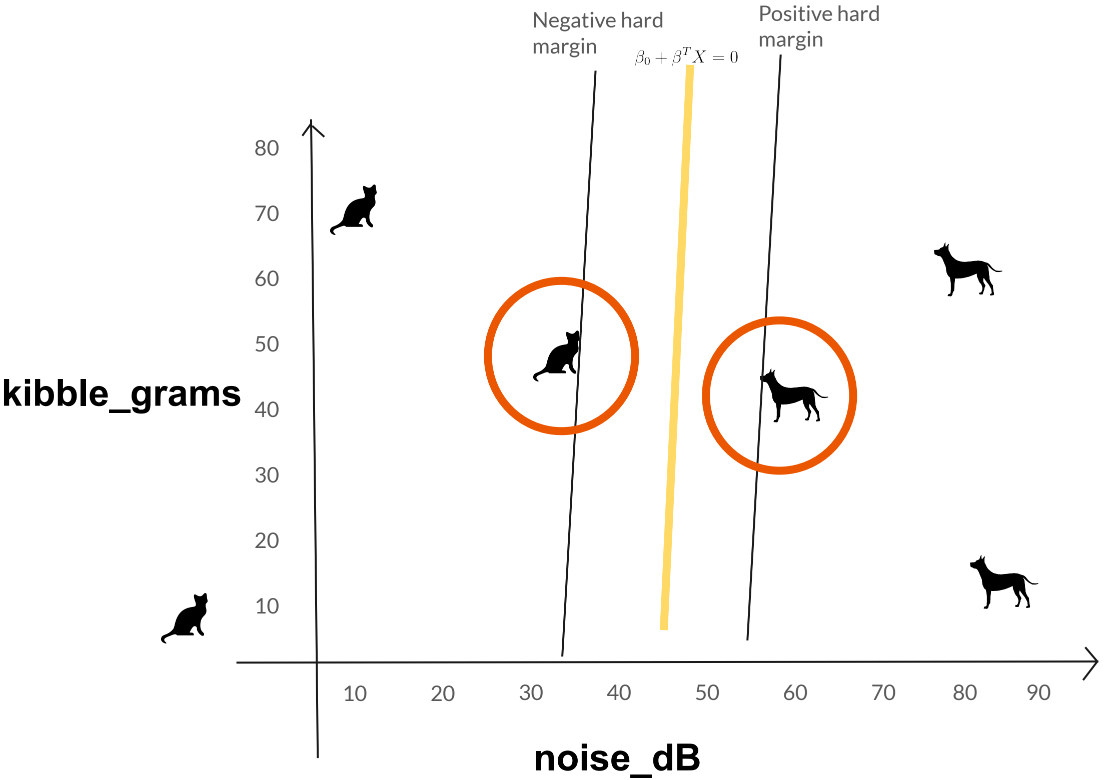
With the way SVM's are trained, only the support vectors truly matter in our calculation of the maximal margin. Consider the updated graph above. Does our decision boundary move?



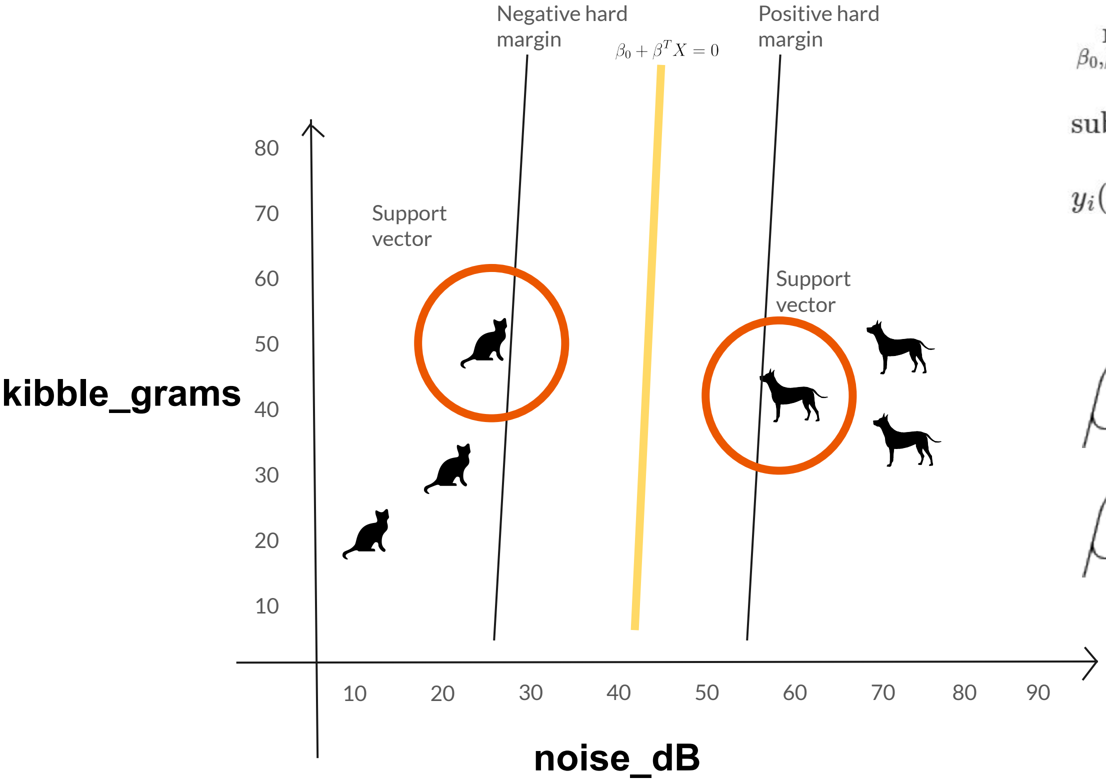
No! As our support vectors stayed in place, the decision boundary remains unchanged.



What if I moved a support vector however?



This updates the possible maximal margin, so **therefore our decision boundary shifts**!



$$\max_{\beta_0,\beta_1,\ldots,\beta_p} M$$

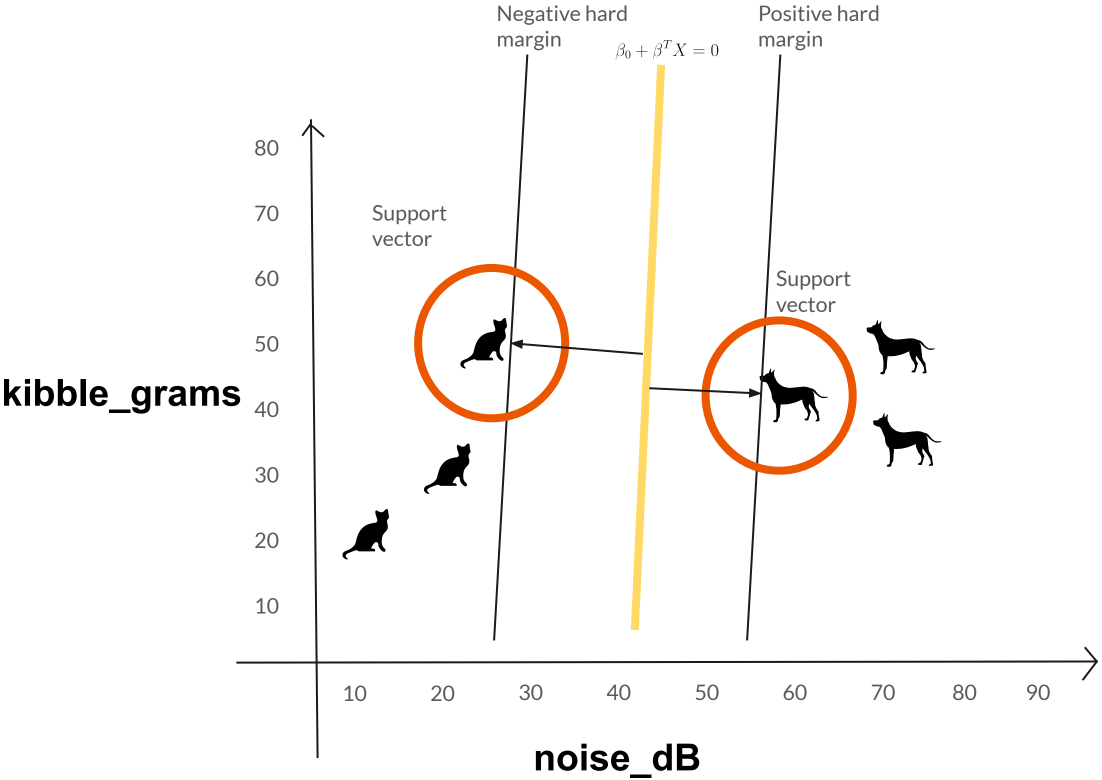
$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq M \quad \text{for all } i = 1,\ldots,n.$$

$$\beta^T X + \beta_0 = 1$$

$$\beta^T X + \beta_0 = -1$$

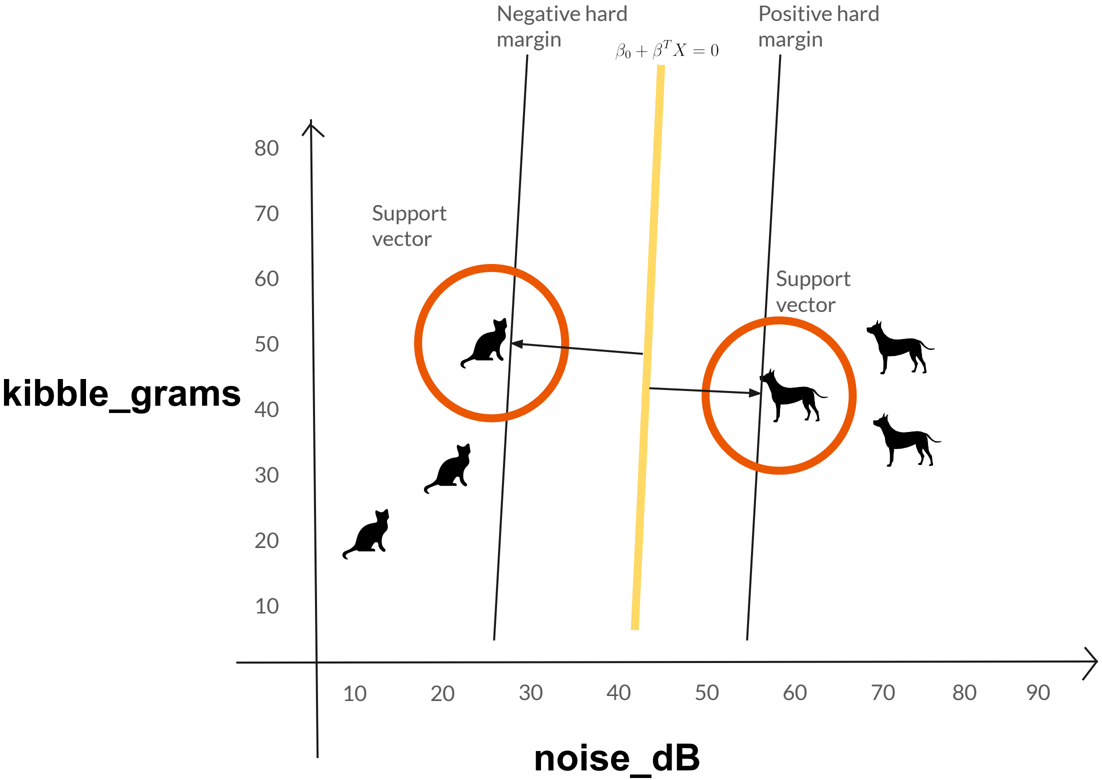
So with these vital points, we can formalize our maximal margin for the two possible kinds of support vectors: both **positive and negative.**



$$\frac{|\beta^T X + \beta_0|}{||\beta^T||}$$

Using something akin to the euclidean distance formula, we can formalize **the distance from a support vector to the hyperplane** with the following formula:

More info here: https://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf

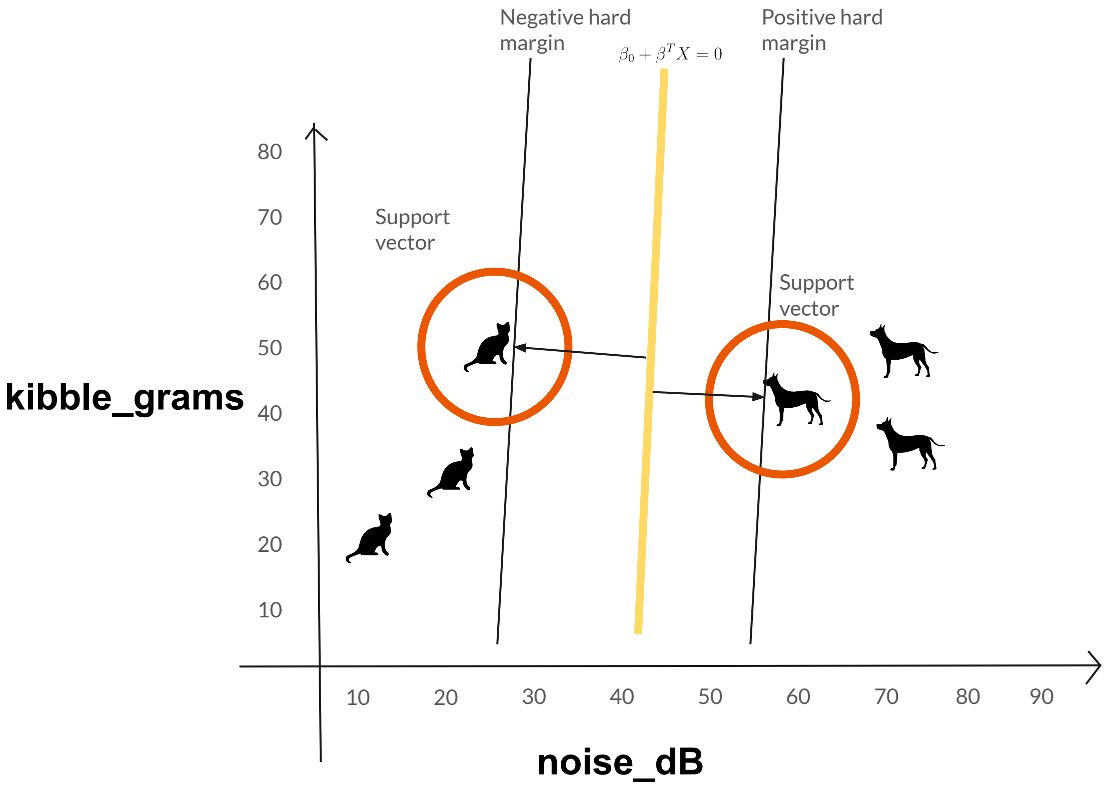


$$\beta^T X + \beta_0 = 1$$

$$\beta^T X + \beta_0 = -1$$

$$\frac{|\beta^T X + \beta_0|}{||\beta^T||}$$

This bars in the numerator means “*absolute value.*” **Given what we know about our support vectors, what will the numerator ALWAYS evaluate to???**



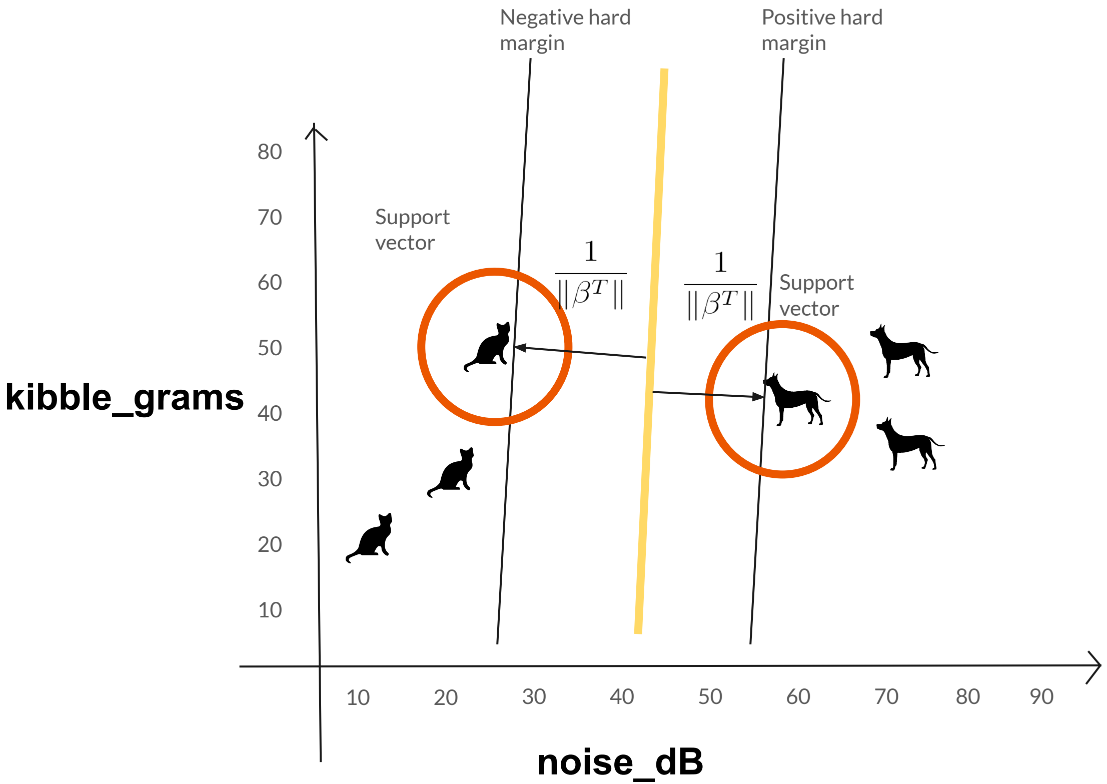
$$\beta^T X + \beta_0 = 1$$

$$\beta^T X + \beta_0 = -1$$

$$\frac{1}{||\beta^T||}$$

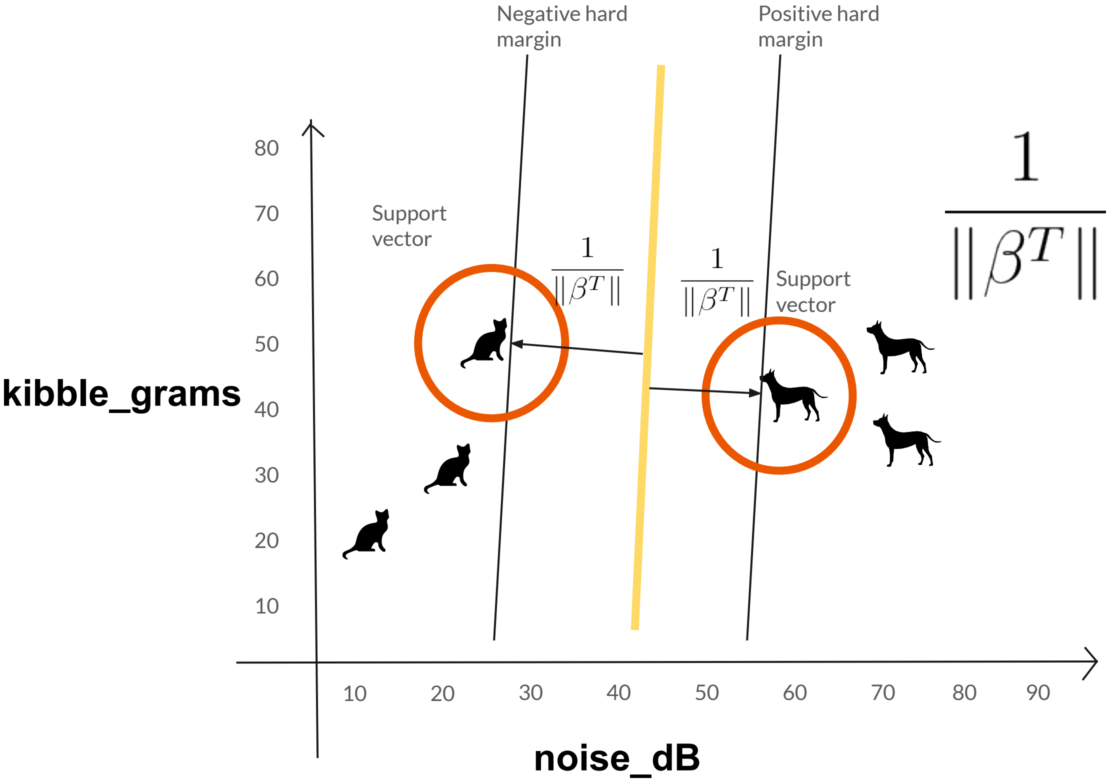
The most satisfying thing in maths is when a complex calculation evaluates to a beautiful number.

Remember, simplicity is the key! **This is the distance between our support vector and the hyperplane.**



$$\frac{1}{||\beta^T||} + \frac{1}{||\beta^T||}$$

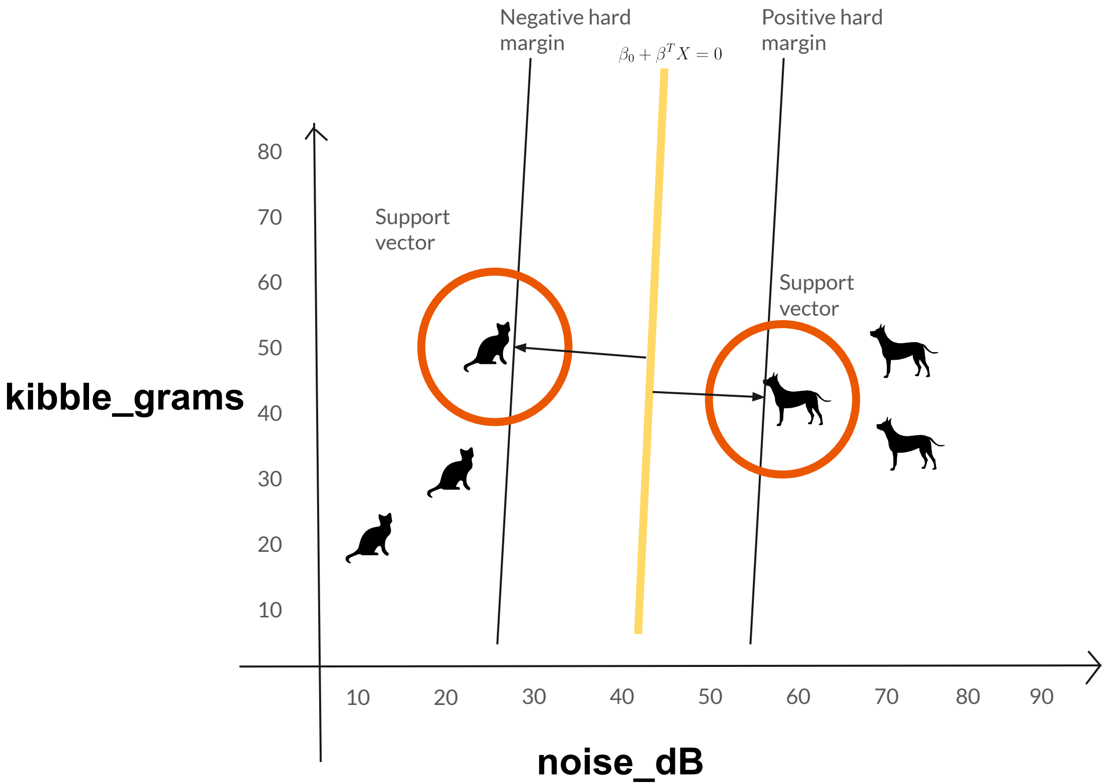
We want to calculate the maximal margin. **Since the distance to a support vector on both the positive and negative side is 1/norm(B^T)** what is the **total width of the margin**?



$$\frac{1}{||\beta^T||} + \frac{1}{||\beta^T||} = \frac{2}{||\beta^T||}$$

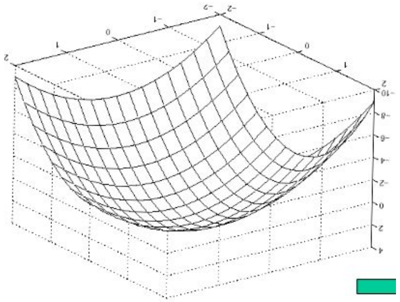
$$||\beta^T|| = \sum \beta_i$$

The denominator is what we call **the "norm" of a vector.** This is a summation of all the coefficients in our vector of coefficients. Since this is division, how can we get the **maximum width**? Do we need our coefficients to be as large, or as small as possible???



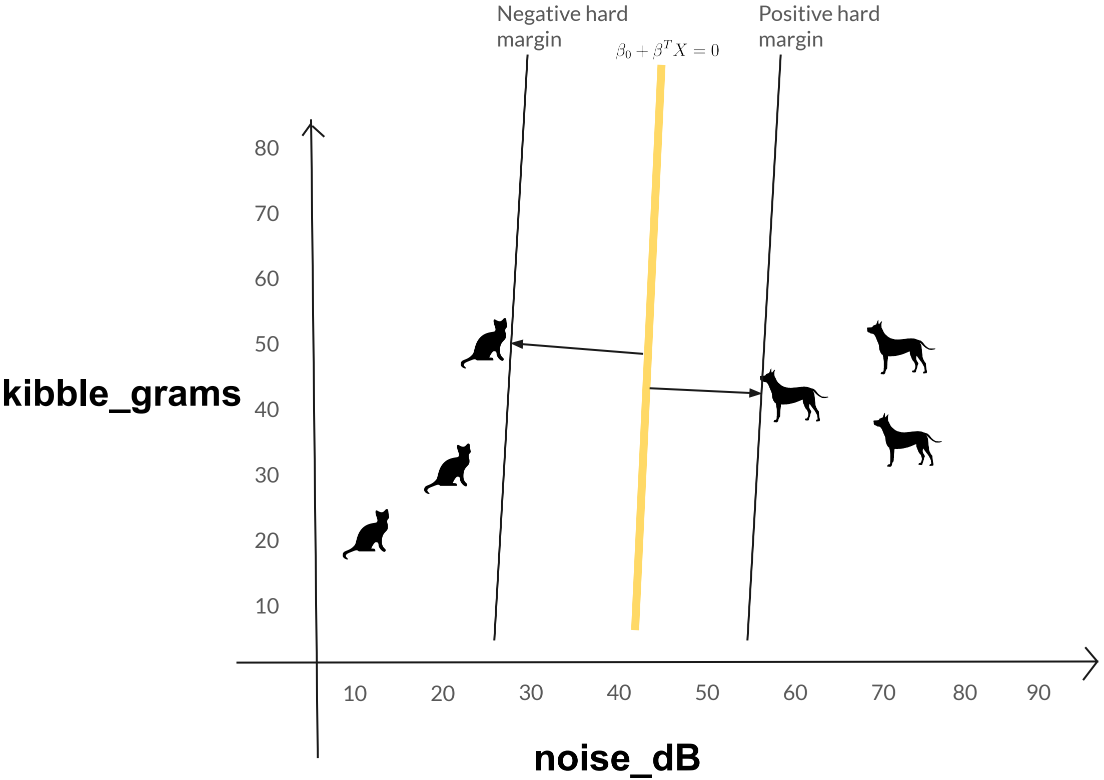
$$\min \frac{1}{2} \|\beta\|^2$$

$$[y_i(\beta * x_i)] = 1$$



This is a minimization problem. We want to find the **smallest weights possible** given the following constraints. **This involves finding a global minimum on a quadratic surface via quadratic programming.**

Let’s stop here in terms of theory. Look at the MIT link for more info.



$$\beta_0 + \beta^T X = 0$$

Upon completion of this optimization problem, we get a hyperplane with coefficients that perfectly separates two classes of data.

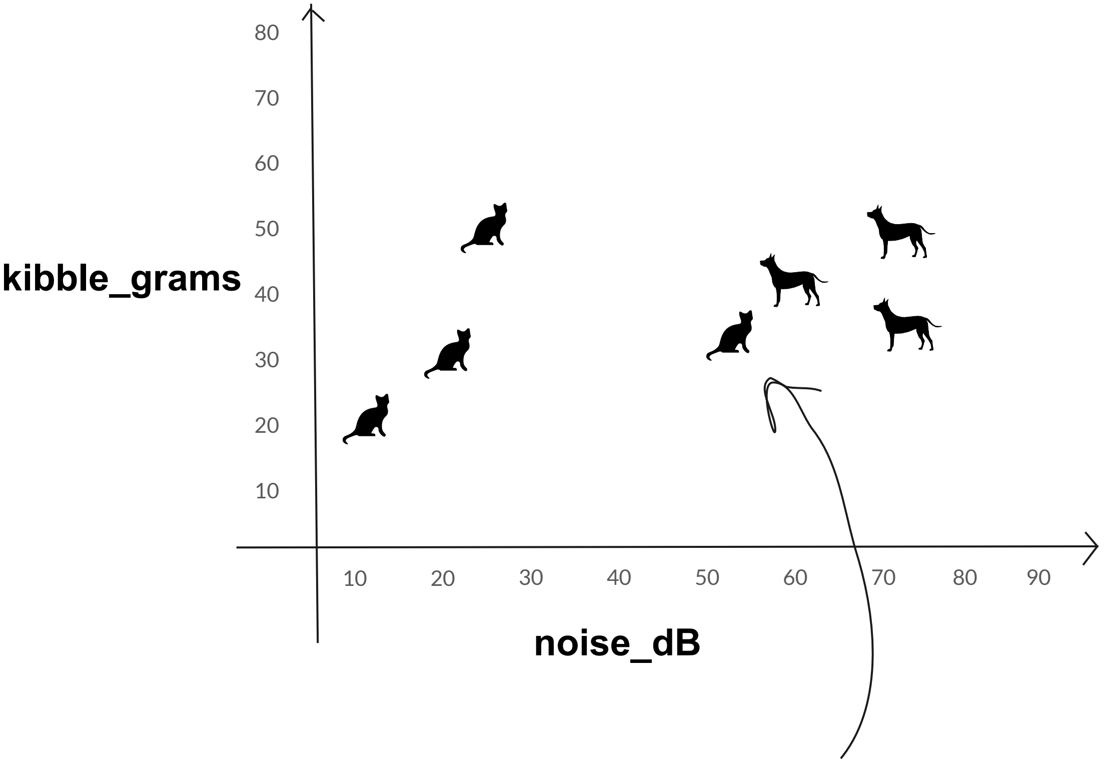
Maximal Margin Classifier

In summary, the maximal margin classifier *calculates*

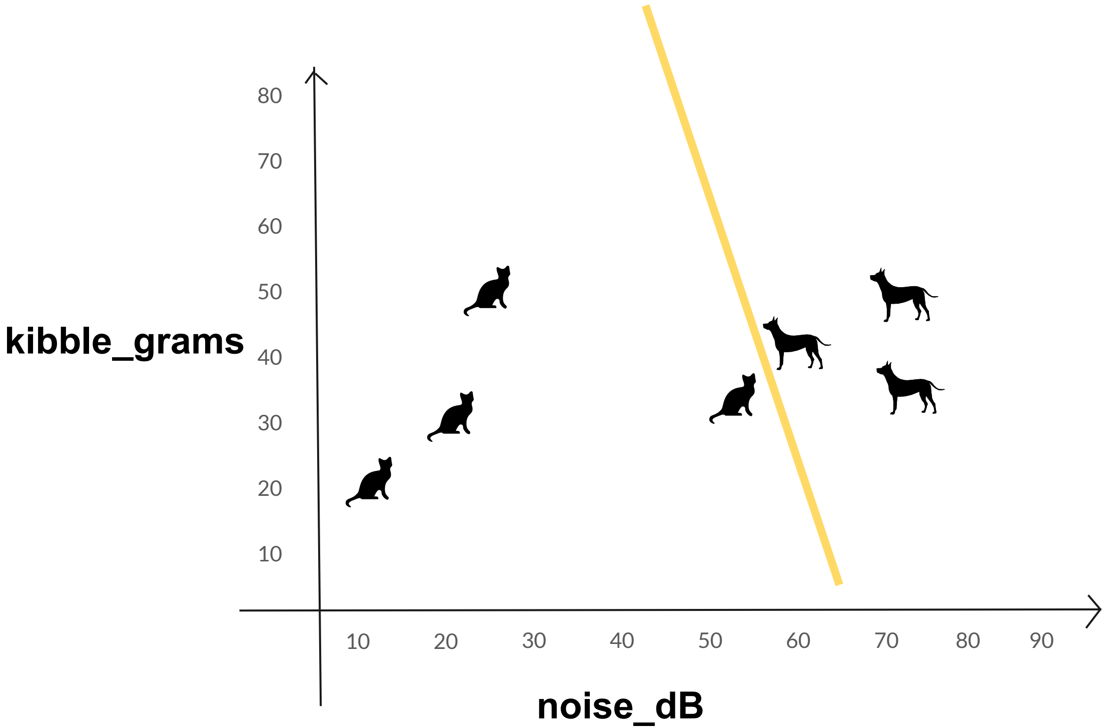
> *Coefficients which result in the **largest possible margins**, aka distance between hyperplane and support vectors.*

This give us something known as the **hard margin!** It cleanly separates our data into two decision regions.

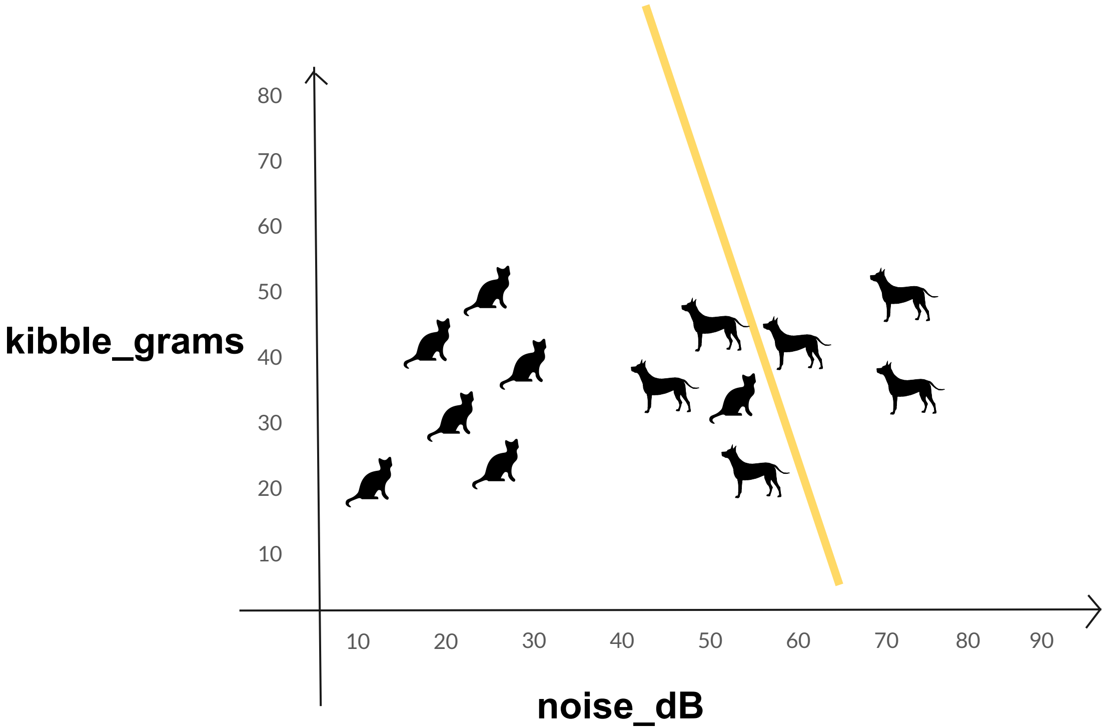
However, this is not ideal. Can anyone see why?



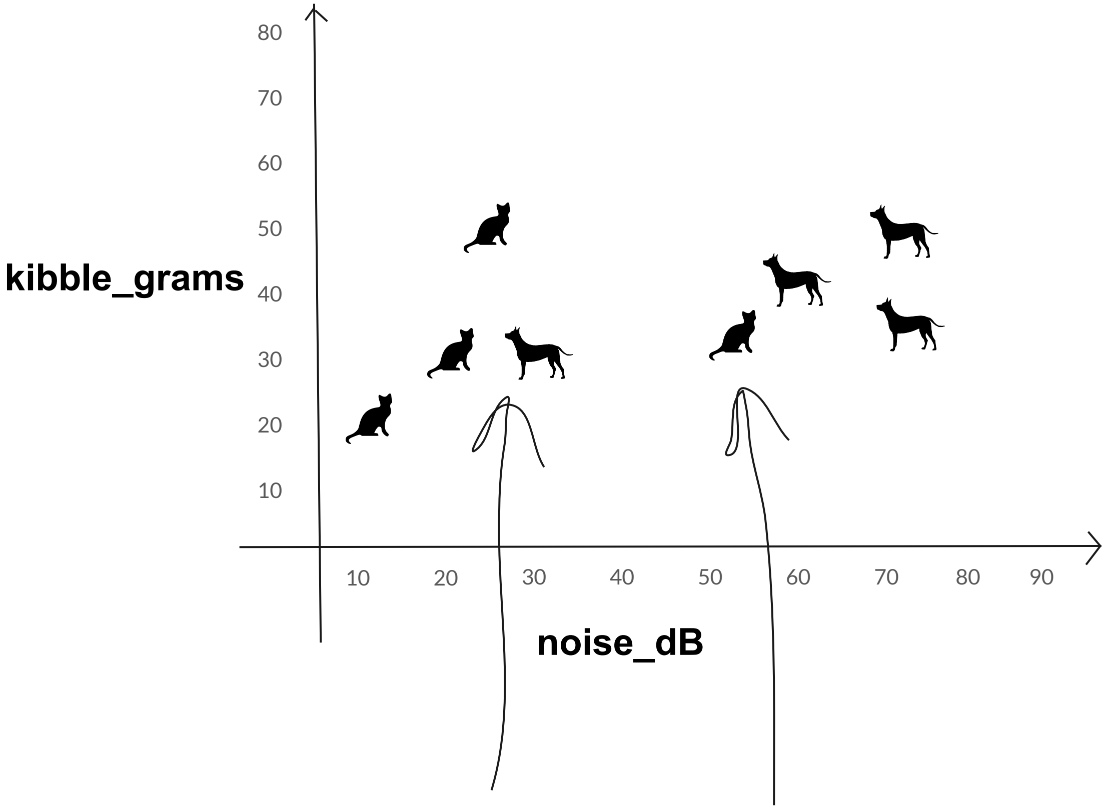
What will be the **hyperplane** if we have the **following training datapoint?**



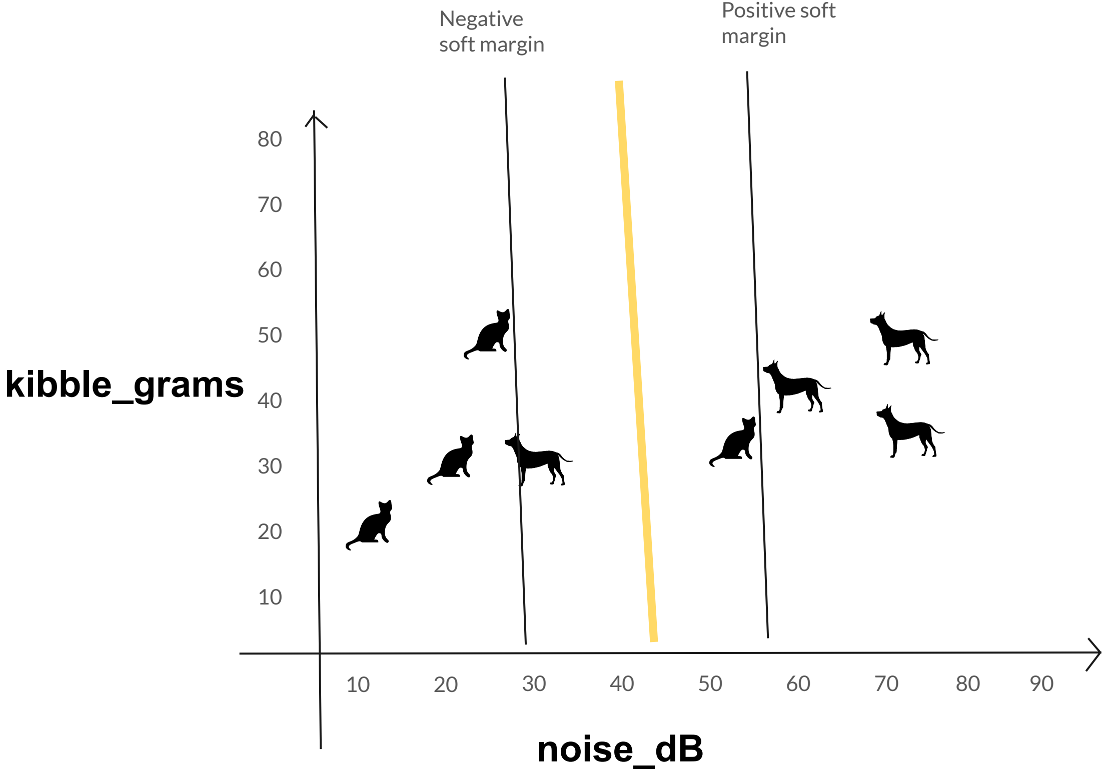
Of course, we evenly separate the data, but what will this model suffer from?



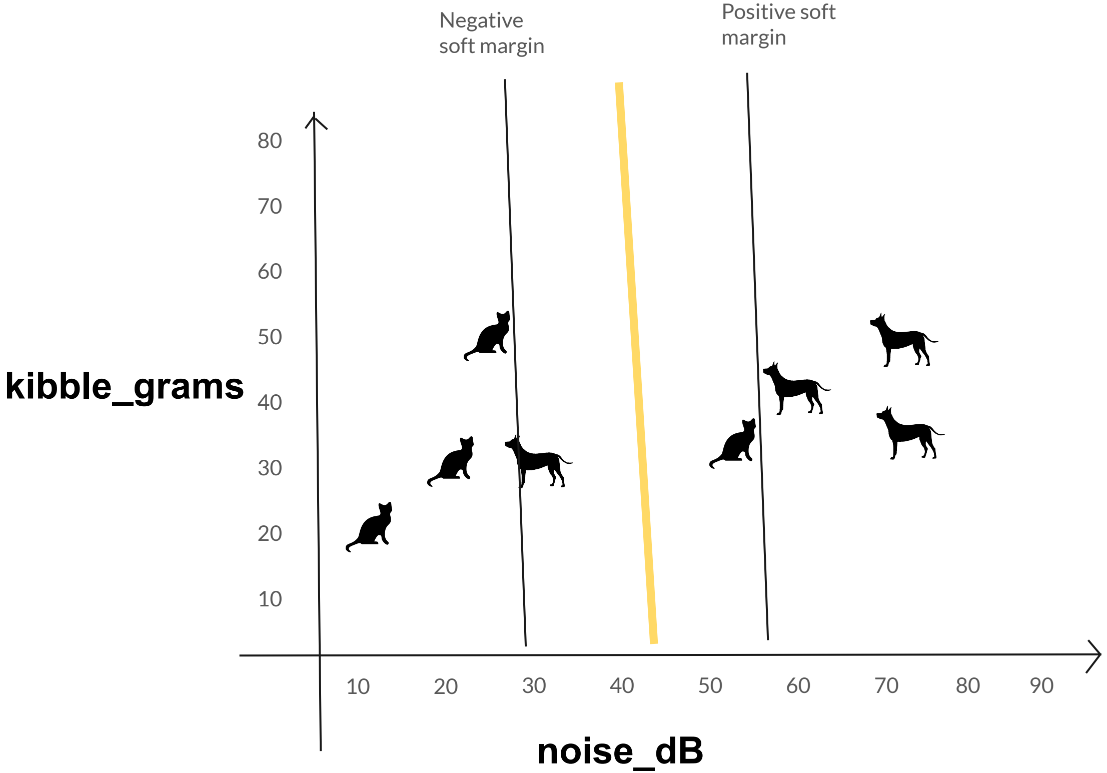
High bias. It will fail to generalize to test samples. Since this cat certainly is loud, it is a poor representation of a *general cat*, so therefore we end up misclassifying many dogs.



Or, alternatively, what if a hard margin is not possible due to *noisy(messy) data*?



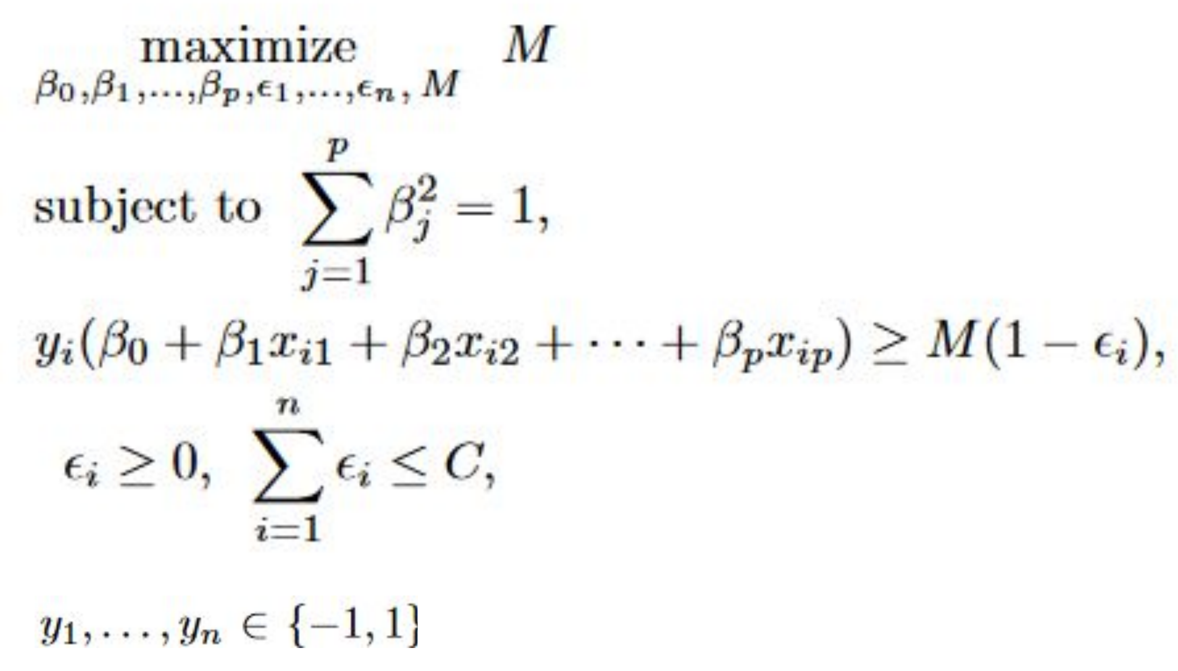
Therefore, we utilize something called a ***soft margin.*** This is also known as a **support vector classifier.** We are willing to misclassify a few examples in other for **returns on variance**.



"...because weakness is a great thing, and strength is nothing."

- *Andrei Tarkovsky*

Remember everyone, **generalization is the name of the game**. Aka good variance-bias tradeoff.



$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n,\,M}{\text{maximize}} \quad M$$

$$\text{subject to} \ \sum_{j=1}^{p} \beta_j^2 = 1,$$

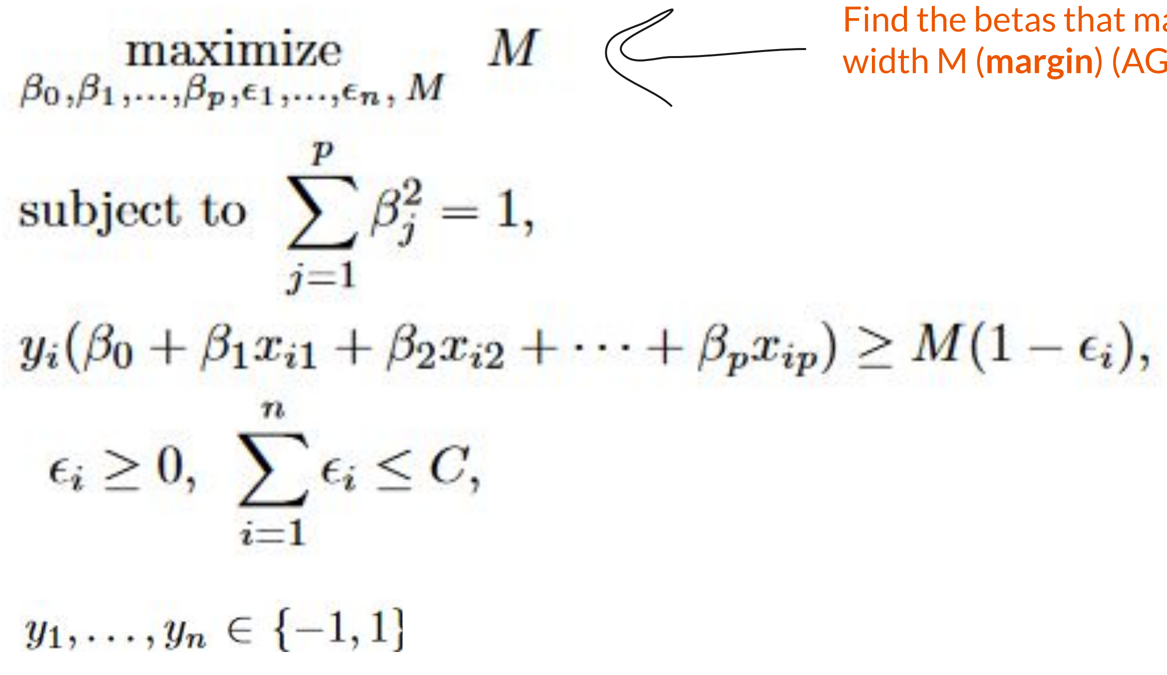
$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \ \sum_{i=1}^{n} \epsilon_i \leq C,$$

$$y_1, \ldots, y_n \in \{-1, 1\}$$

To reiterate, we use a **support vector classifier** when a **separating hyperplane** does not exist (often the case in the real world).

Once again, we have many possible support vector classifiers to choose from. We want to choose the classifier that once again maximizes width *M*, but now, **with the freedom to make mistakes given a constant “C”, which is the sum of i constraint variables ε. Let’s break down this formula as well.**



$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n,\,M}{\text{maximize}} \quad M$$

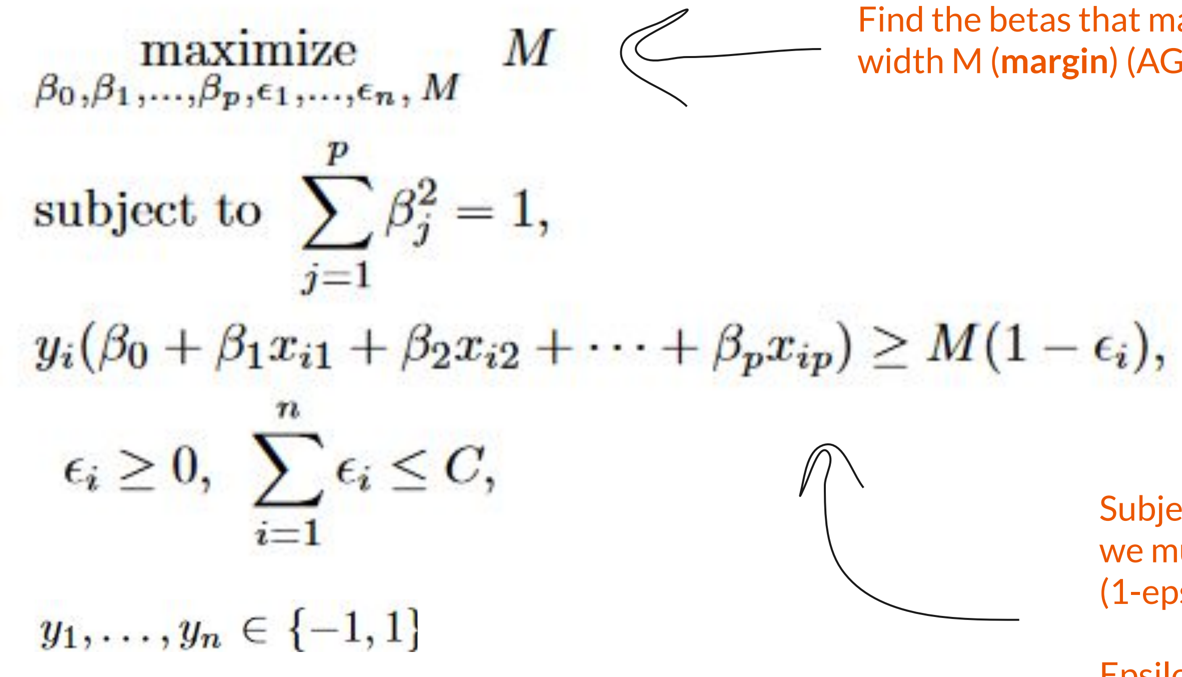
$$\text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C,$$

$$y_1, \ldots, y_n \in \{-1, 1\}$$

Find the betas that maximize the width M (**margin**) (AGAIN!)



$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n,\,M}{\text{maximize}} \quad M$$

$$\text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M(1-\epsilon_i),$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C,$$

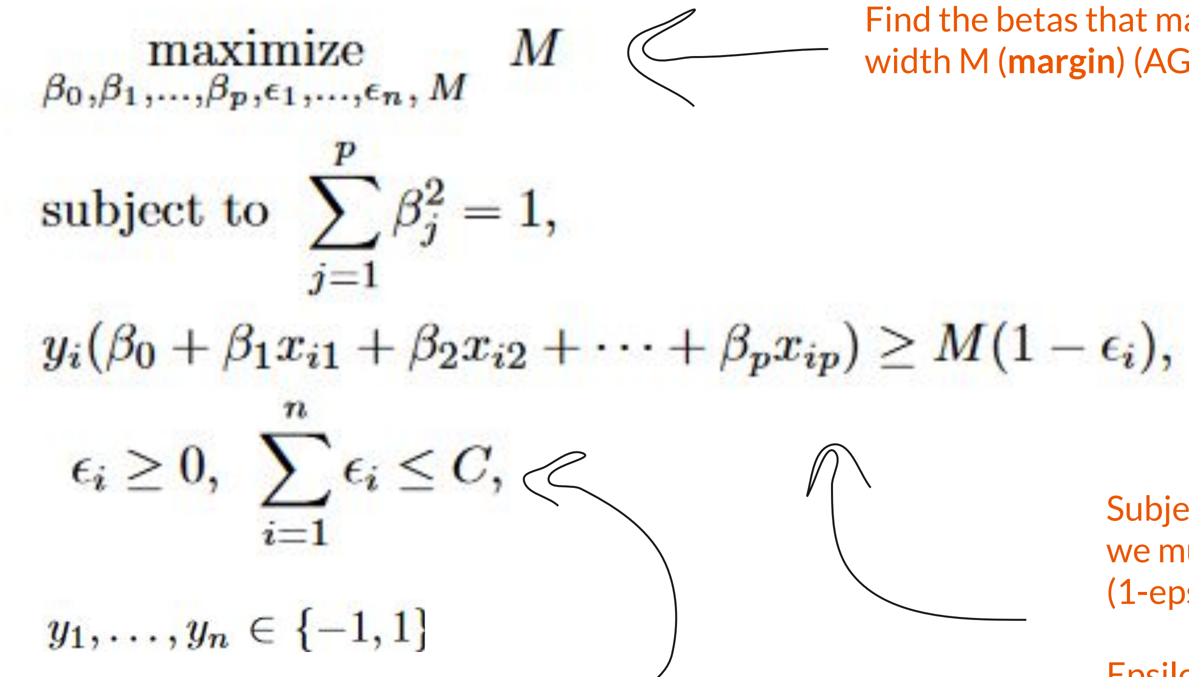
$$y_1, \ldots, y_n \in \{-1, 1\}$$

Find the betas that maximize the width M (**margin**) (AGAIN!)

Subject to a degree of error, where we multiply the width M by (1-epsilon)

Epsilon is known as the *slack variable*

I.e. "cut me some slack" (don't judge my character too harshly)



$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n,\,M}{\text{maximize}} \quad M$$

$$\text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C,$$

$$y_1, \ldots, y_n \in \{-1, 1\}$$

Find the betas that maximize the width M (**margin**) (AGAIN!)

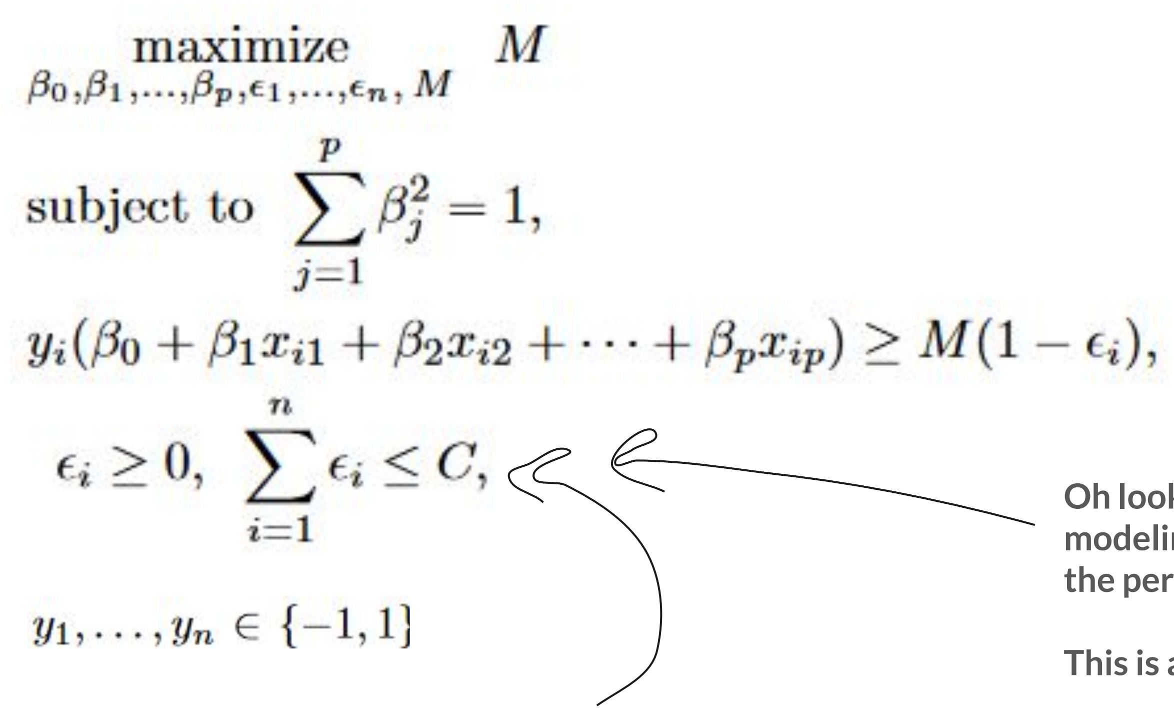
Subject to a degree of error, where we multiply the width M by (1-epsilon)

Epsilon is known as the *slack variable*

I.e. "cut me some slack" (don't judge my character too harshly)

Constrained by a constant "C".

Think of "C" as your budget for slack. The higher the "C", the more tolerance you have for error.



$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n,\,M}{\text{maximize}} \quad M$$

$$\text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

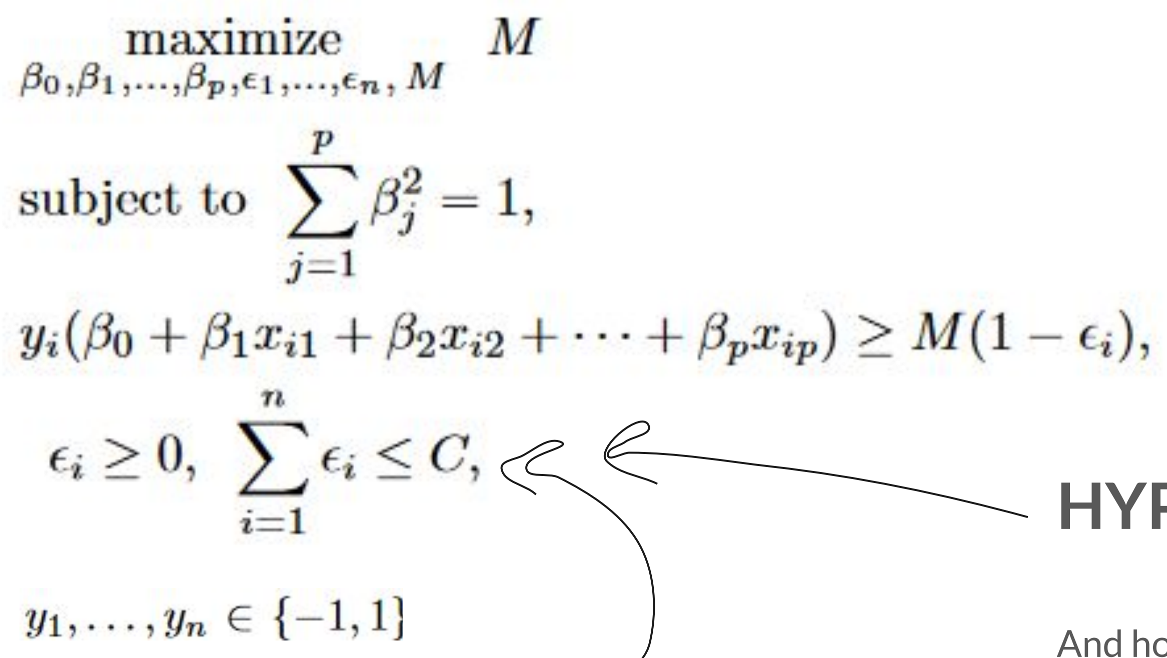
$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C,$$

$$y_1, \ldots, y_n \in \{-1, 1\}$$

Oh look, a variable that is not involved in modeling of the data, but yet influences the performance of my model.

This is a....

Constrained by a constant “C”



$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n,\,M}{\text{maximize}} \quad M$$

$$\text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

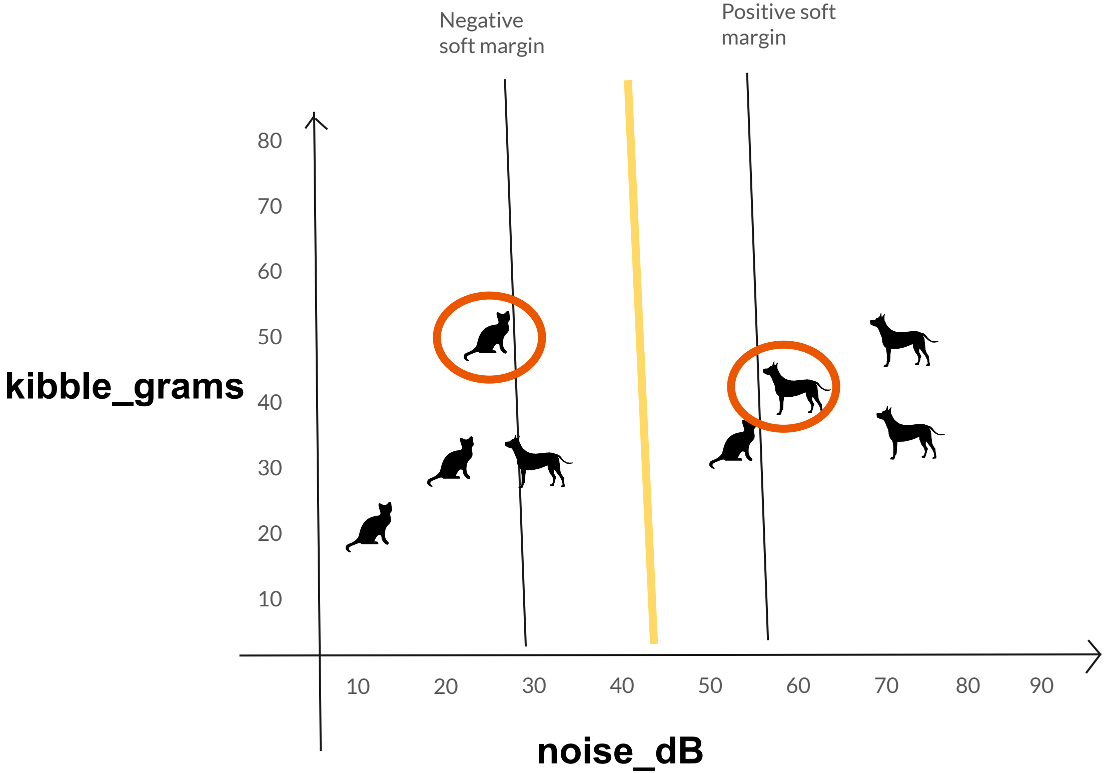
$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C,$$

$$y_1, \ldots, y_n \in \{-1, 1\}$$

HYPERPARAMETER!

And how do we find the best hyperparameter???

Constrained by a constant “C”



So, after finding the best possible C, we refit our hyperparameter with the added tolerance for error. This gives us the best possible **support vector classifier**. Notice that our support vectors from before are still sitting on the margins!

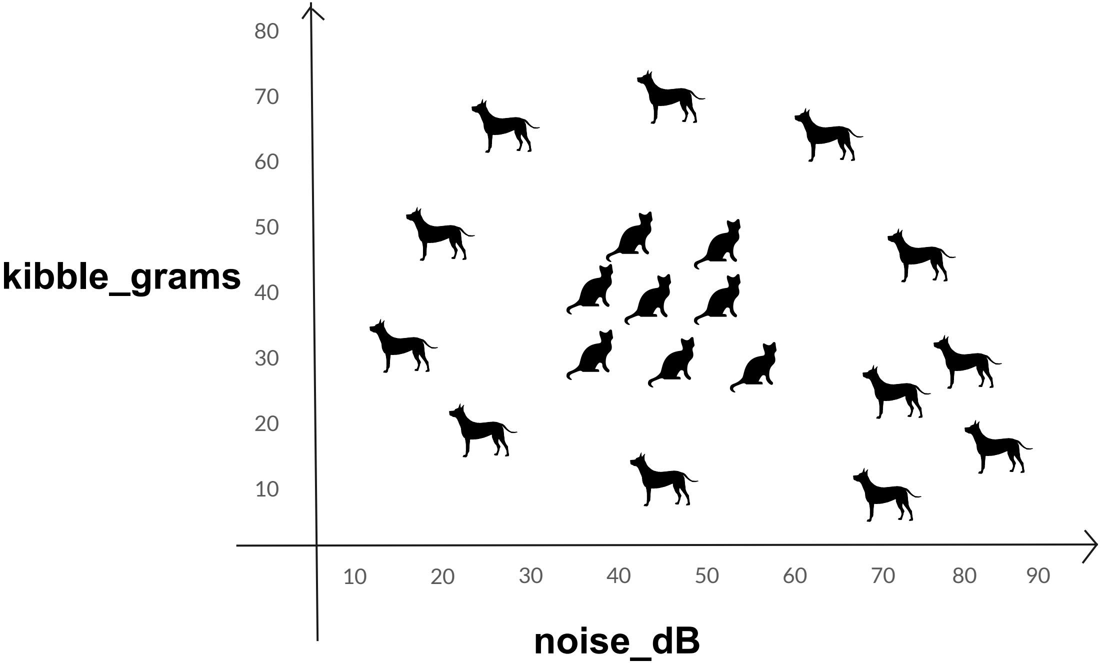
Support Vector Classifier

In summary, the support vector classifier:

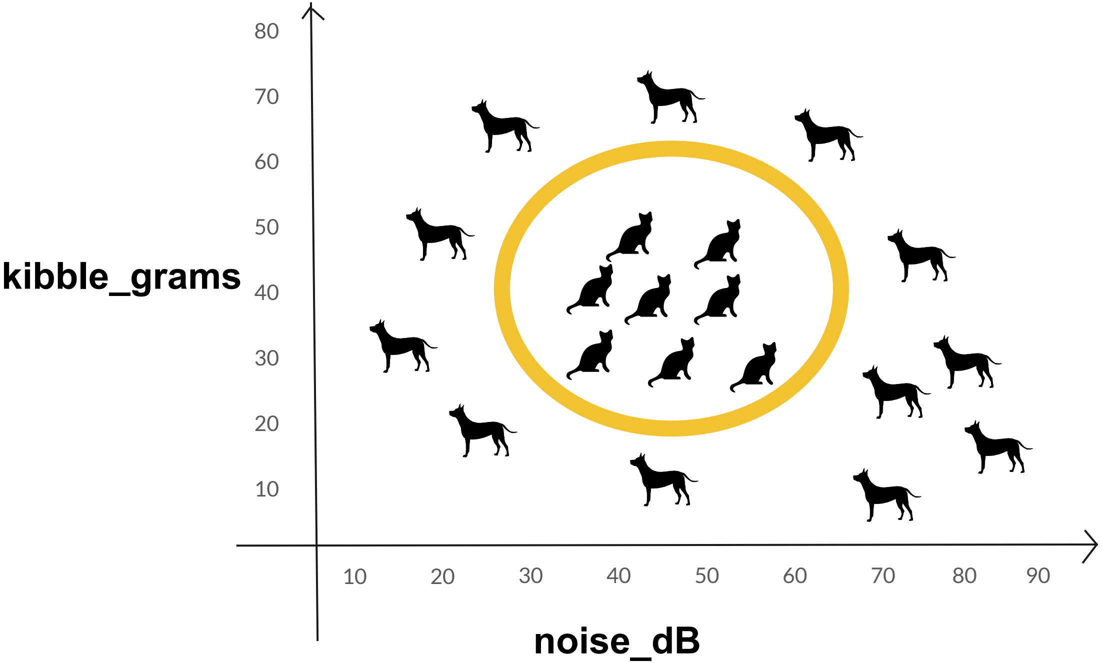
> *Coefficients which result in the **largest possible margins**, aka distance between hyperplane and support vectors **with tolerance for misclassifications based on some hyperparam C**.*

This give us something known as the **soft margin!** It separates the two classes of data to the best of its ability.

However, this (again) **might** not be ideal. Can anyone see why?



What if we have a non-linear decision boundary. Even with room for error (slack), you will be creating **inaccurate models.**

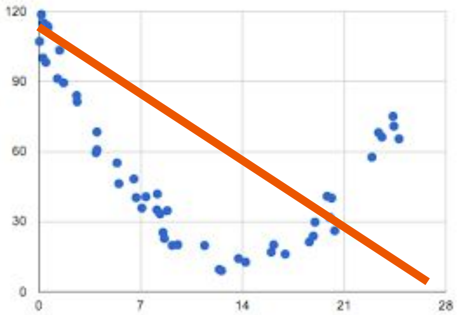


We need a non-linear decision boundary.

Support Vector Machines & Kernel Trick

Non-Linearity

Thinking back to **linear regression**, how did we model a nonlinear relationship between predictor and target? *(if you don't recall, just think about this question generally, how do we model nonlinear relationships?)*

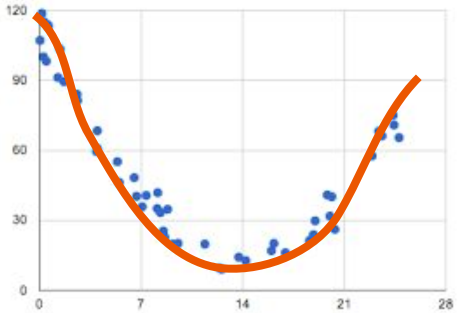


$$y = \beta_1 x + \beta_0$$

**You are working with 3 dimensions now (X^2, X, Y) not just 2 (X, Y)!*

Non-Linearity

We introduced a quadratic (or another other polynomial term) to model this relationship. This gave us a non-linear feature to work with (*aka* ***increases the dimensional space*** *of our predictors*)

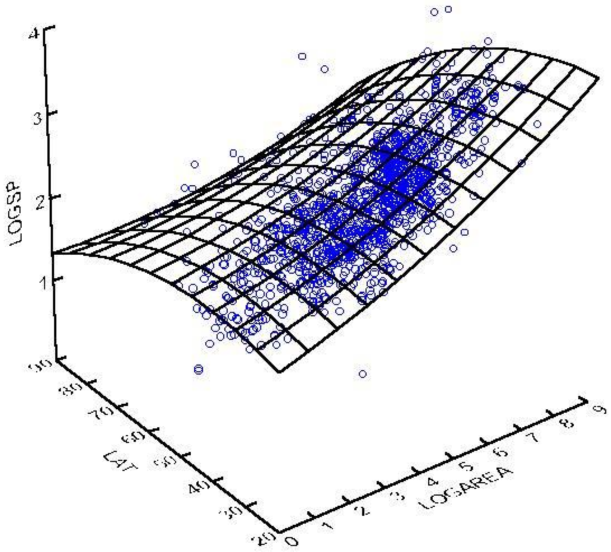


$$y = \beta_2 x^2 + \beta_1 x + \beta_0$$

Non-Linearity

In less abstract terms, this added dimension (*X^2 or X^3 or log(X)*) actually projects our simple 2D scatter plot into **3 dimensions**.

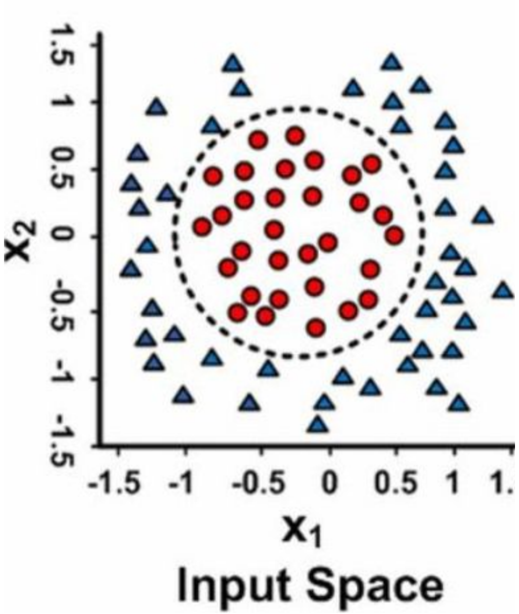
This projection allowed us to create a better model which captures the nuances of the complex shape.



Non-Linearity - SVM

Let’s see if we can extend this intuition to SVM’s.

If the decision boundary exhibits non-linear properties, **what could we modify to capture this complex shape? (i.e. what does the modeling???)**

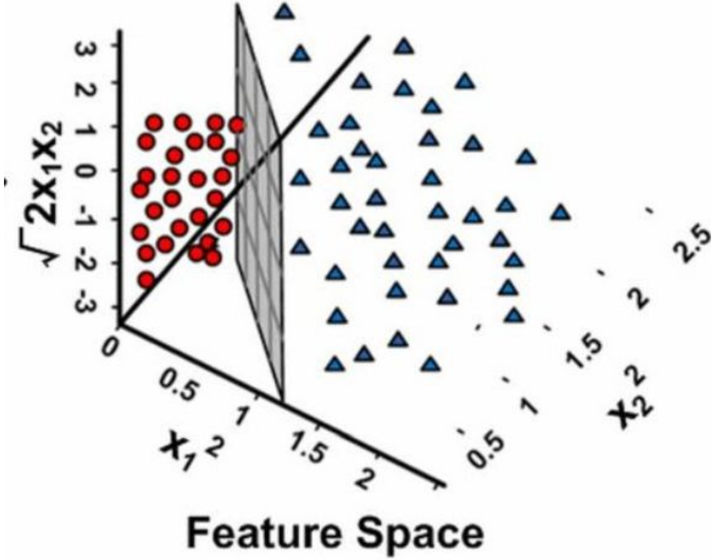


$$\beta_2 X_2 + \beta_1 X_1 + \beta_0 = 0$$

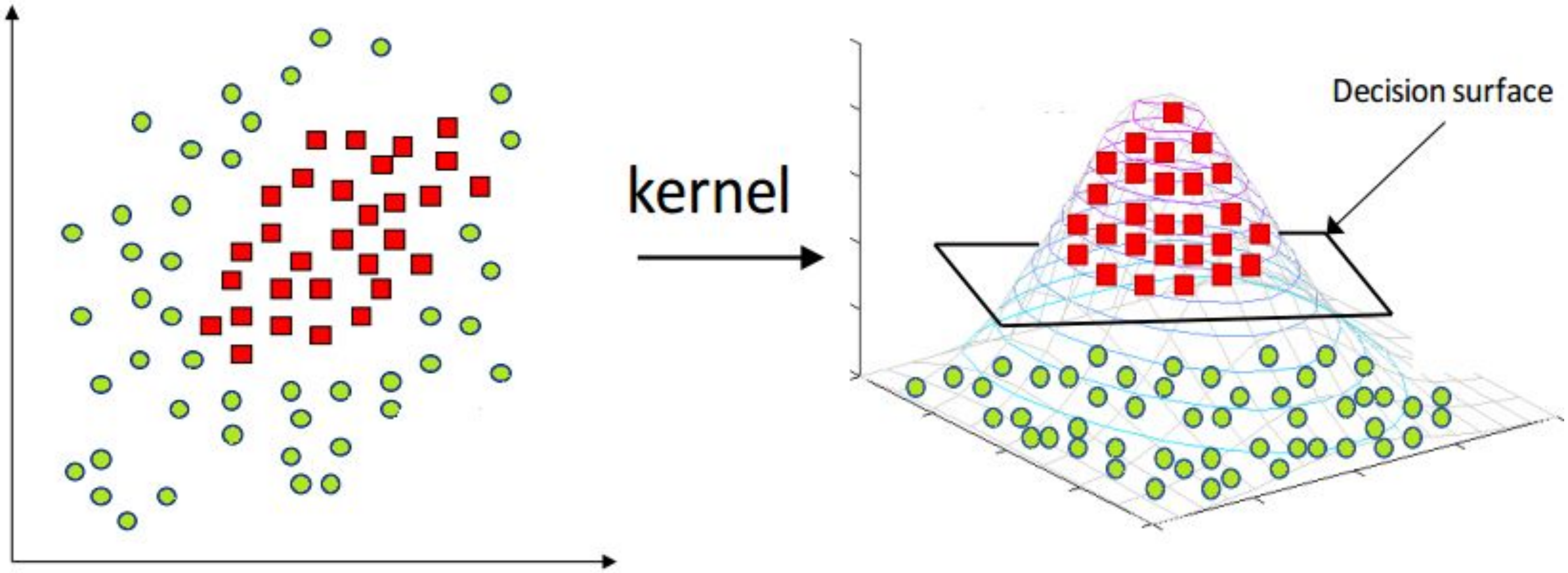
Non-Linearity - SVM

We add an additional dimension to **model our data in higher dimensional space.**

This allows us to still find a linear hyperplane that separates our data! **However now, a linear hyperplane in the additional dimension *manifests* itself as a nonlinear boundary!**



$$\beta_3\sqrt{X1 * X2} + \beta_2 X_2 + \beta_1 X_1 + \beta_0 = 0$$



Added dimensionality to a two-dimensional dataset allows us to train a flat hyperplane that exists in 3D space, which will provide a nonlinear decision boundary.

Support Vector Machines (SVM)

This finally allows us to train a *support vector machine*. This describes a binary supervised learning classifier which can model non-linear decision boundaries.

We are extracting the true signal from the noise of data.

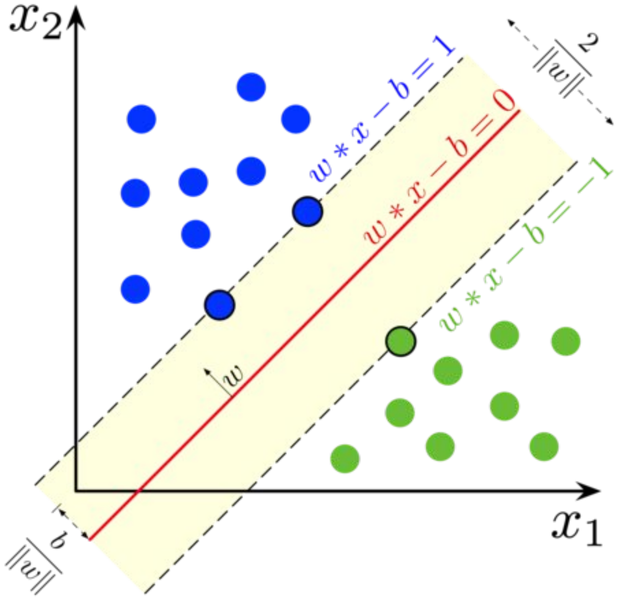
However, there are still a few hurdles to get through:

- *Which higher order function do we use? Quadratic ? Cubic? Something else entirely?*
- *How do we deal with the computational complexity of such an operation! We're adding more data!*

Support Vector Machines (SVM)

As was alluded to in the training of our maximal margin classifier, we mostly care about the **dot product of our values** as opposed to the values themselves.

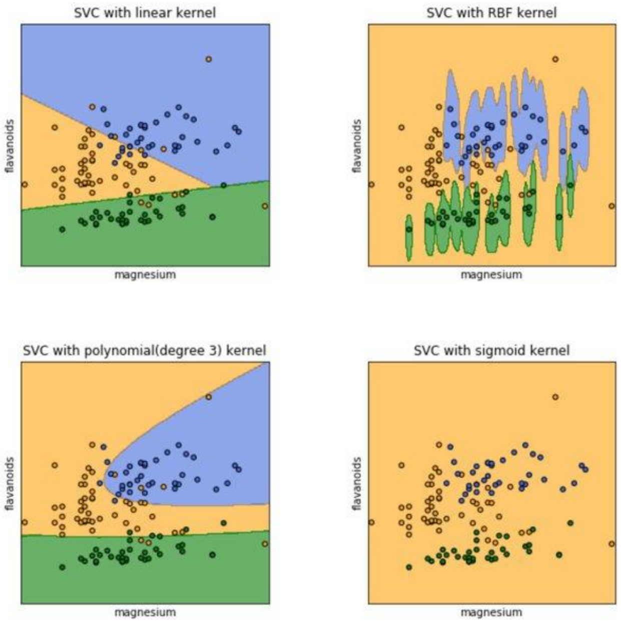
Using this idea, we establish a **general** form of the support vector classifier via the following formula:



$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle,$$

Kernel Trick

However if we're interested in higher order dimensions, we need to generalize this relationship into a component that can capture the **relationship between multi-dimensional data points. (i.e. model non-linear boundaries) without actually mapping these values to higher dimensions!**



SVC with linear kernel

SVC with RBF kernel

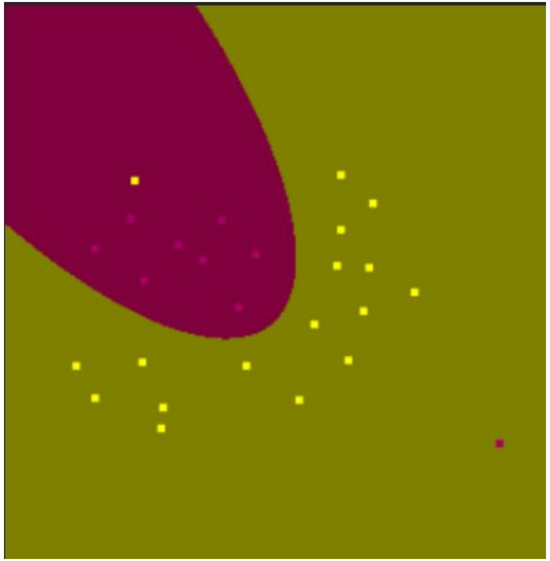
SVC with polynomial(degree 3) kernel

SVC with sigmoid kernel

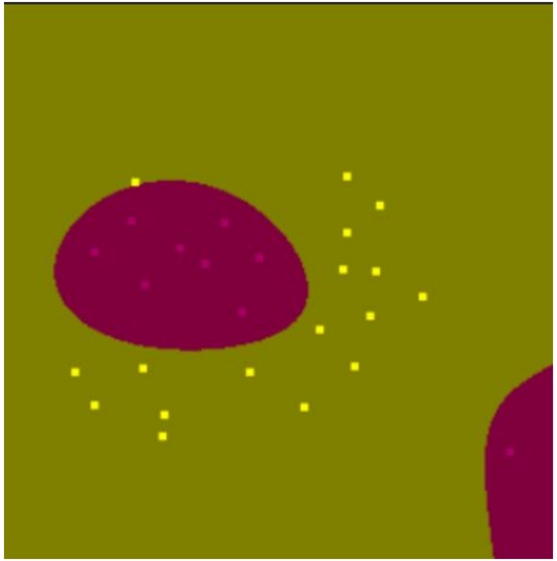
This is known as a *kernel function.*

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i K(x, x_i).$$

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j},$$



Polynomial Kernel



Radial Kernel

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij}x_{i'j})^d.$$

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2).$$

To model different kinds of decision boundaries, we use different kernels. All of these entail hyperparameters (ex: “d” in polynomial). **How do we select the best hyperparam?**

How to Choose A Kernel?

Start with the simplest kernel (linear) and if (and only if) it underfits (poor training, poor testing accuracy), switch to a more complex kernel.

Fig.3

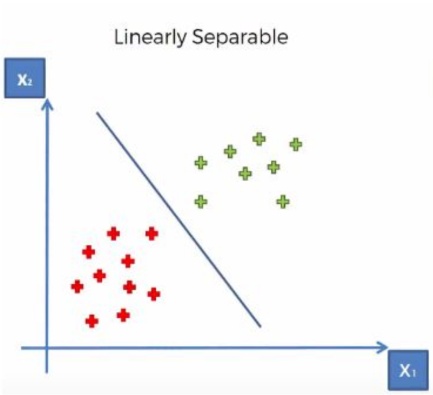
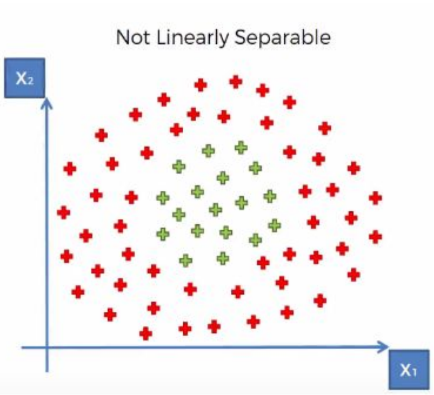


Fig.4



End of Class Announcements

Lab (Due 7/23)

Zurich, Switzerland

You are a data scientist working for a Zurich-based international bank called Caishen. The company announced in an all-hands meeting that they are aiming to develop a classification algorithm **that can identify 99% of all fraudulent activity within customer-facing bank accounts.**

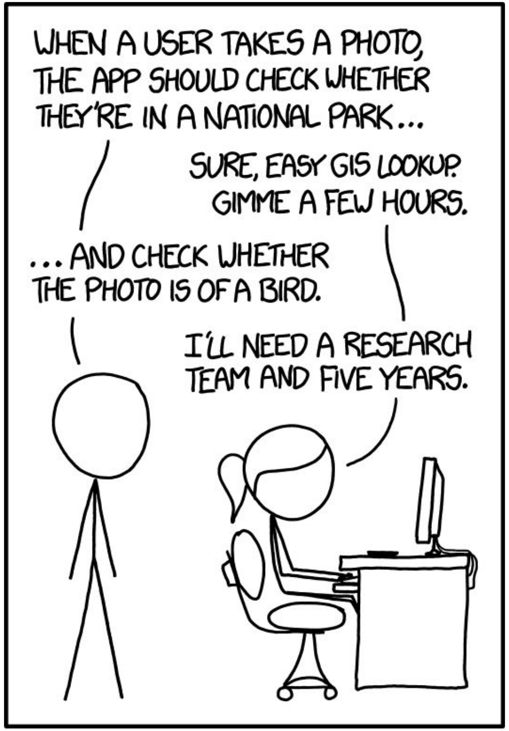
For this project, you will use a **dataset of 1 million bank transactions to create a classifier** that will detect if fraudulent activity has occurred for a transaction.

Your second checkpoint will be to complete your EDA (of template files) by **7/16.**

Tuesday

On Tuesday we will discuss decision trees

- What is **entropy** in information science?
- How do we “grow” a **tree** using entropy?
- How do we use these trees to **predict** samples.


WHEN A USER TAKES A PHOTO, THE APP SHOULD CHECK WHETHER THEY'RE IN A NATIONAL PARK...
SURE, EASY GIS LOOKUP. GIMME A FEW HOURS.
...AND CHECK WHETHER THE PHOTO IS OF A BIRD.
I'LL NEED A RESEARCH TEAM AND FIVE YEARS.

IN CS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALLY IMPOSSIBLE.