Introduction to K-Nearest-Neighbors

THE KNOWLEDGE HOUSE

Agenda - Schedule

1. Warm Up
2. The kNN Classifier
3. Hypertuning a kNN Classifier
4. Break
5. kNN Lab

Guess if this is AI or human generated (note: this was for the 2023 fellows and it was a lot more impactful back then)

Agenda - Goals

-

Warm Up

k-Nearest Neighbors

k-Nearest Neighbors (kNN)

While the bayes classifier is a nice idea, it makes a major assumption that we cannot always fulfill: We know the **probability distribution** of a predictor.

This problem will often arise in our exploration of machine learning.

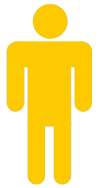
How do I make predictions when I don't know some vital metric?

k-Nearest Neighbors (kNN)

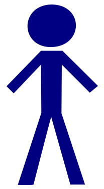
The k-nearest-neighbor **provides an intuitive solution** to estimating our co**nditional probabilities**.

kNN is a **supervised learning multiclass classifier**. Much like bayes classifier, we do not use a formula such as “y=mx+b” to model our data.

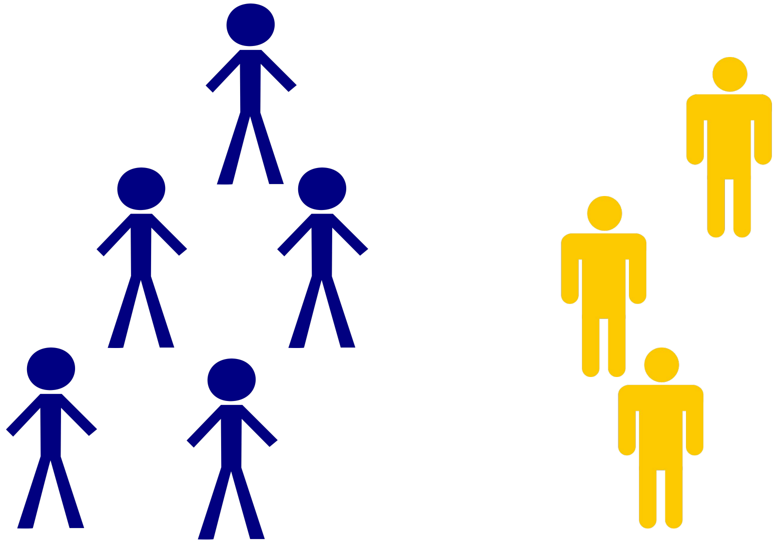
The concept of kNN is an easy idea to internalize. Let’s use a real world example. You are at an industry event full of TKH fellows…



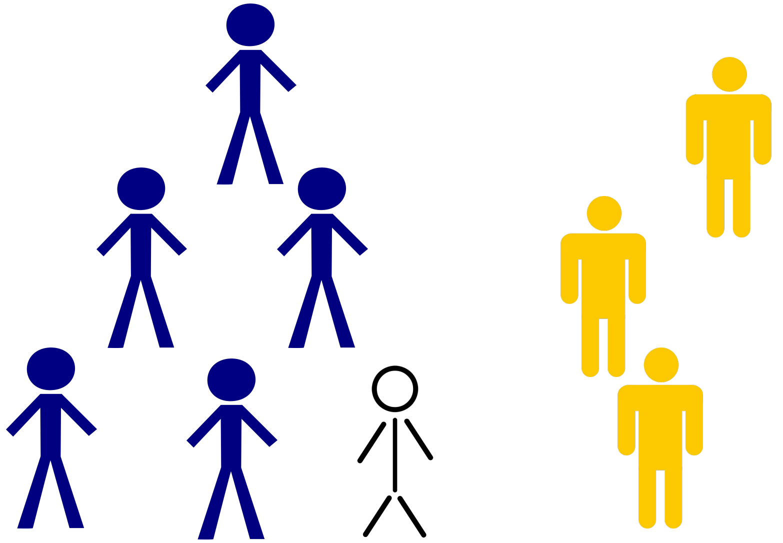
= Data Science Fellow



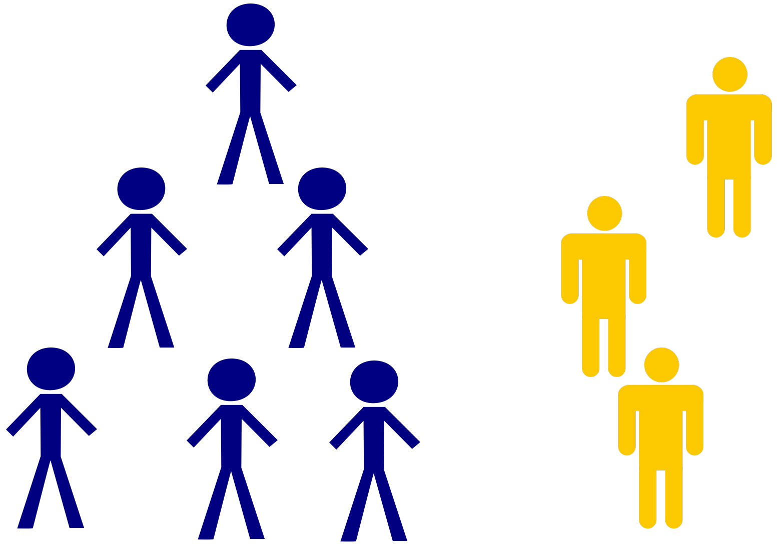
= Cyber Fellow



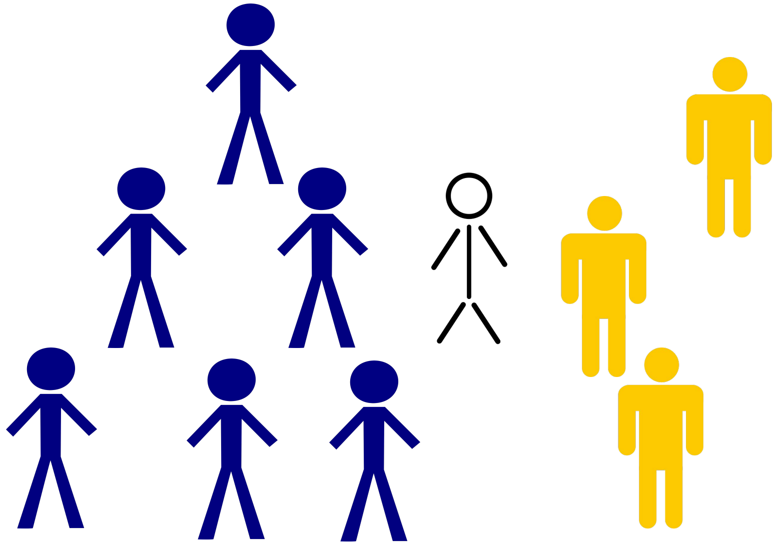
You recognize some cybersecurity fellows, as well as data science fellows.
You notice that everyone is congregating and chatting in groups.



But then, you notice someone that you do not recognize. Considering however that all cyber-security fellows are sticking together, **which fellowship does this person most likely belong to?**



Most likely a **cyber-security fellow,** no?



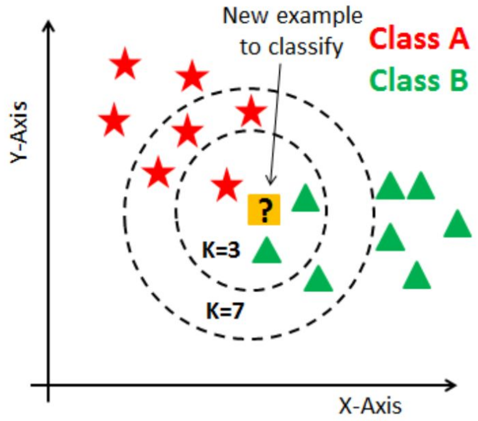
What if we have someone that’s on the **periphery of both groups**? We’ll solve this during our discussion of the kNN algorithm.

k-Nearest Neighbors (kNN)

We extend this idea to our **predictors** and **target labels**.

Given an integer *K* and test observation x_0, we identify the *K* **closest** points to x_0.

We estimate the **conditional probability** that x_0 belongs to **class *j*** as the fraction of **points whose response values equal *j*.**



$$P(Y=j|X=x_0) = \frac{1}{K} \sum_{i \epsilon N_0} I(y_i = j)$$

We formalize this idea via the following formula for conditional probability. Once again let's break this down before moving to an example.

$$P(Y=j|X=x_0) = \frac{1}{K}\sum_{i \in N_0} I(y_i = j)$$

The likelihood the **class of this sample is "j"**, given **the point x_0** is equal to

*Note, we identify the set of "K" nearest points as N_0. There's nothing special about this name (for now), we could have named it "Roofus" if we wanted to.

$$P(Y=j|X=x_0) = \frac{1}{K}\sum_{i \epsilon N_0} I(y_i = j)$$

For all "K" points that are "closest" to our test observation*

The likelihood the **class of this sample is "j"**, given **the point x_0** is equal to

*Another note, in the world of stats the "I()" function is an "indicator." If the condition in the parentheses is true it evaluates to 1, otherwise it is 0.

Calculate the sum of points that belong to class "j"*

$$P(Y=j|X=x_0) = \frac{1}{K}\sum_{i\epsilon N_0} I(y_i = j)$$

For all "K" points that are "closest" to our test observation*

The likelihood the **class of this sample is "j"**, given **the point x_0** is equal to

Divide by the number of neighbors we consider

Calculate the sum of points that belong to class "j"*

$$P(Y=j|X=x_0) = \frac{1}{K}\sum_{i \epsilon N_0} I(y_i = j)$$

For all "K" points that are "closest" to our test observation*

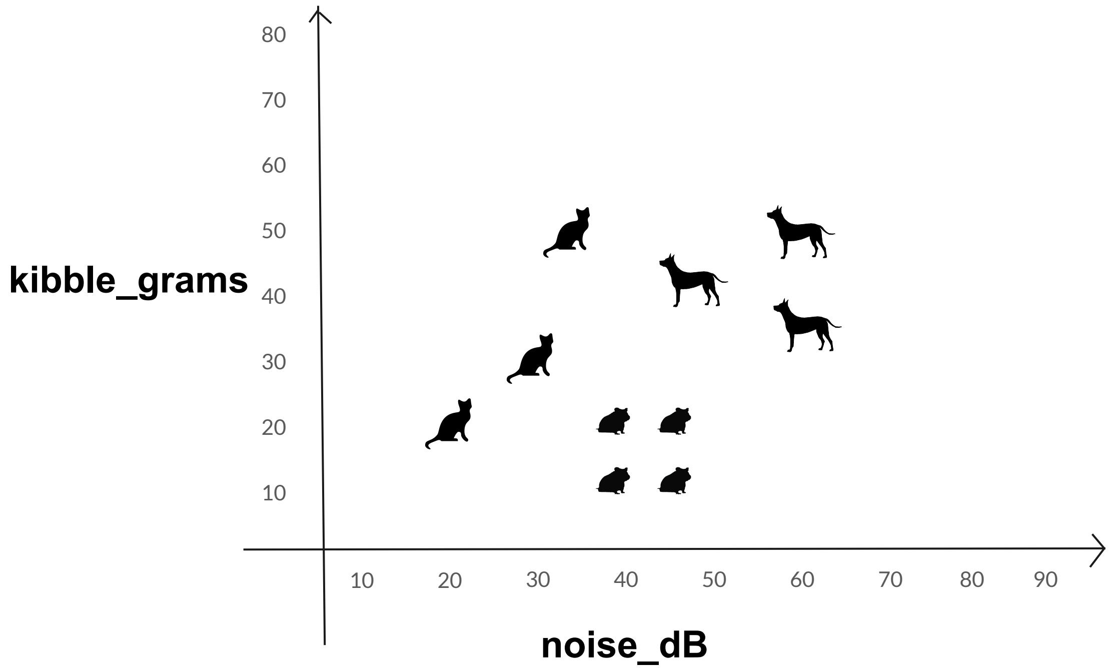
The likelihood the **class of this sample is "j"**, given **the point x_0** is equal to

You may be thinking "*Darn that's a whole lot of symbols to say something so simple.*"

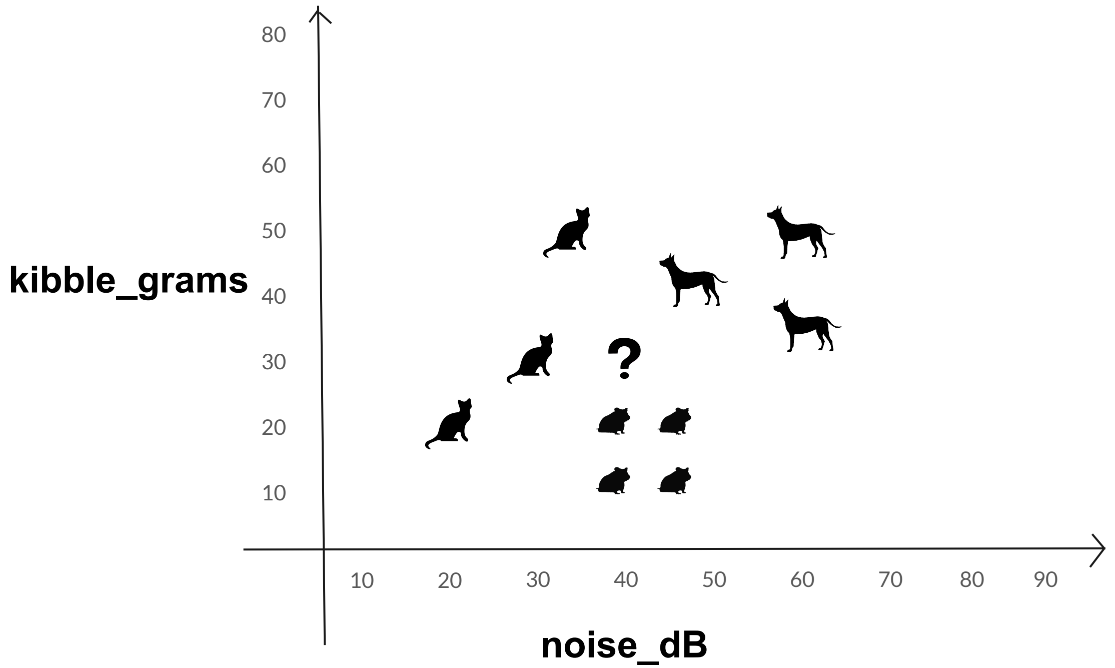
However, when it comes to creating ideas in maths, we **cannot be ambiguous**. Therefore we formalize these ideas in a **common set of symbols that all humans agree on**. It's a beautiful thing really.

$$P(Y=j|X=x_0) = \frac{1}{K}\sum_{i \epsilon N_0} I(y_i = j)$$

Succinctly put, we estimate the **conditional probability that x0 belongs to class j as the fraction of the K nearest points whose response values equal j**.

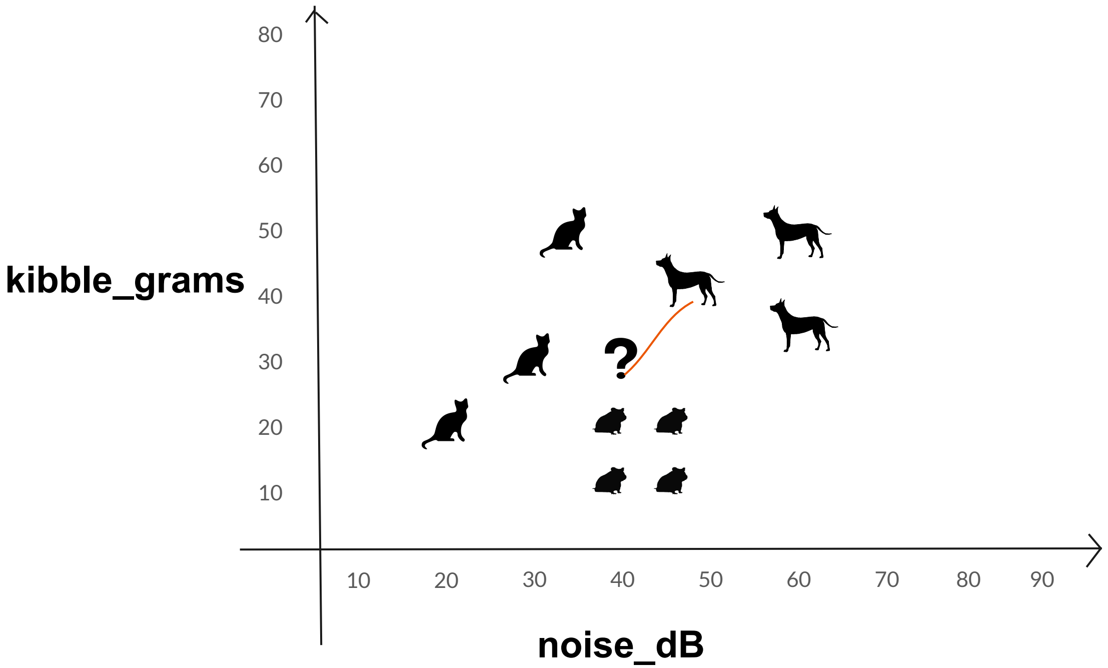


Let’s bring back our kibble/loud dataset that identifies **cats, hamsters, and dogs**. I reduced the data-points to make this graph easier to discuss. You may notice the axes now have equal range, this will be explained **tomorrow**.



$$P(Y=j|X=x_0) = \frac{1}{K}\sum_{i\epsilon N_0} I(y_i = j)$$

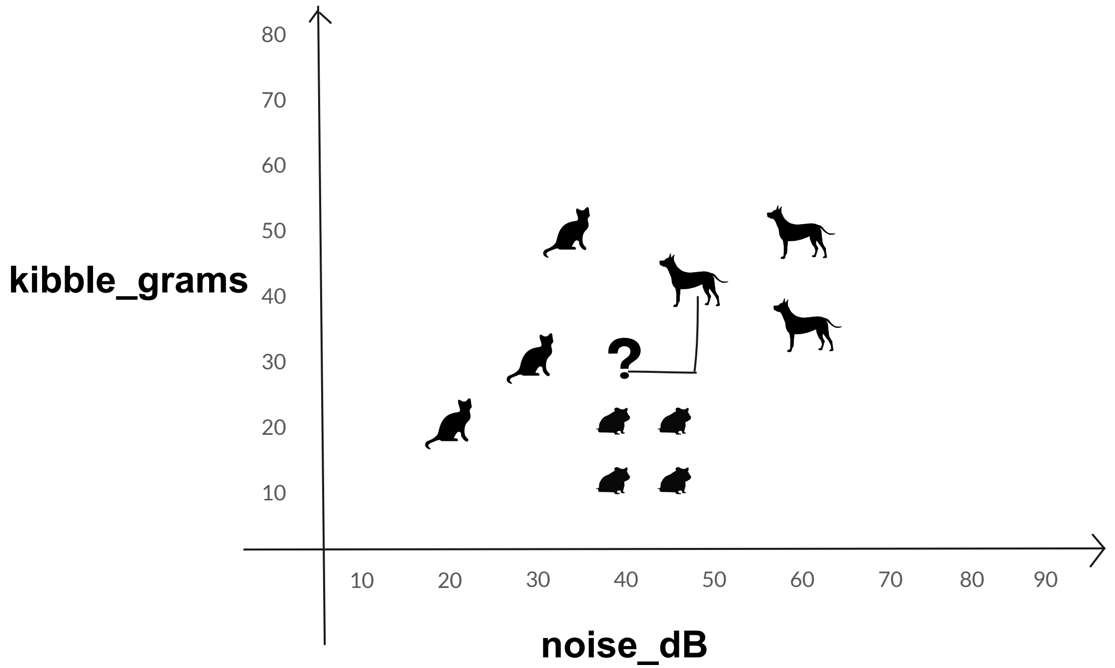
I introduce a mystery animal. Let's assume they are located on coordinate (40, 30). Let's estimate the probability that this animal is a cat, dog, or hamster using our **estimated conditional probability.**



$$P(Y=j|X=x_0) = \frac{1}{K}\sum_{i \epsilon N_0} I(y_i = j)$$

*Whenever you hear the word "Euclidian", just think about high-school geometry

One question we need to get out of the way is, how do we measure distance? Do we use the **euclidean distance** (l2 distance) where we **measure the direct distance between two points (**which is really the hypotenuse of a right triangle)



$$P(Y=j|X=x_0) = \frac{1}{K}\sum_{i \epsilon N_0} I(y_i = j)$$

*Whenever you hear the phrase“manhattan distance”, just think about the perfect grid structure of manhattan’s streets

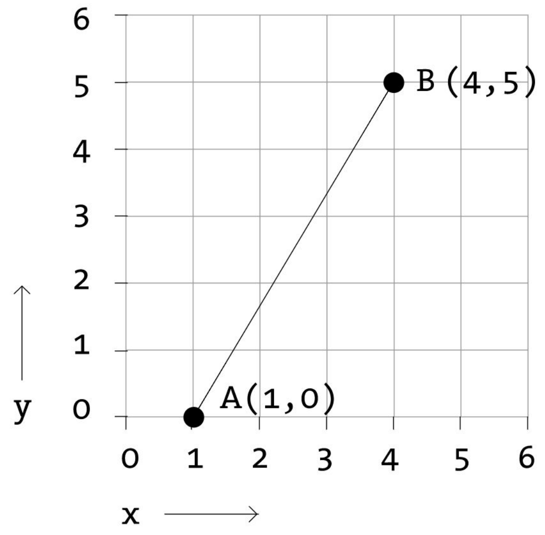
Or, do we measure the **shortest distance** when only taking **straight lines and right angles** aka the **manhattan distance (l1 distance)**.

Small Aside - Euclidean Distance (L2)

Let’s take a moment to touch on the mathematics of measuring distances between two points.

Thinking back to trigonometry (the study of triangles), the distance between two points can always be expressed as the **hypotenuse of a right triangle**.

Anyone recall **pythagorean's theorem**?

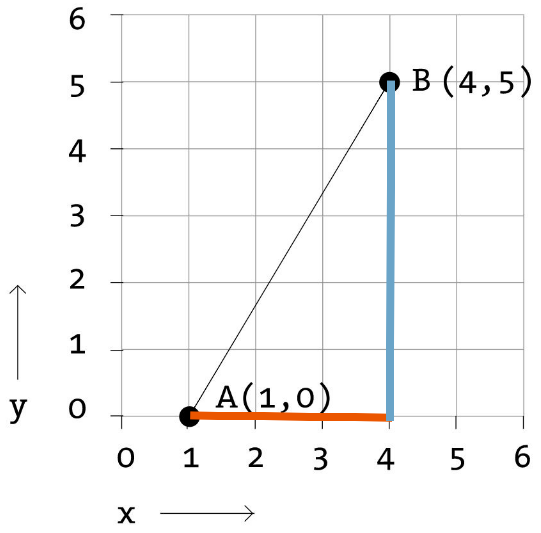


Small Aside - Euclidean Distance (L2)

$$a^2 + b^2 = c^2$$

Since we're interested in "*c*" the hypotenuse, let's solve for "*c*"

Any volunteers before we show the solution?



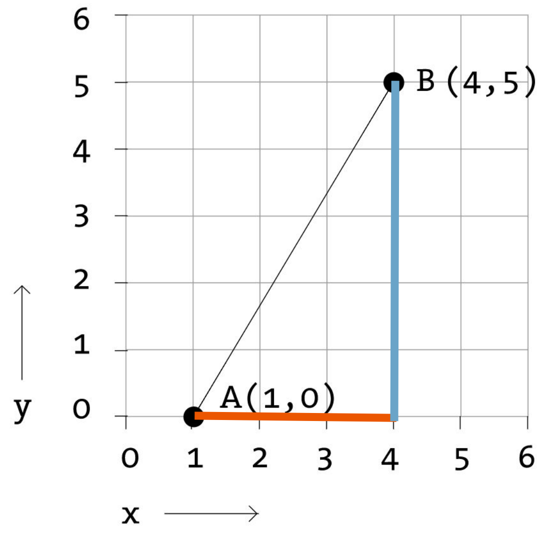
Small Aside - Euclidean Distance (L2)

$$a^2 + b^2 = c^2$$

Since we're interested in "*c*" the hypotenuse, let's solve for "*c*"

$$\sqrt{a^2 + b^2} = \sqrt{c^2}$$

$$c = \sqrt{a^2 + b^2}$$

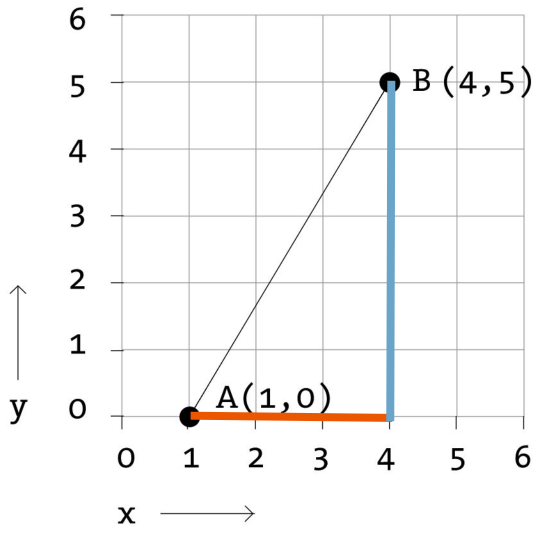


Small Aside - Euclidean Distance (L2)

$$c = \sqrt{a^2 + b^2}$$

So we need to figure out the lengths of "*a*" and "*b*" to solve for "*c*" (the euclidean distance).

Considering the points on the graph, what is the measure of the "*a*" side?



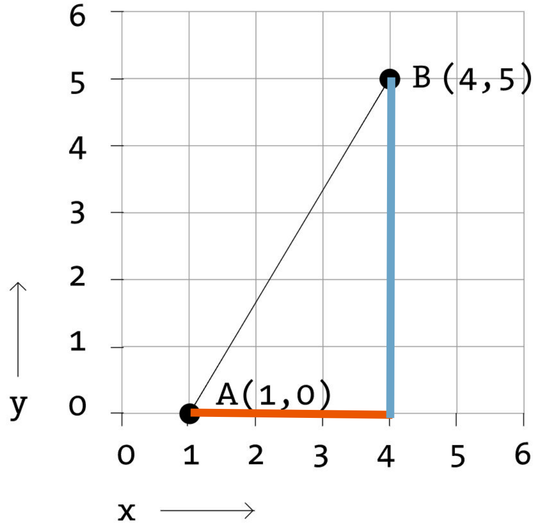
Small Aside - Euclidean Distance (L2)

$$c = \sqrt{a^2 + b^2}$$

a = *4 - 1 = 3*

In a more general sense, this is *x1 - x0*

How about the "*b*" side?



Small Aside - Euclidean Distance (L2)

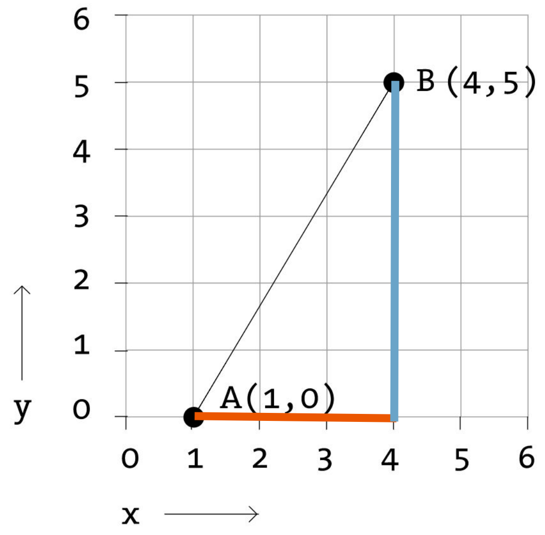
$$c = \sqrt{a^2 + b^2}$$

b = *5 - 0* = *5*

In a more general sense, this is *y1 - y0*

Therefore, we can generalize this equation to:

$$c = \sqrt{(x1 - x0)^2 + (y1 - y0)^2}$$

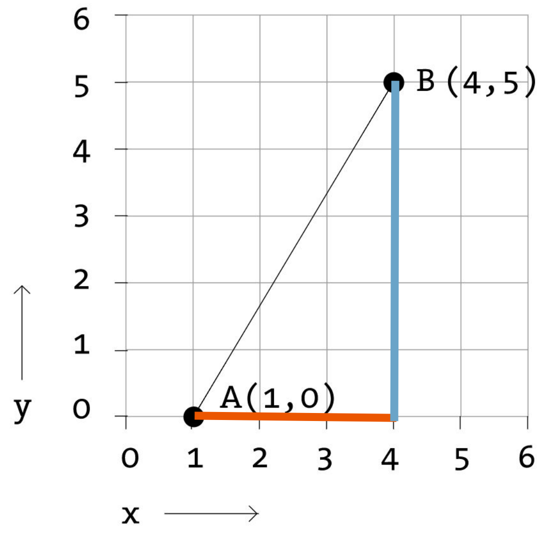


Small Aside - Euclidean Distance (L2)

$$c = \sqrt{(x1 - x0)^2 + (y1 - y0)^2}$$

Let's plug our values in and solve. Try this yourself.

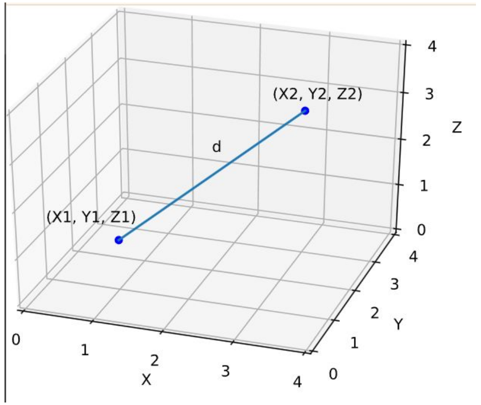
$$c = \sqrt{(4 - 1)^2 + (5 - 0)^2}$$



Small Aside - Euclidean Distance (L2)

"But wait, (Anil/Farukh), this is only for two dimensions, how do you solve for 3? Or 4? Or 500 dimensions?"

EZ! Turns out the pythagorean theorem works for all dimensions, simply add the additional dimension in your formula:



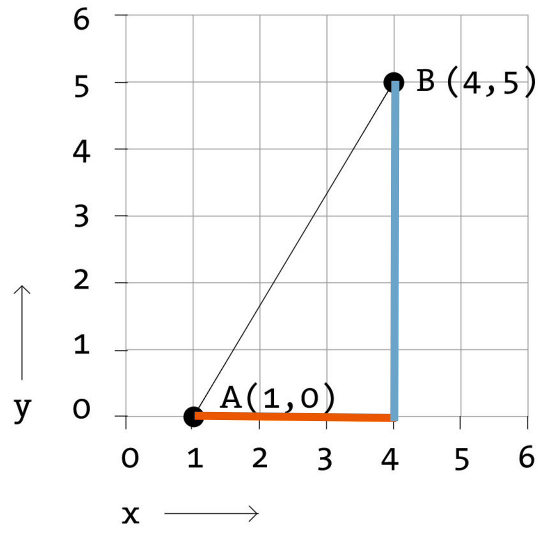
$$c = \sqrt{(x1 - x0)^2 + (y1 - y0)^2 + (z1 - z0)^2}$$

Small Aside - Manhattan Distance (L1)

Now, picture we can only travel in straight lines and turn in right angles (as if we were on the streets of **manhattan**).

This entails simply adding the sides of our right triangle!

Can anyone calculate this?



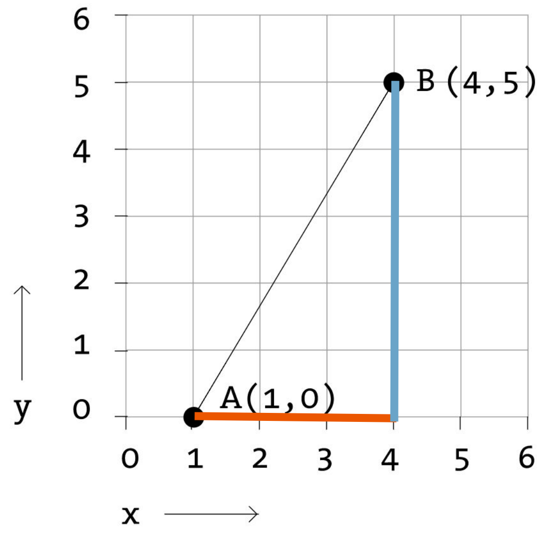
Small Aside - Manhattan Distance (L1)

Now, picture we can only travel in straight lines and turn in right angles (as if we were on the streets of **manhattan**).

This entails simply adding the sides of our right triangle!

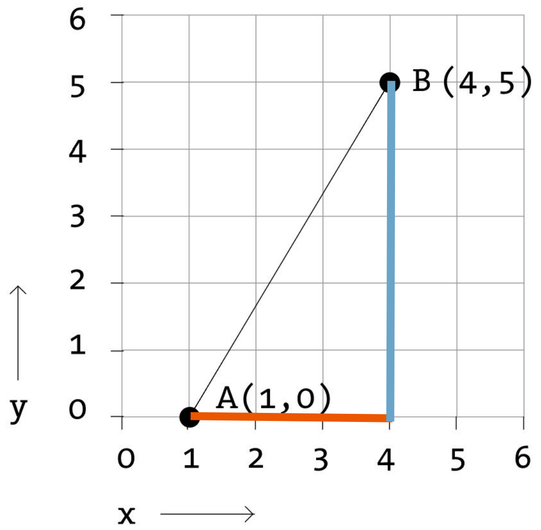
Can anyone calculate this?

And perhaps propose a formula for us to use?



Small Aside - Manhattan Distance (L1)

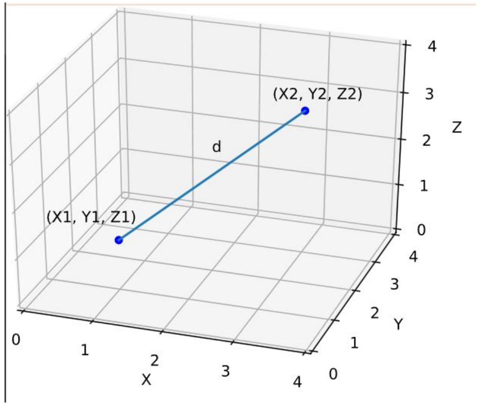
$$|(x1 - x0)| + |(y1 - y0)|$$



This is simply the absolute value (distance between points) added for each dimension.

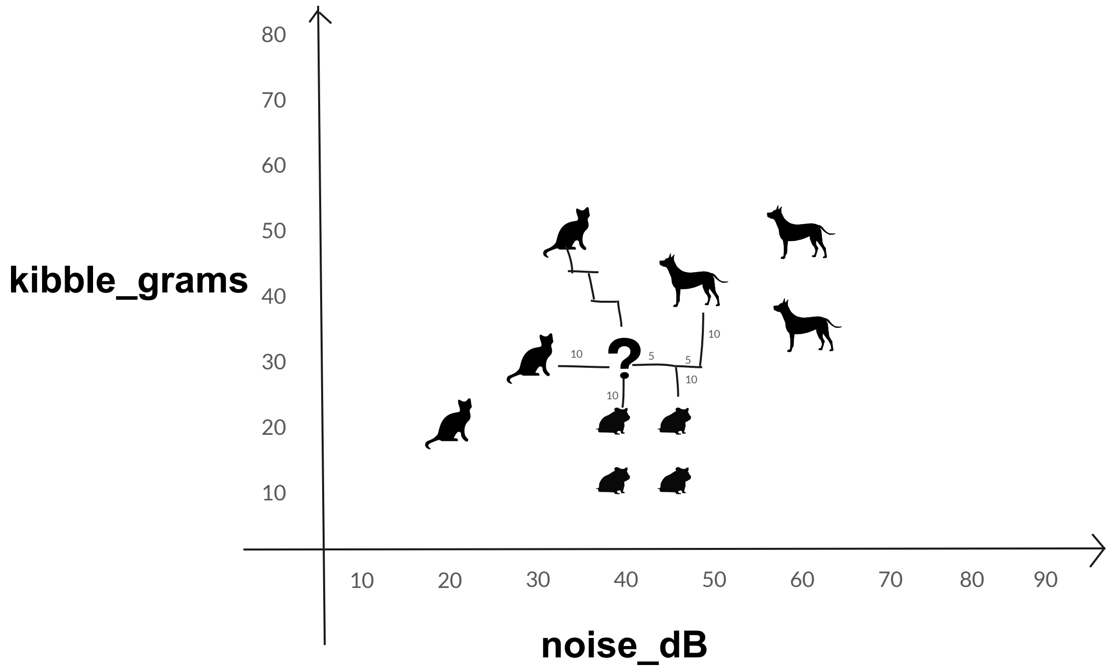
So once again, if we have more than 2 dimensions…

Small Aside - Manhattan Distance (L1)



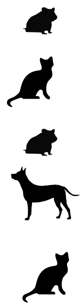
We simply include that measure of distance in that dimension to our formula.

$$|(x1 - x0)| + |(y1 - y0)| + |(z1 - z0)|$$



$$P(Y=j|X=x_0) = \frac{1}{K}\sum_{i\epsilon N_0} I(y_i = j)$$

k=3



distance = 10

distance = 10

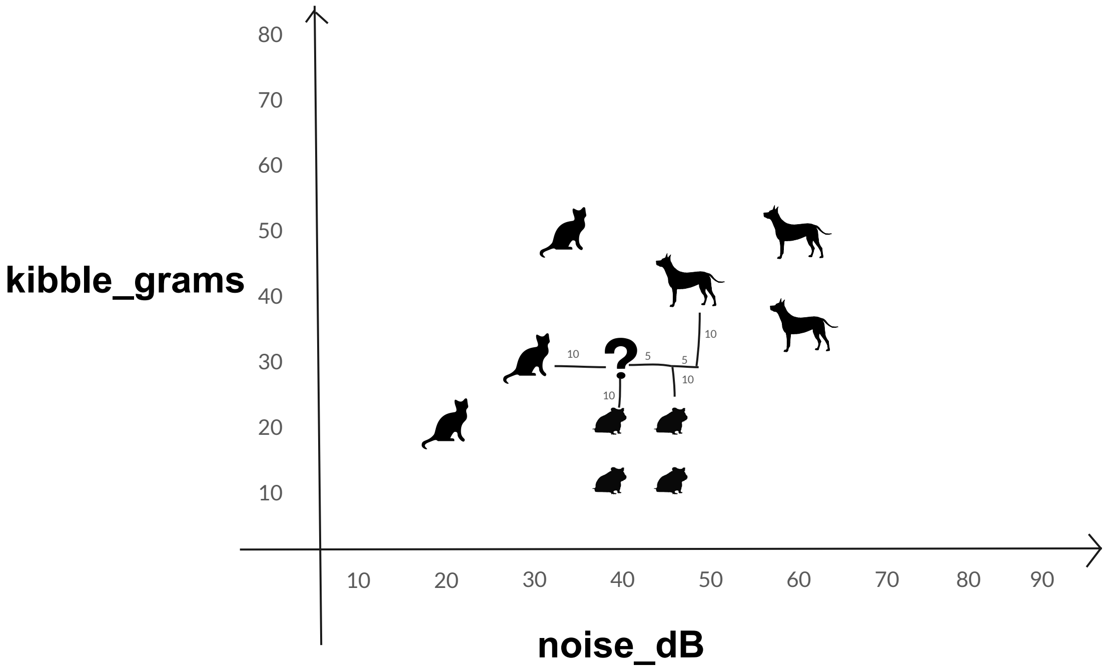
distance = 15

distance = 20

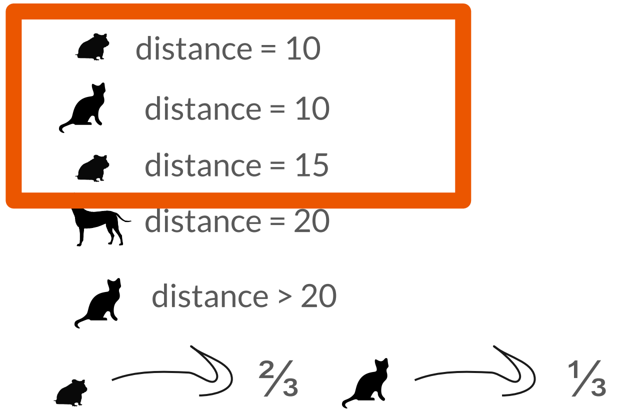
distance > 20

For now, let's use manhattan distance to get the **K=3 closest neighbors**. Who are the 3 closest neighbors?

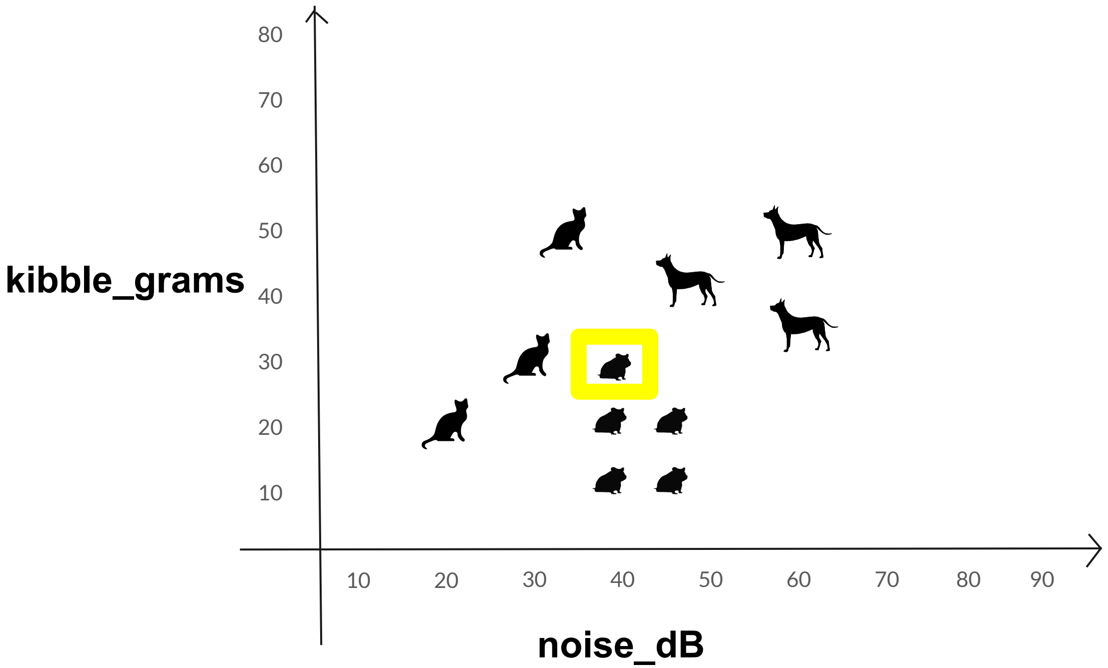
$$P(Y=j|X=x_0) = \frac{1}{K}\sum_{i \epsilon N_0} I(y_i = j)$$



k=3



Which class contains the highest ratio?



$$P(Y=j|X=x_0) = \frac{1}{K}\sum_{i\epsilon N_0} I(y_i = j)$$

k=3

distance = 10

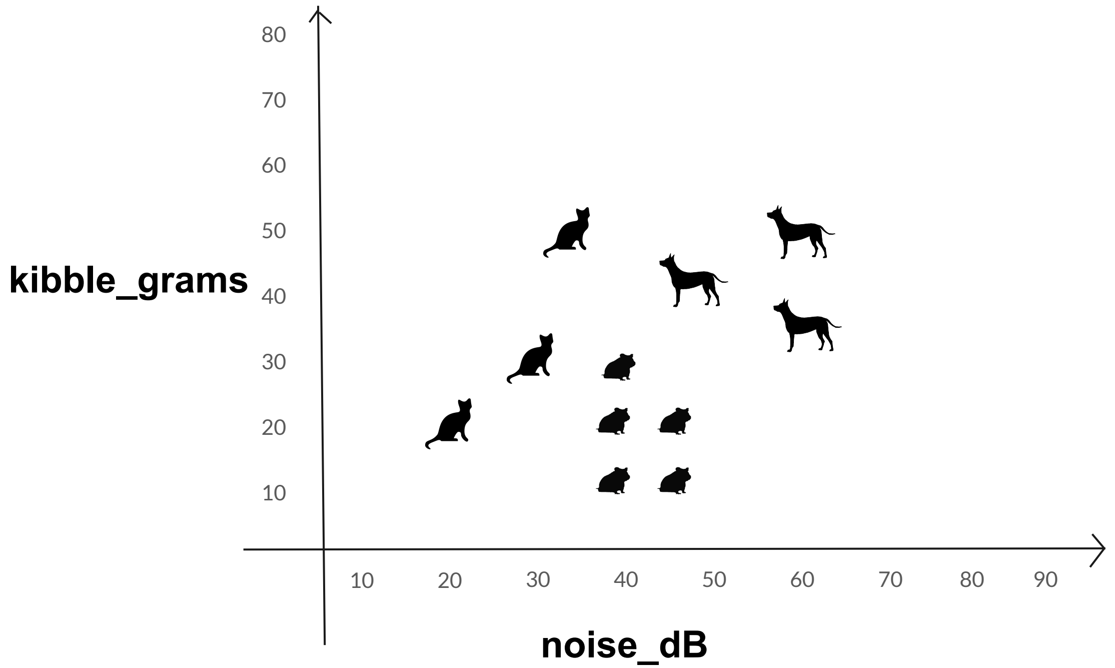
distance = 10

distance = 15

distance = 20

$\longrightarrow \frac{2}{3}$ $\longrightarrow \frac{1}{3}$

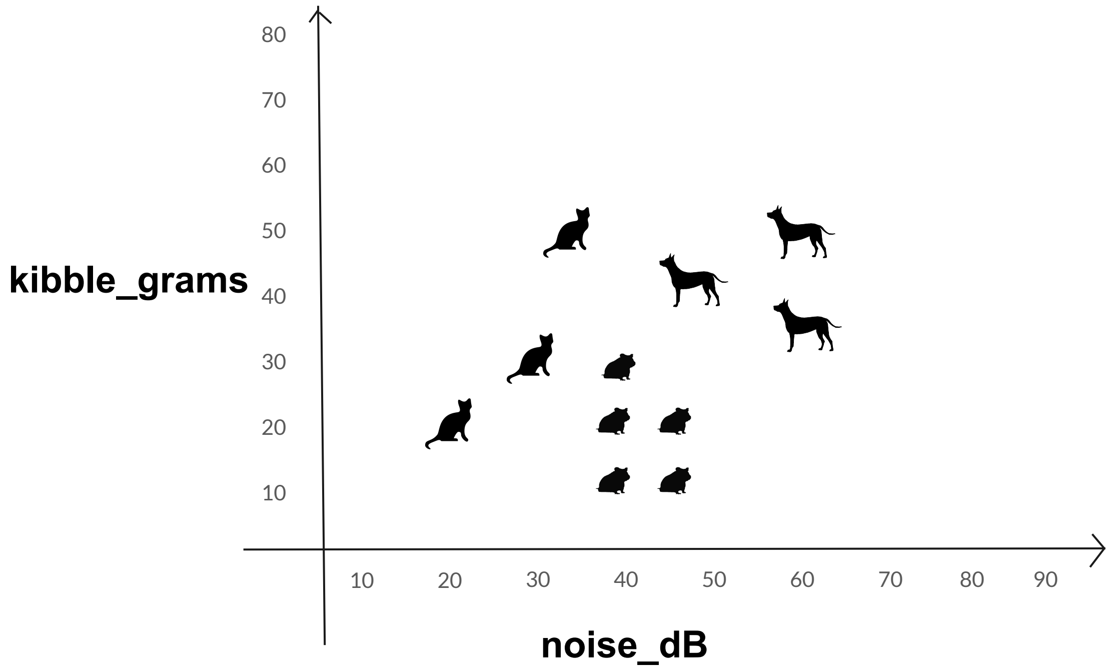
Team hamster is victorious once again.



We're going to pause our discussion of kNN here.

However there is more to evaluate when discussing this **simple, yet powerful algorithm.**

What are some questions you might have regarding kNN?

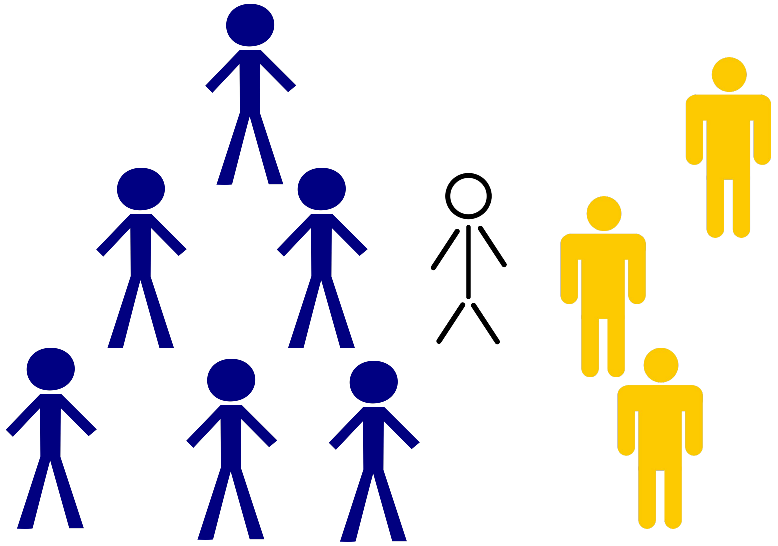


We're going to pause our discussion of kNN here.

However there is more to evaluate when discussing this **simple, yet powerful algorithm.**

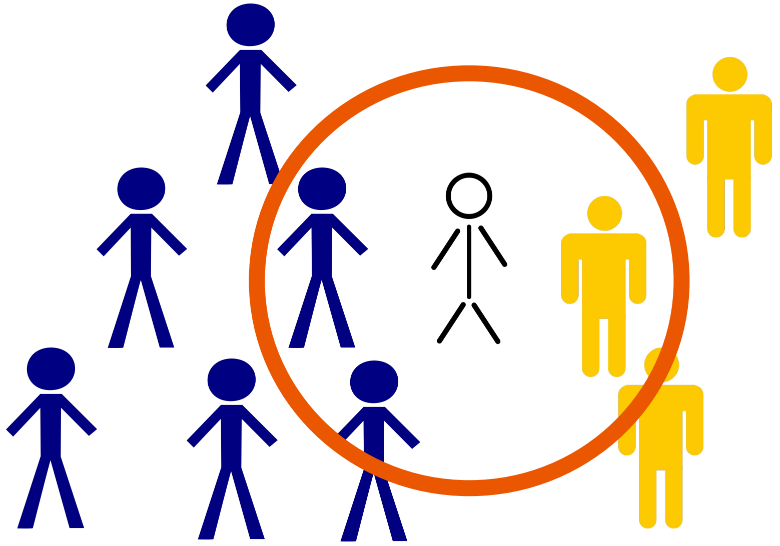
What are some questions you might have regarding kNN?

- **Why are my axes normalized?**
- **When do we use euclidean vs manhattan?**
- **What happens if we have categorical data?**
- **What happens as I increase my dimensions?**
- **What are some cons to kNN?**



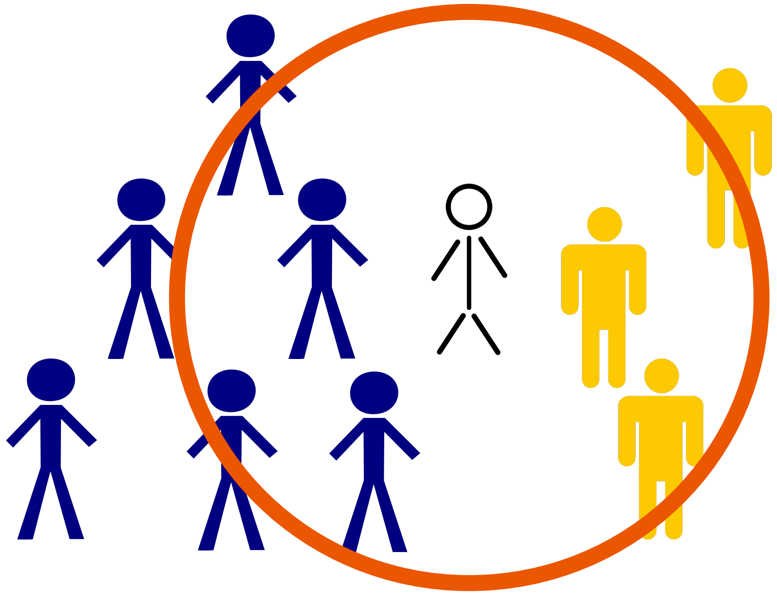
Going back to our TKH industry event discussion, we simply select some “K” that accurately classifies this person as a data or cyber fellow.

k=3



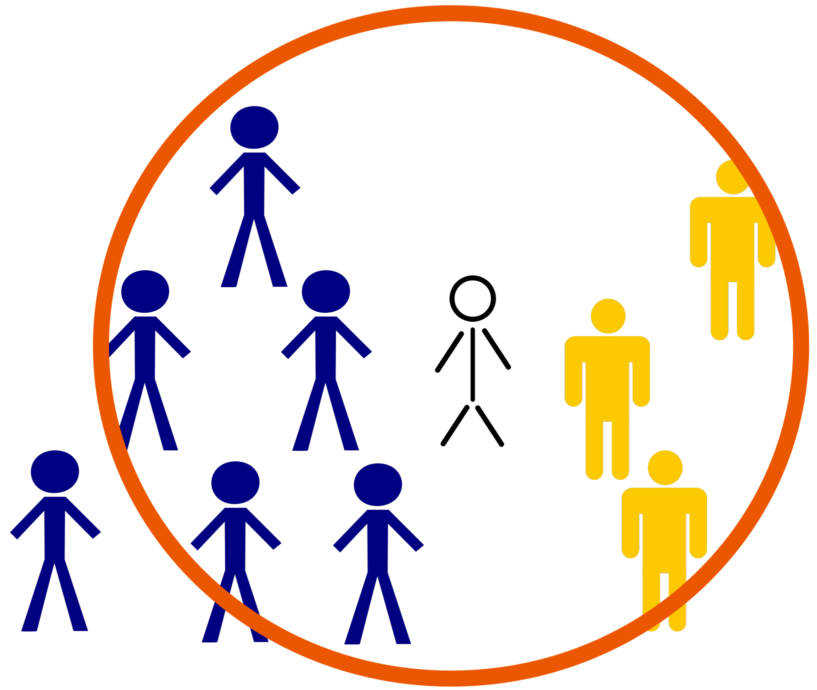
Class = Cyber

k=5



Class = Cyber

k=8



Class = Cyber (*are we suffering from imbalance???*)

Choosing a K - HyperParam

*Decision boundaries can be calculated for any classifier!!

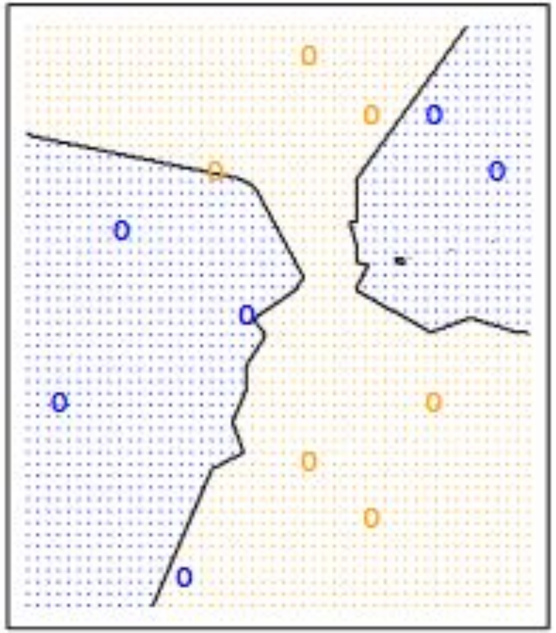
kNN - Decision Boundaries

Let’s expand this discussion to kNN.

When learning about classification algorithms, we utilize **decision boundaries** to visualize the behaviour of models.

This works well as the behavior of our boundaries generalize to any **n-dimensional space.**

This signifies “*for any point in this space, it will be classified as Y point*”

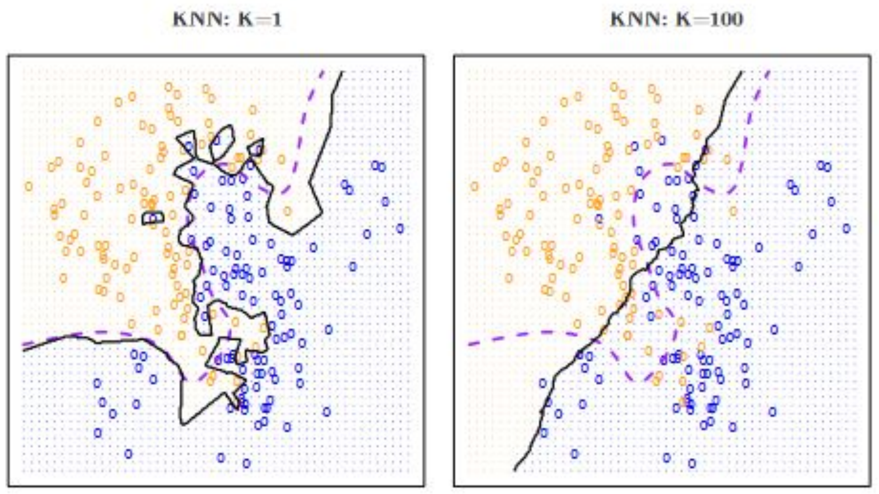


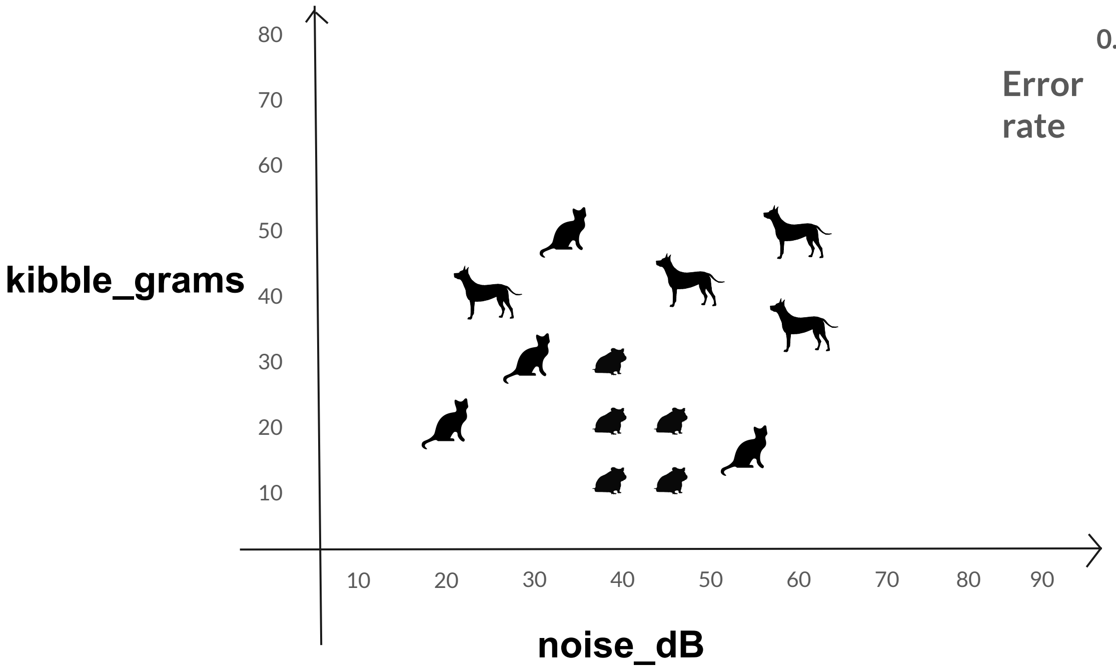
kNN - Choosing a K

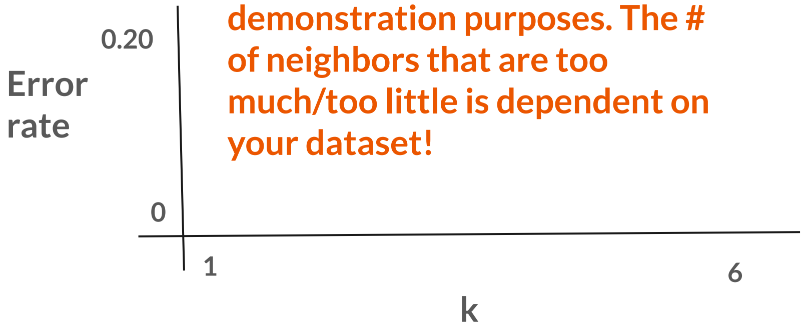
“*The choice of K has a **drastic** effect on the KNN classifier obtained.*”

Let’s observe how our decision boundaries (and subsequent accuracy) shift as we increase the **number of neighbors** to consider.

What do we notice happens to the decision boundary as we go from K=1 to K=100?







Please note: do not take these values at face value. They are only intended for demonstration purposes. The # of neighbors that are too much/too little is dependent on your dataset!

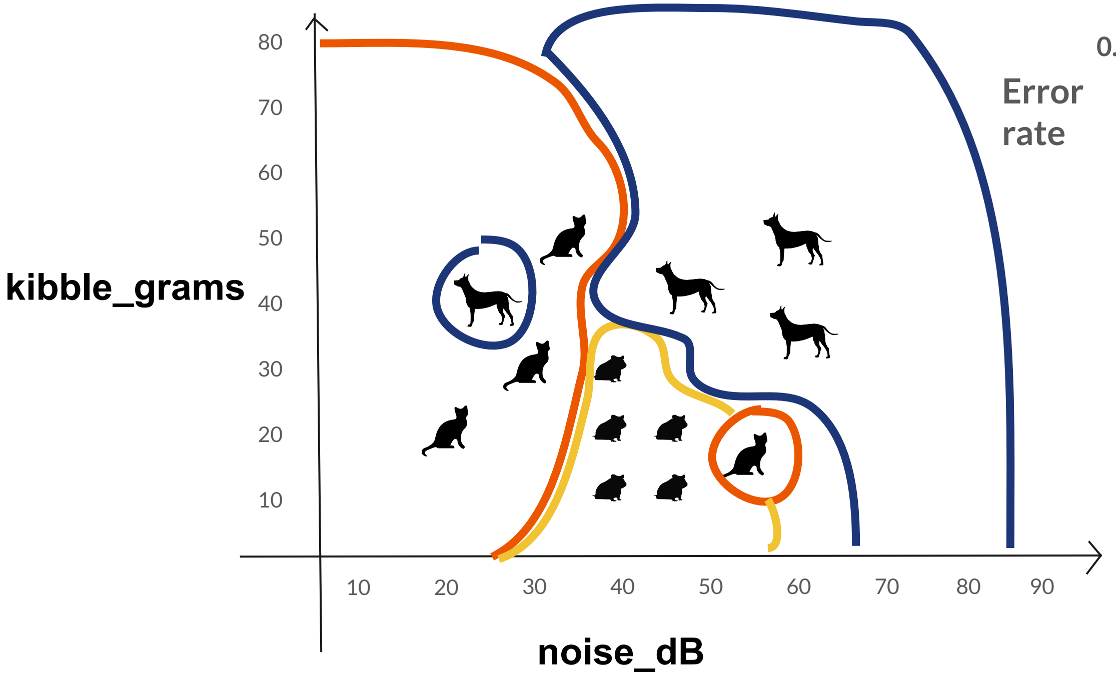
● Training accuracy

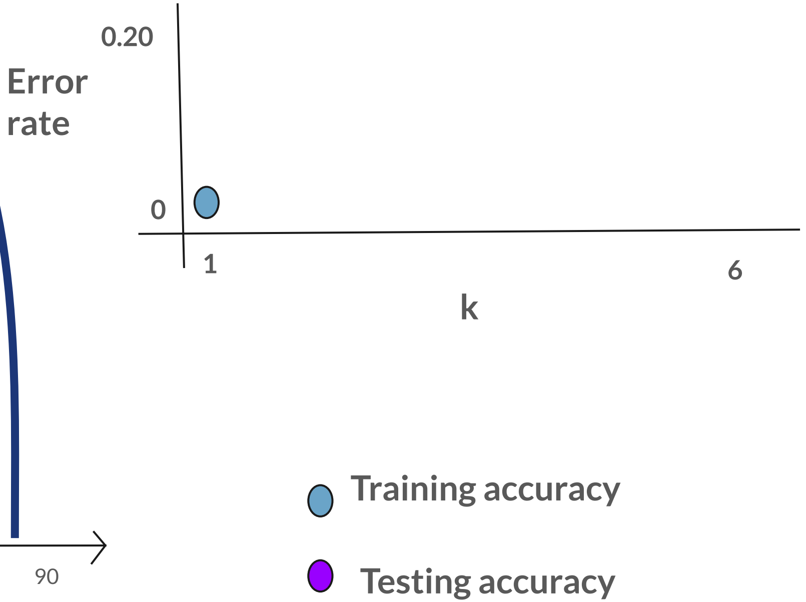
● Testing accuracy

Let's bring back our scatter plot (with a twist!)

To obtain the decision boundary for kNN, we could **compute the class using kNN** for **each possible point in our search space**. Let's see how our **error rate** updates as we increase K.

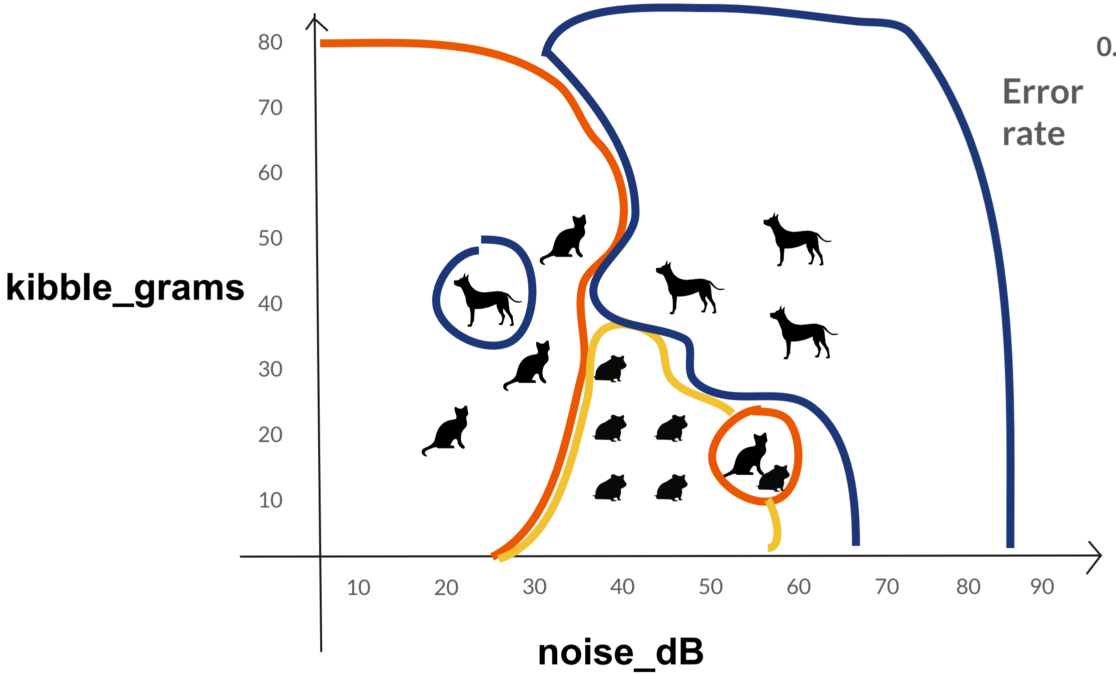
Red = cat; yellow = hammie; blue = dog

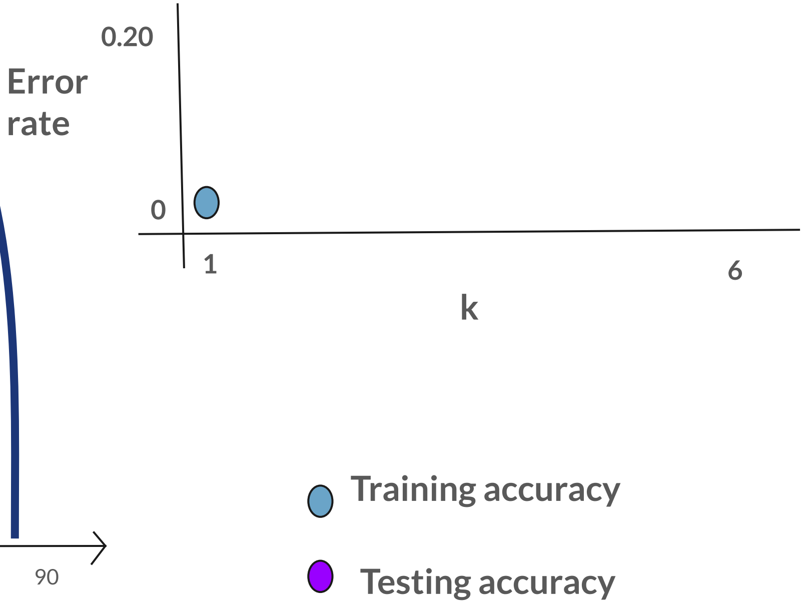




We begin with **k=1. What do you notice about the error rate?** Does it make any mistakes?

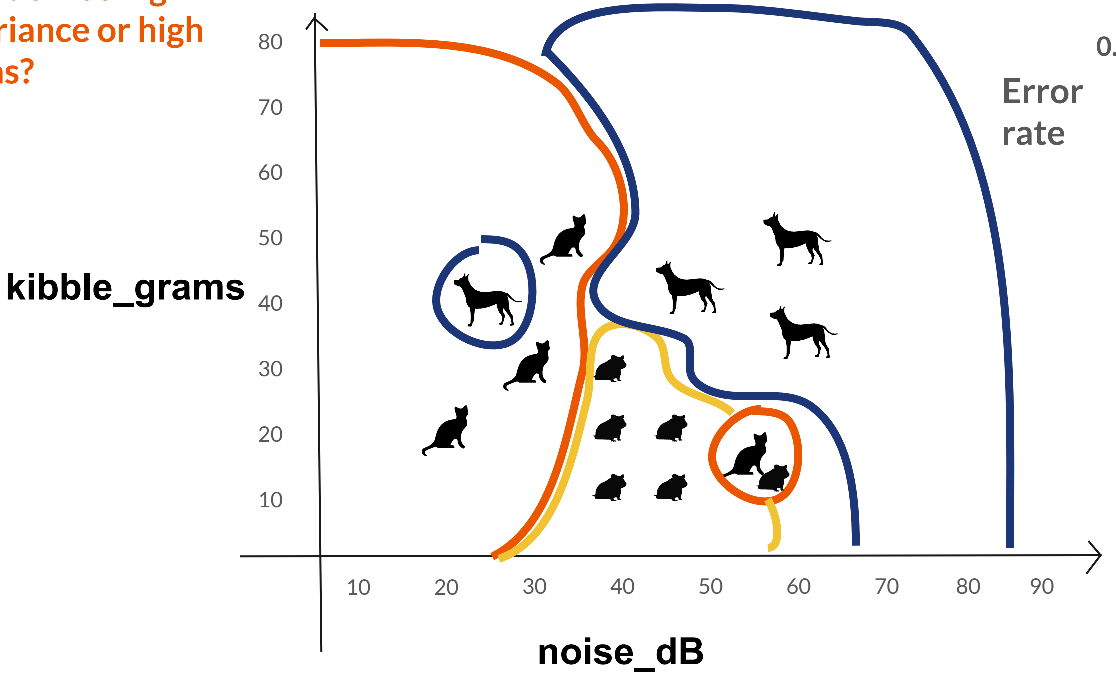
Red = cat; yellow = hammie; blue = dog

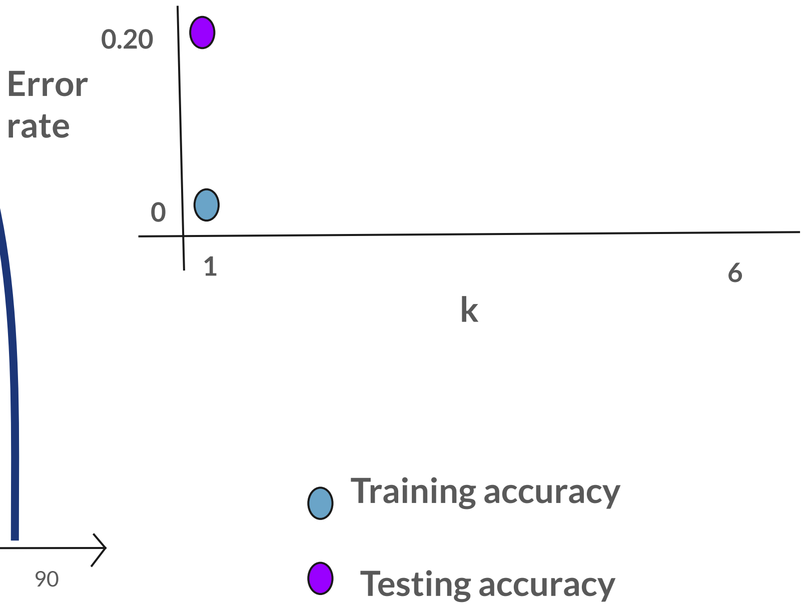




None whatsoever! While our "training" rate might seem ideal, let's see what happens **when we introduce a new test sample. Who is our hamster friend classified as?**

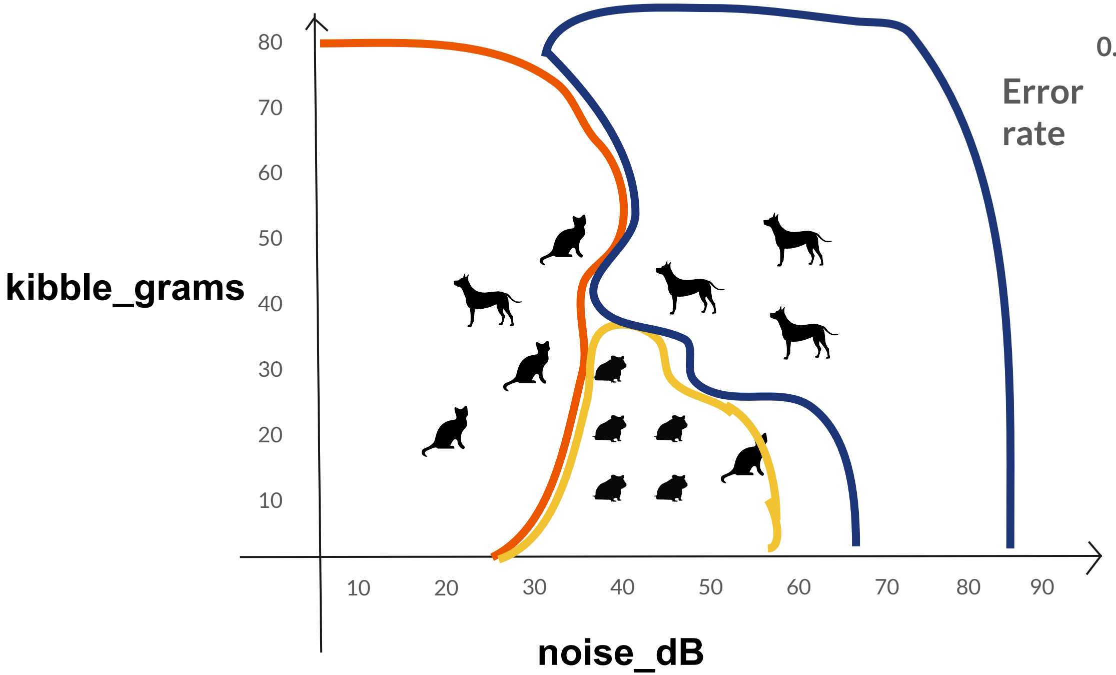
Do we say that this model has high variance or high bias?

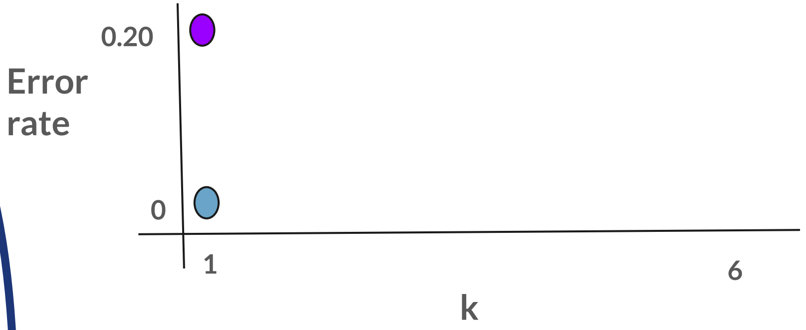




A cat. Remember everyone, a classifier that perfectly fits our training data could be a terrible classifier. We need this model to generalize well on new data.

Red = cat; yellow = hammie; blue = dog



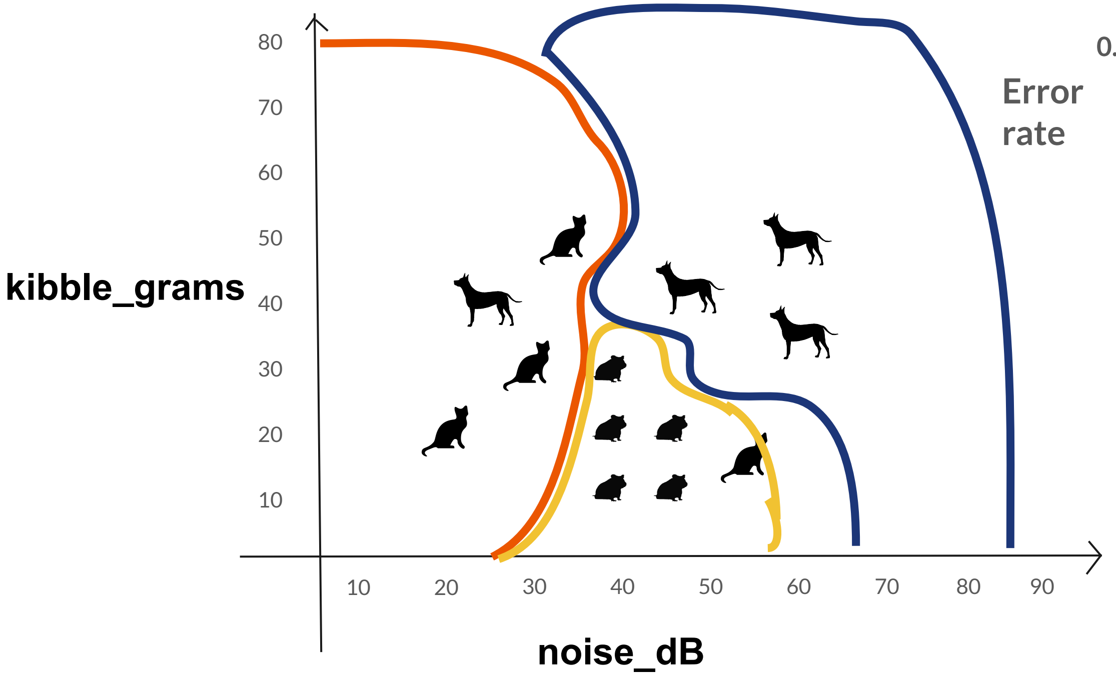


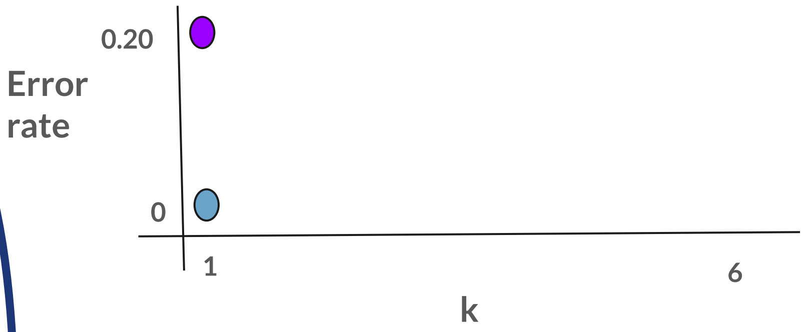
● Training accuracy

● Testing accuracy

We move onto **k=3. What happens to our training error rate now?**

Red = cat; yellow = hammie; blue = dog

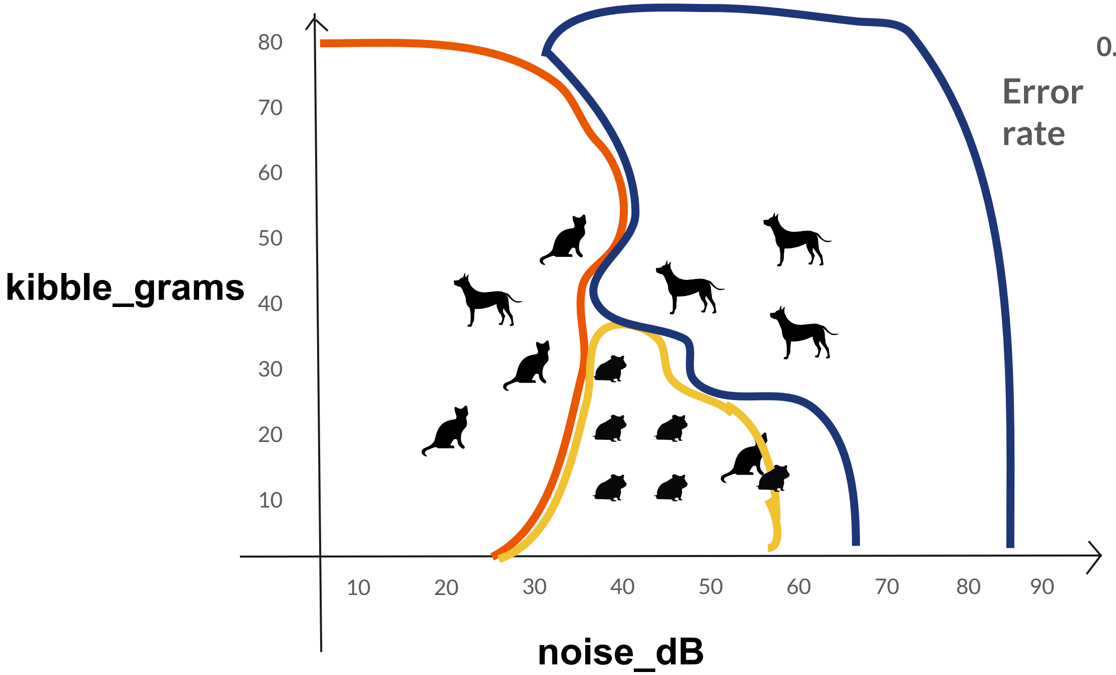


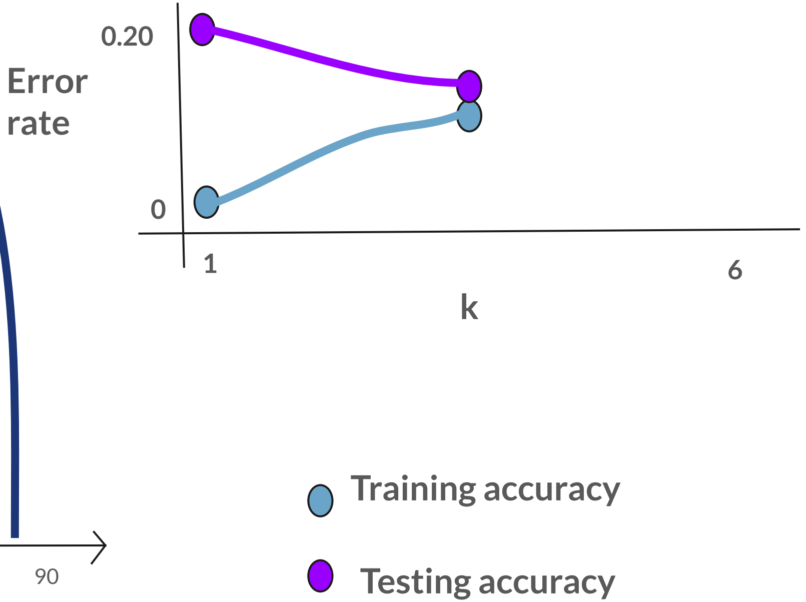


● Training accuracy

● Testing accuracy

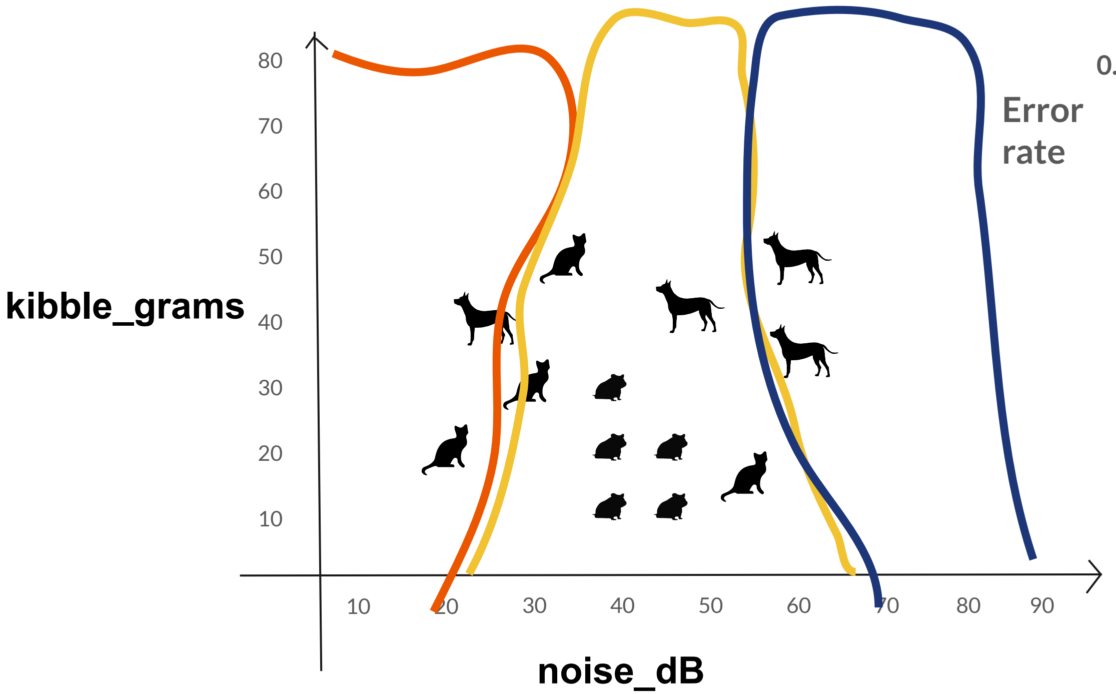
Unfortunately a few training samples are classified incorrectly, but this could actually be preferable when it comes to our testing data!

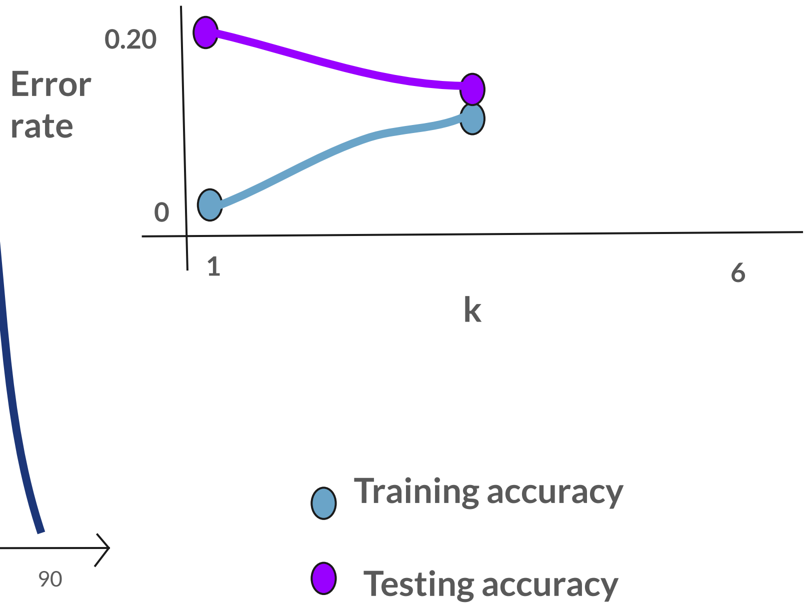




Bringing back our hamster, we see that it **is now correctly classified as a hamster** (ignore the roughness of my sketches)

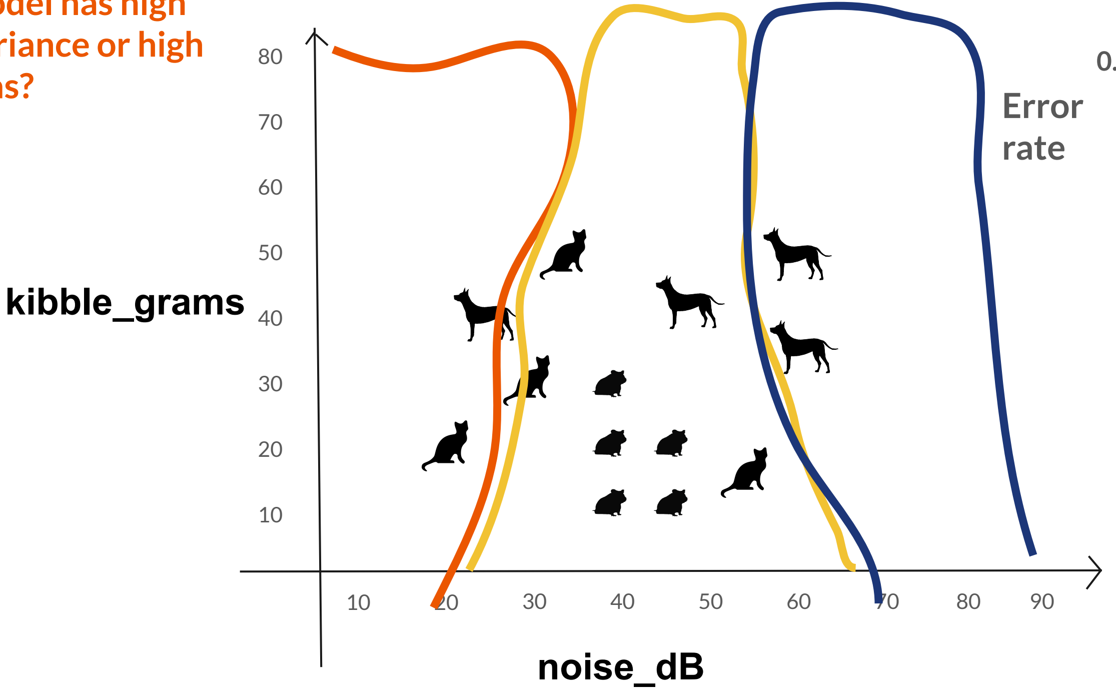
Red = cat; yellow = hammie; blue = dog

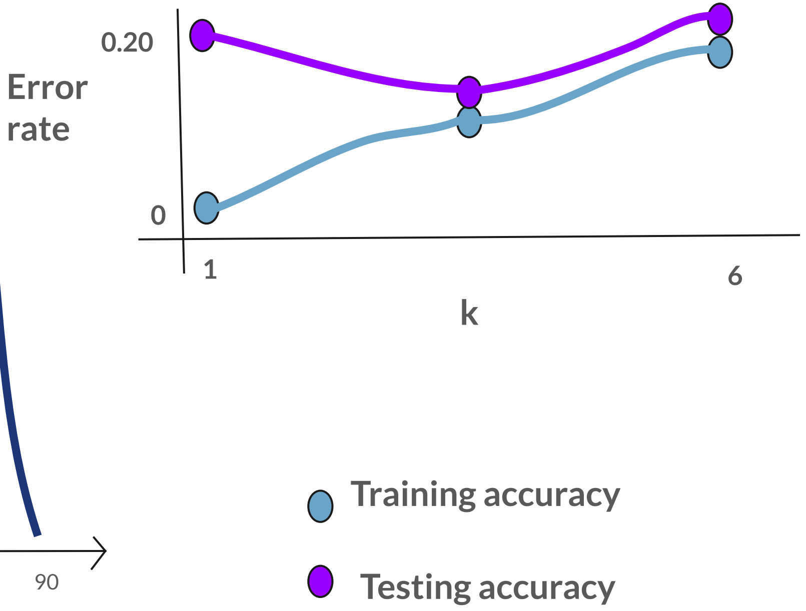




Finally lets try **k=6. What happens to our training error rate now?**

Do we say that this model has high variance or high bias?





Too many neighbors result in **linear decision boundaries.** This maximizes both training and test error. **We need to discover the best possible `k` via hyperparameter search.**

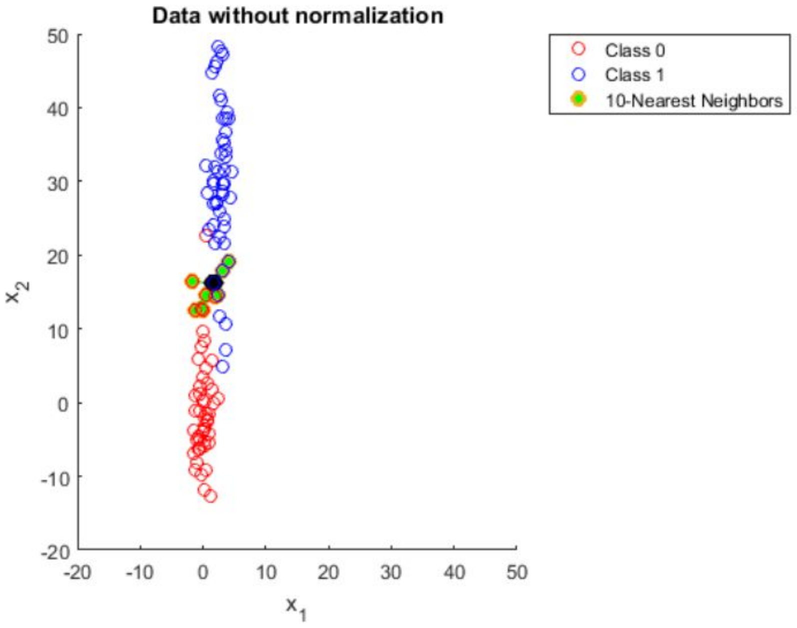
Credit to: https://stats.stackexchange.com/questions/287425/why-do-you-need-to-scale-data-in-knn

kNN - Importance of Data Scaling

One thing we established without discussing is the **importance of scaling our dataset** before we begin using kNN.

We do this to ensure that **kNN assigns equal importance** to features when considering distances.

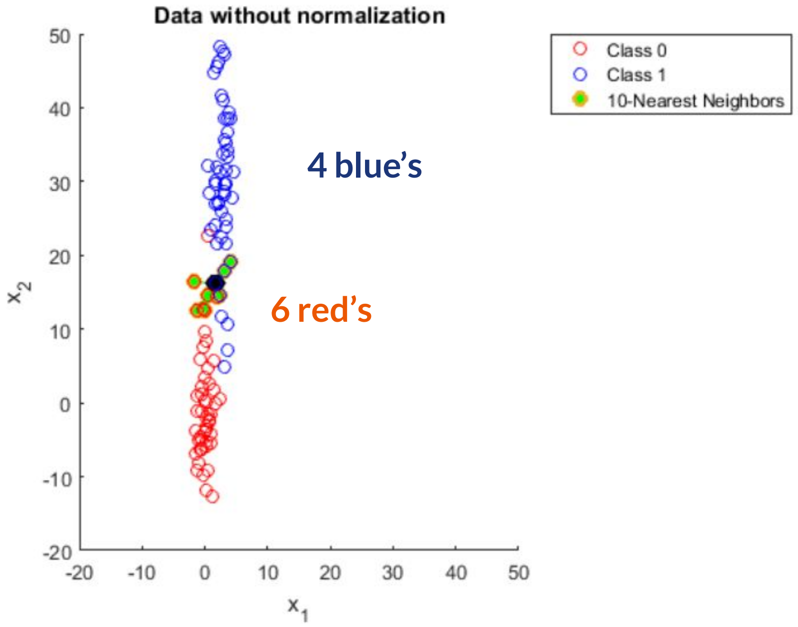
Let's see a visual example (via euclidean)



kNN - Importance of Data Scaling

X1: (-2 to 2)
X2: (-20 to 50)

When running kNN, all nearest neighbors are **aligned to the axes with the smallest range**.

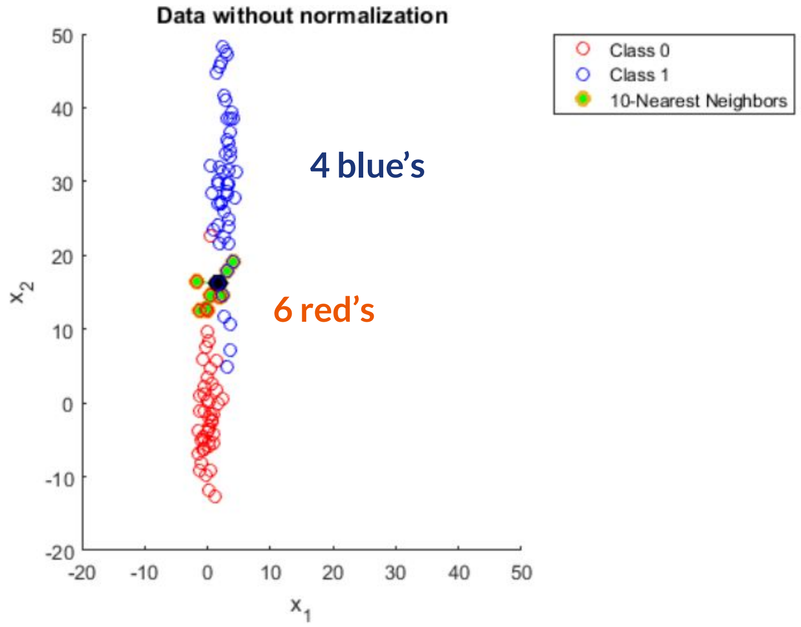


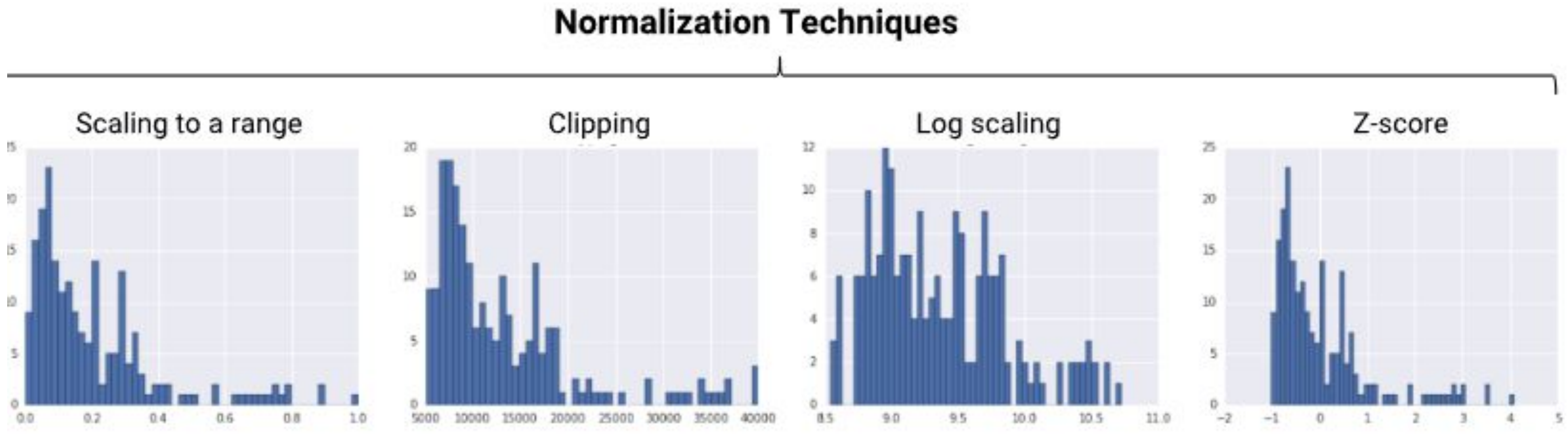
kNN - Importance of Data Scaling

X1: (-2 to 2)
X2: (-20 to 50)

This is because the “distance” of **choosing a neighbor that varies in the X2 direction** is too large.

$$c = \sqrt{(0-1)^2 + (5-15)^2}$$





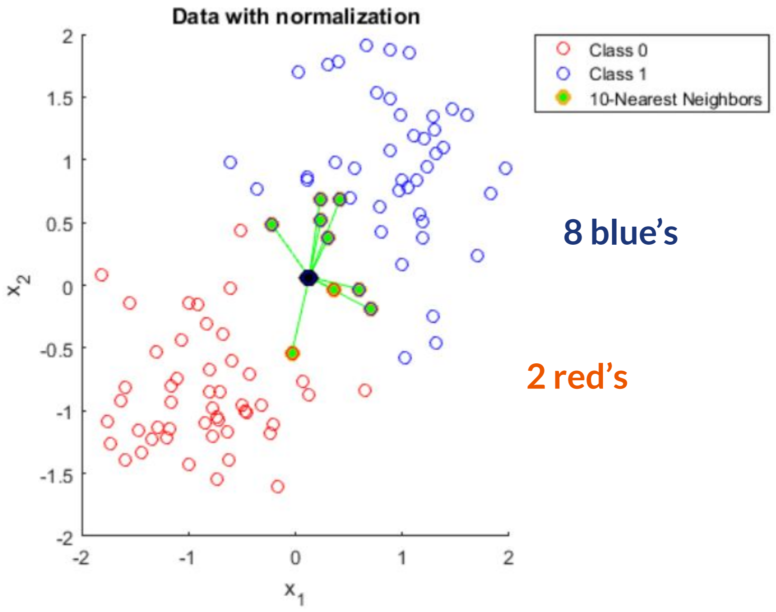
We have a couple of options when **normalizing our data**. For now, we will use **standard scaler**, but for a review check out: https://developers.google.com/machine-learning/data-prep/transform/normalization

kNN - Importance of Data Scaling

X1: (-2 to 2)
X2: (-2 to 2)

Now that we've standardized our dataset, we **can assign equal importance to dimensions.**

Subsequently, we update our predictions.



8 blue's

2 red's

kNN

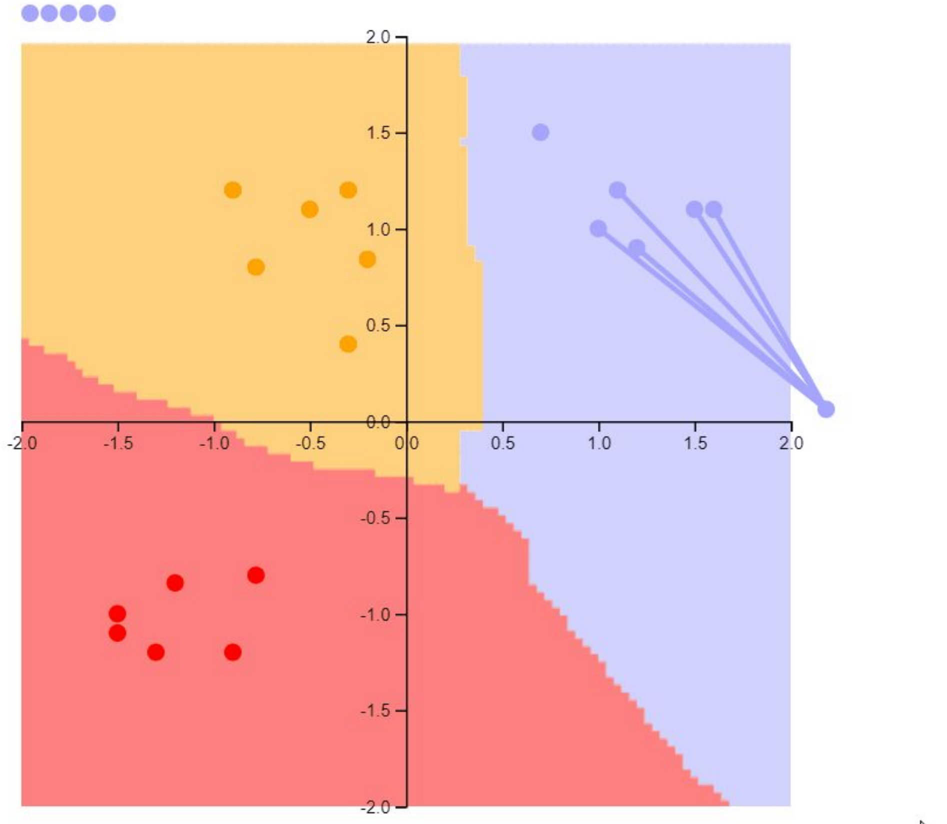
To conclude our conversation on the supervised learning classifier kNN, it is a **powerful non-parametric supervised learning algorithm that utilizes a fairly “lazy” classification method**.

Pros

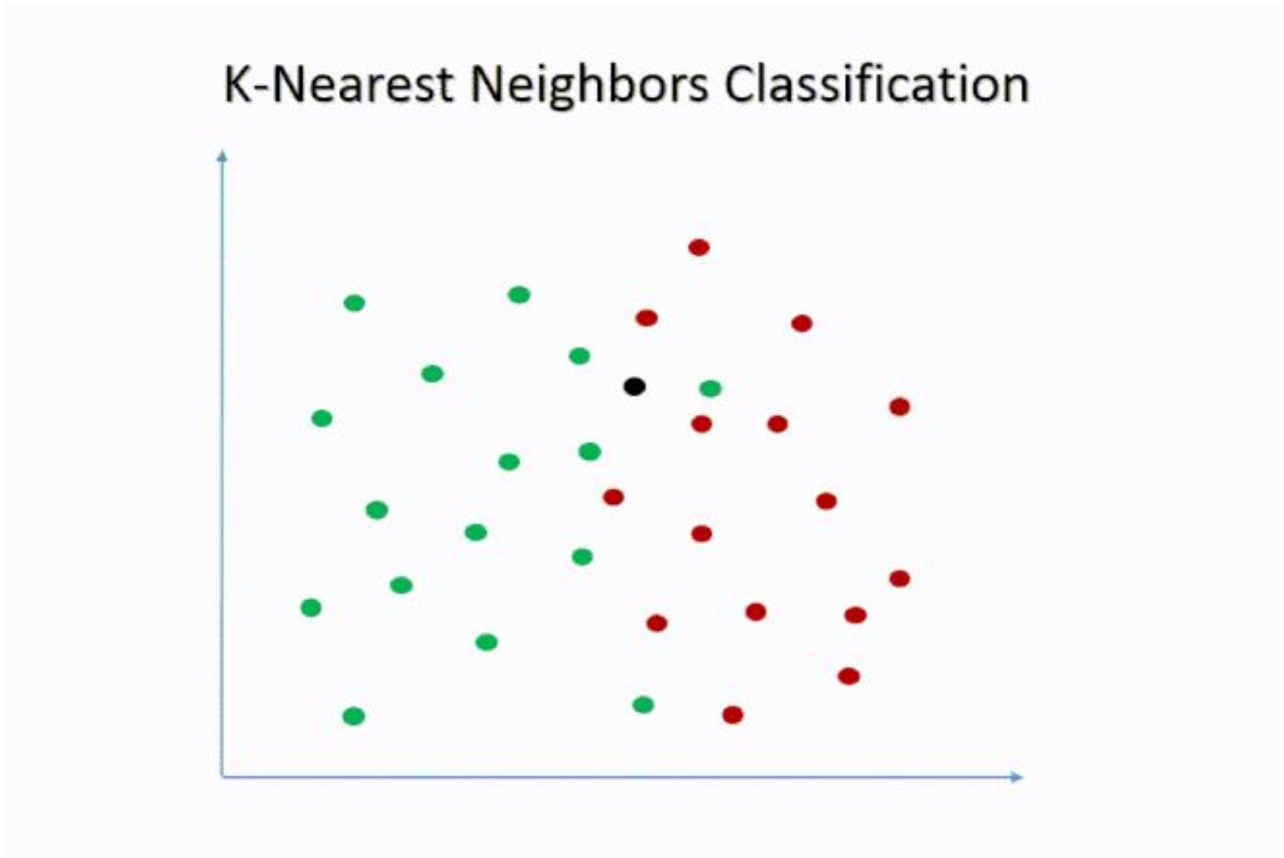
- No **optimization** involved
- Comparable **performance** to Naive Bayes
- **Easy to understand**

Cons

- Sensitive to **class imbalance**
- Need to store **entire training dataset for prediction**



Good visual of kNN from https://knn-notebook.netlify.app/



Let’s review the following gif to see the process that kNN takes to classify a new test observation.

Classification Challenges

Classification Challenges

In machine learning, we will consistently deal with a set of issues that arise throughout all prediction tasks:

- **High-dimensionality** (aka a lot of predictors) leads to model overfit
- **Class under-representation** leads to poor model specificity.

Let's identify what these issues look like before prescribing a set of fixes.

Choosing a Distance Metric

“*The better [the] metric reflects label similarity, the better the classified will be*”

We’ve established two different types of distance metrics. **Which one should we choose?**

Choosing a Distance Metric

The annoying answer: “*Well, try out both and see what happens*.”

The more descriptive answer:

- With **relatively few dimensions** (aka features/predictors) the performance of both euclidean and manhattan are comparable.
- However, in **highly dimensional spaces** (100, 1000, 10,000 dimensions) points become uniformly distant from each other.

This is known as…

The Curse of Dimensionality

THE CURSE (...of dimensionality)

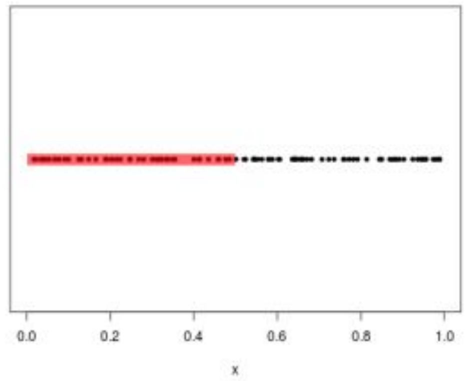
The entire point of kNN is to recognize *distances between points*, right?

Well **it turns out that as you increase your dimensions** (the # of predictors), **the distance between your points become uniform!**

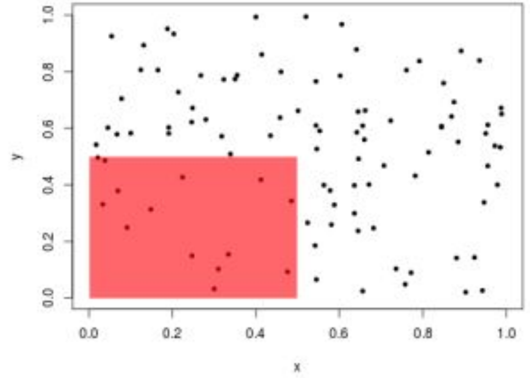
This leads to highly biased models (aka model overfit).

Basically it becomes harder to distinguish which class of data a new test sample belongs to.

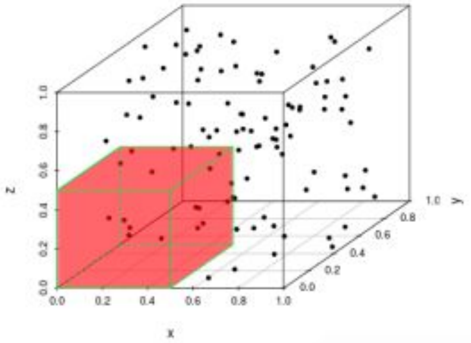
1-D: 42% of data captured.



2-D: 14% of data captured.

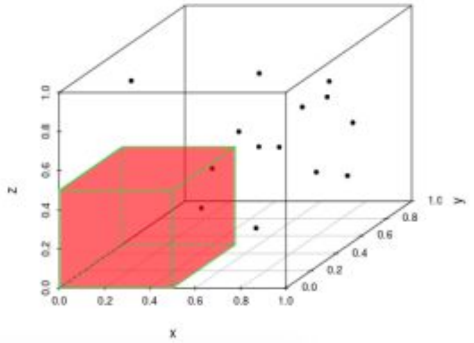


3-D: 7% of data captured.

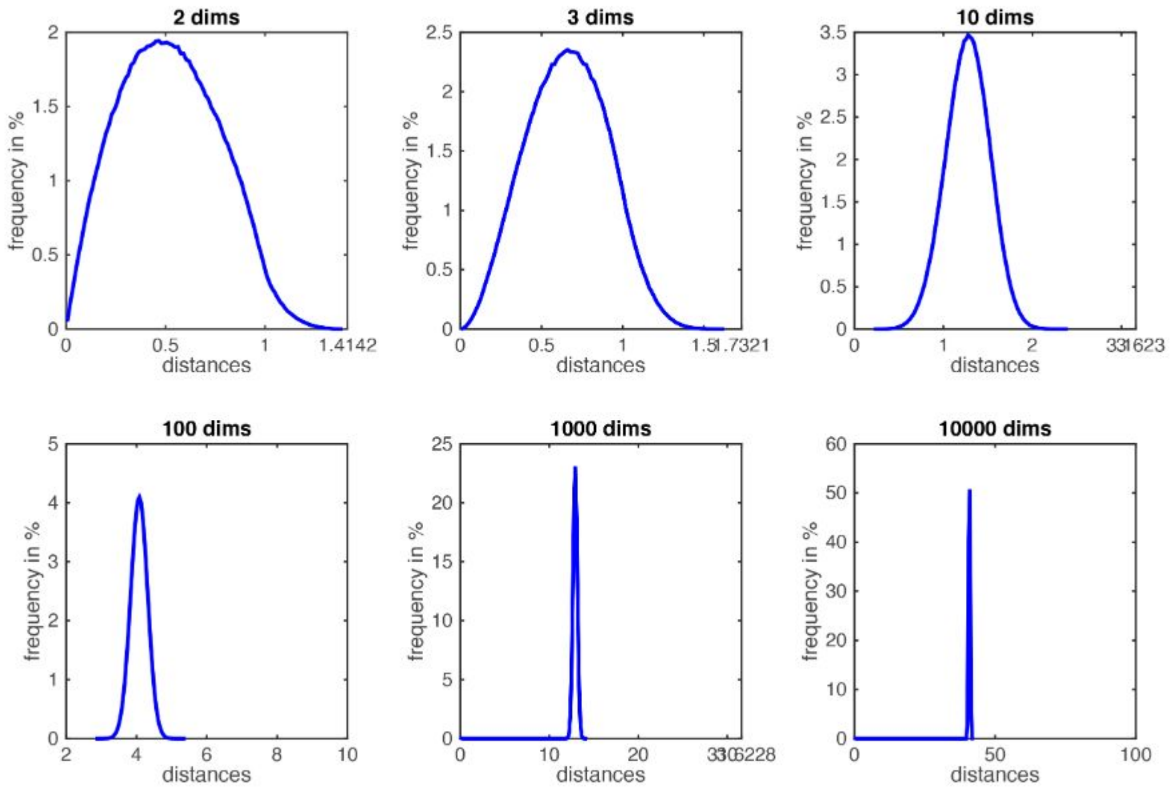


4-D: 3% of data captured.

t = 0



What do you notice about the **sparsity** of our data as we increase dimensions?



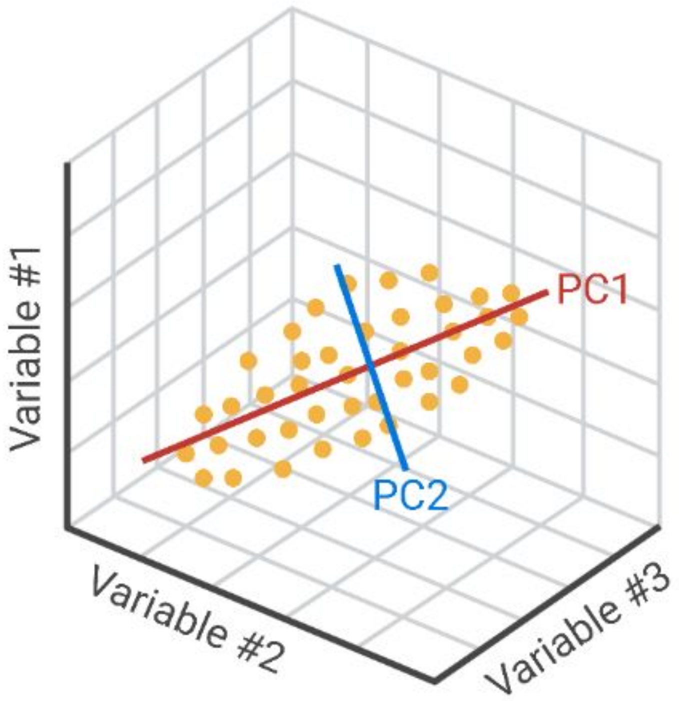
Notice how the frequency of distances converges on a specific value as we increase dimensions:
https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html

The Curse of Dimensionality

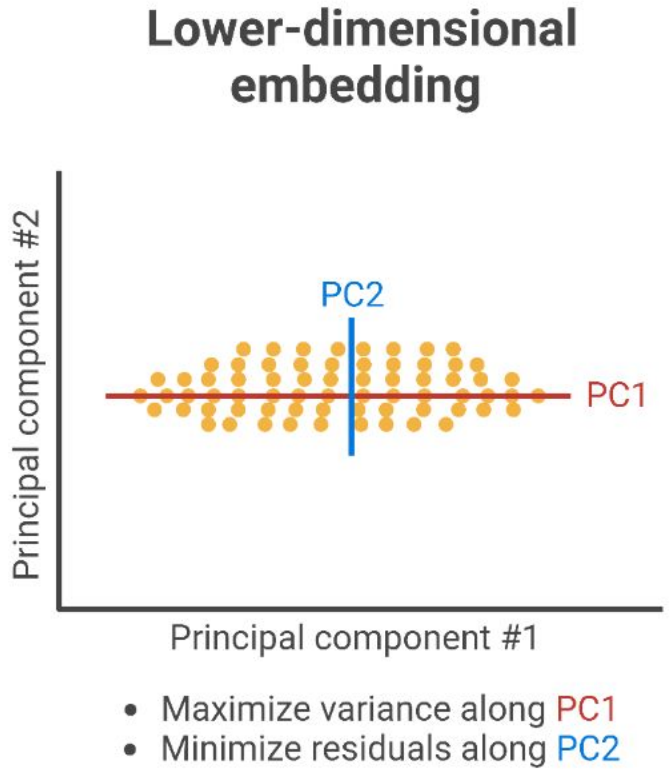
While all types of distances (there are more than 2) suffer from ***THE CURSE***, **it is the most pronounced in euclidean distances**.

However there are solutions:

- Use more data (*dubious advice*)
- Use the **manhattan distance**
- Use **another classification algorithm**
- Reduce your dimensions (we will go over this)
    - Blessing of non-uniformity: highly dimensional datasets are concentrated near lower-dimension manifolds



Lower dimensional manifold in a nutshell: “*Wow this looks really complex…*”


Lower-dimensional embedding
PC2
PC1
Principal component #2
Principal component #1
Maximize variance along PC1
Minimize residuals along PC2

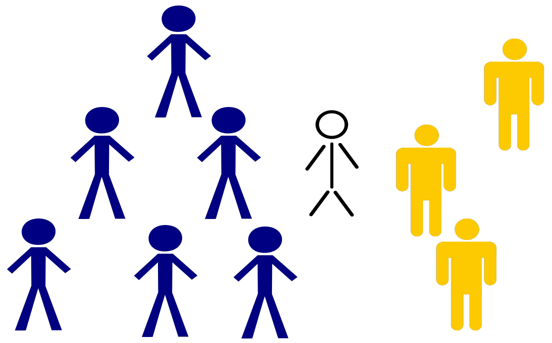
"Oh wait, I can map these values to a 2D space. Nice!" We will apply this type of dimensionality reduction algorithms in the upcoming weeks.

Handling Data Imbalance

*Keep in mind that this problem is not limited to kNN, but could be an issue for any other classifier!

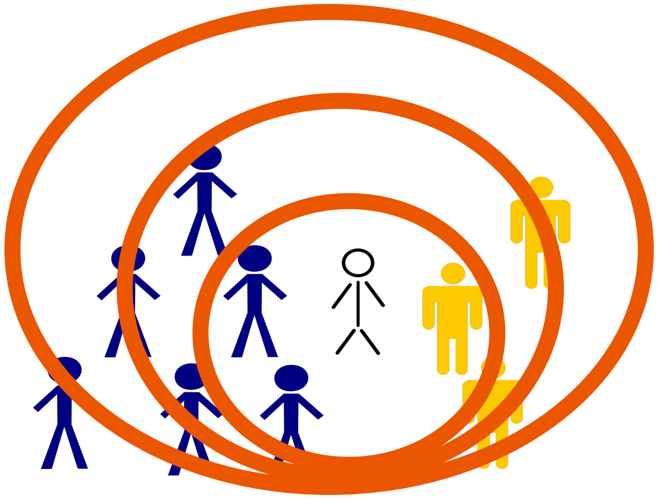
- Logistic
- Bayes
- Forest
- etc

Remember our industry event example. What happened when there were **too many** cyber fellows???



Handling Data Imbalance

Regardless of the **k** we used, we always classified this new fellow as a cyber fellow! **Unfair**!



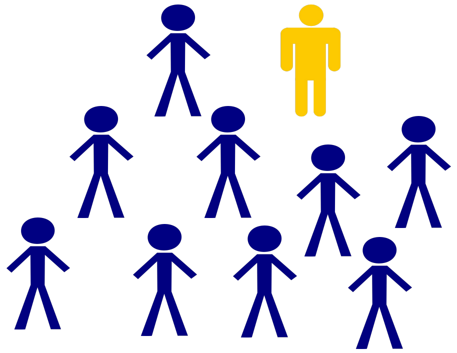
Handling Data Imbalance

$$\frac{\textit{True Positives + True Negatives}}{\textit{True Positives + True Negatives + False Positives + False Negatives}}$$

This leads to what we call the **accuracy paradox** (null accuracy).

Let's say you go to another industry event and you see 9 cyber fellows and 1 data fellow.

You train a kNN algorithm and it gives you a 90% accuracy rate. Wow! **This must be a good model, no?**

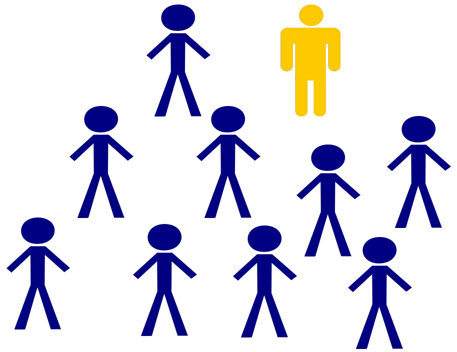


Handling Data Imbalance

Well, since we know the kNN algo is quite susceptible to class imbalance, it will essentially classify each fellow as a cyber fellow.

True Positives = 9
False Positives = 1
True Negatives = 0
False Negatives = 0

$$\frac{\textit{True Positives} + \textit{True Negatives}}{\textit{True Positives} + \textit{True Negatives} + \textit{False Positives} + \textit{False Negatives}}$$



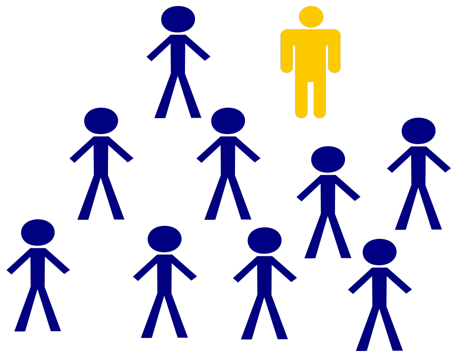
Handling Data Imbalance

$$\frac{9+0}{9+1+0+0} \rightarrow 90\%$$

This model's accuracy is simply a reflection of the distribution of cyber fellows.

This is why it's important to look at metrics such as **specificity**

$$\frac{\textit{True Negatives}}{\textit{True Negatives} + \textit{False Positives}}$$



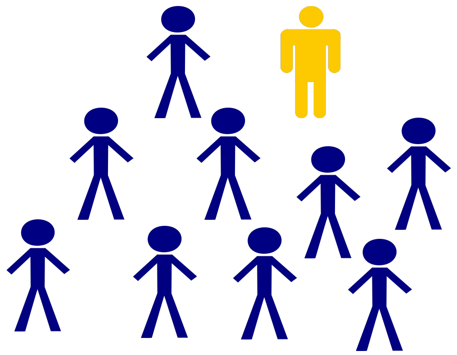
Handling Data Imbalance

$$\frac{9+0}{9+1+0+0} \rightarrow 90\%$$

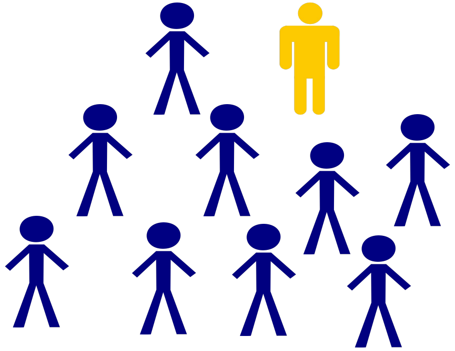
This model's accuracy is simply a reflection of the distribution of cyber fellows.

This is why it's important to look at metrics such as **specificity**

$$\frac{0}{0+1} \rightarrow 0\%$$



Handling Data Imbalance



To handle this imbalance we can do:

- **Over-sampling**: *compensate for imbalance by duplicating random (with replacement) minority-class samples*

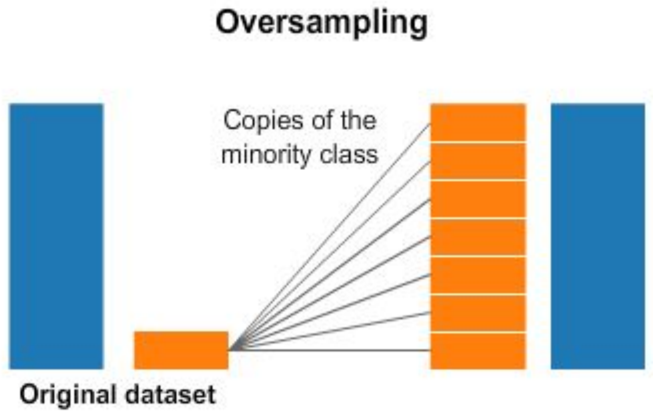
- **Under-sampling**: *compensate for imbalance by removing random majority-class samples*

- **SMOTE**: *Synthetic Minority Over-sampling Technique*

- **A different classifier**: *1-NN, Neural Nets, a domain expert*

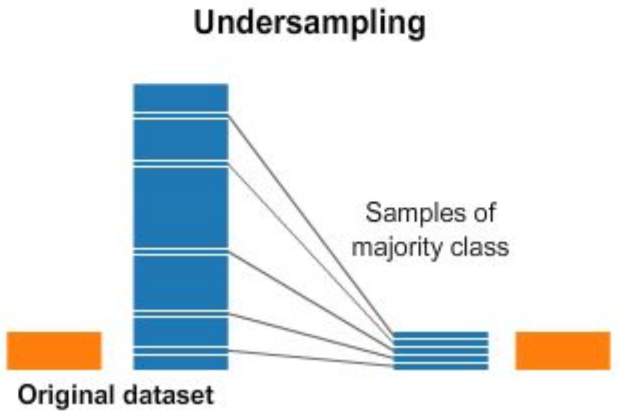
Over-Sampling Vs Under-Sampling

These two concepts are quite simple, we either **duplicate random underrepresented samples** until we have **balanced classes**.



Over-Sampling Vs Under-Sampling

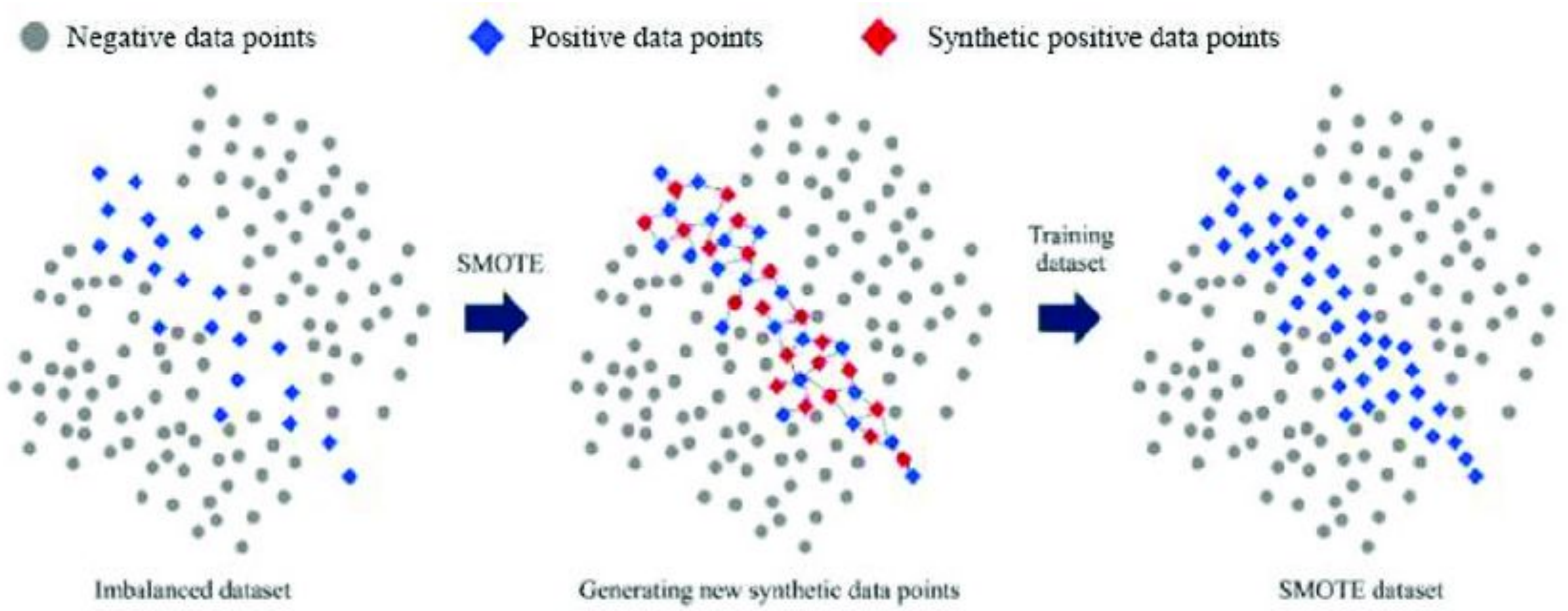
Or we **randomly remove overrepresented samples** until we have **balanced classes**. Keep in mind this removes training data (**information loss**)



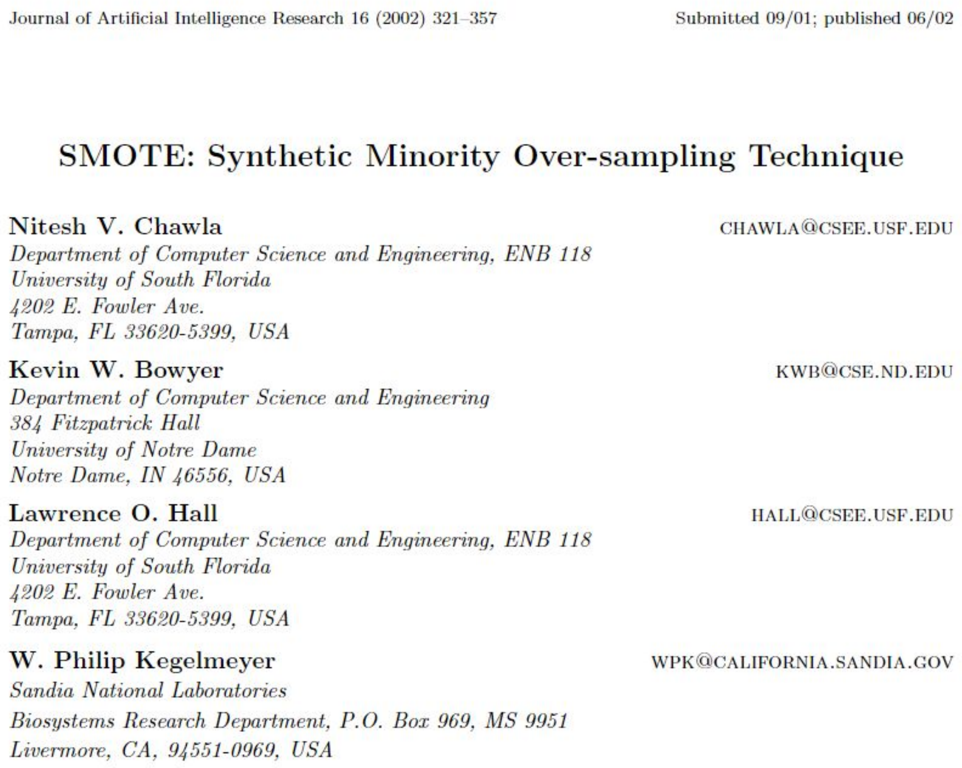
SMOTE

Synthetic Minority Over-sampling Technique
This technique entails creating new *synthetic minority classes* via a somewhat randomized process.

1. Choose a random set of the minority class
2. Find k-nearest-neighbors for each sample of the same class
3. Calculate a new synthetic class between this sample and all of its neighbors
   a. Choose a random value between 0 & 1 to calculate where this newly random point will exist in the space between your data point and its neighbors



This allows us to “purposefully” oversample our dataset to improve the predictive qualities of our dataset.



Journal of Artificial Intelligence Research 16 (2002) 321–357 Submitted 09/01; published 06/02

SMOTE: Synthetic Minority Over-sampling Technique

Nitesh V. Chawla CHAWLA@CSEE.USF.EDU
Department of Computer Science and Engineering, ENB 118
University of South Florida
4202 E. Fowler Ave.
Tampa, FL 33620-5399, USA

Kevin W. Bowyer KWB@CSE.ND.EDU
Department of Computer Science and Engineering
384 Fitzpatrick Hall
University of Notre Dame
Notre Dame, IN 46556, USA

Lawrence O. Hall HALL@CSEE.USF.EDU
Department of Computer Science and Engineering, ENB 118
University of South Florida
4202 E. Fowler Ave.
Tampa, FL 33620-5399, USA

W. Philip Kegelmeyer WPK@CALIFORNIA.SANDIA.GOV
Sandia National Laboratories
Biosystems Research Department, P.O. Box 969, MS 9951
Livermore, CA, 94551-0969, USA

The OG paper that proposed SMOTE: https://arxiv.org/pdf/1106.1813.pdf

Handling Data Imbalance

This is especially useful when trying to detect some real relatively rare class in your dataset:

- **Actual fraud**
- **A rare disease**
- **A 16-seeded team beating a 1-seeded team**

Classification Challenges

Machine learning is a field ripe for innovation & novelty. Prioritize experimentation above all.

- **High-dimensionality** (aka a lot of predictors) leads to model overfit.
  - Apply **dimensionality reduction algorithms** such as PCA to extract data on lower-dimensional manifolds.
- **Class under-representation** leads to poor model specificity.
  - Apply an over-sampling techniques such as **SMOTE** to improve predictions.

End of Class Announcements

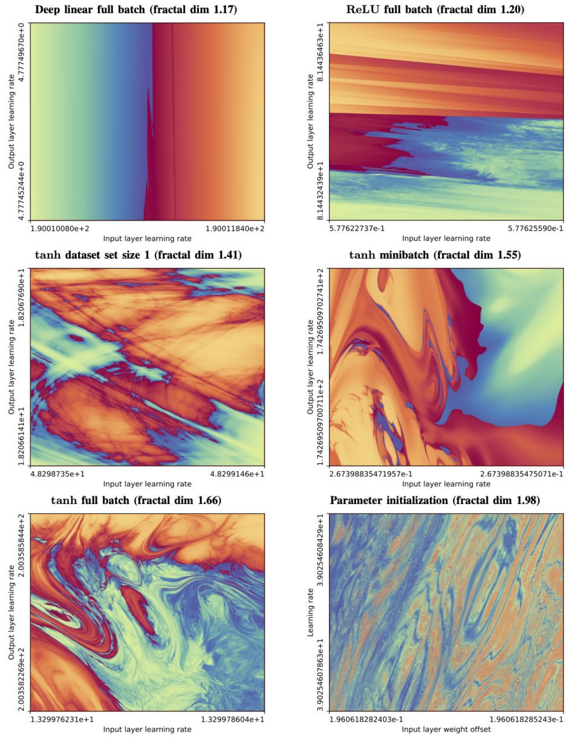
Tomorrow

K-Nearest Neighbors

- Who is my neighbor?
- What is a manhattan distance?

kNN Hyperparameters

- What happens if we increase/decrease k?
- Where does variance/bias exist in kNN?



Neural network training makes beautiful fractals