# Introduction to Adversarial Machine Learning

CMSC 334, Prof Szajda

# Quick ML Recap/Tutorial

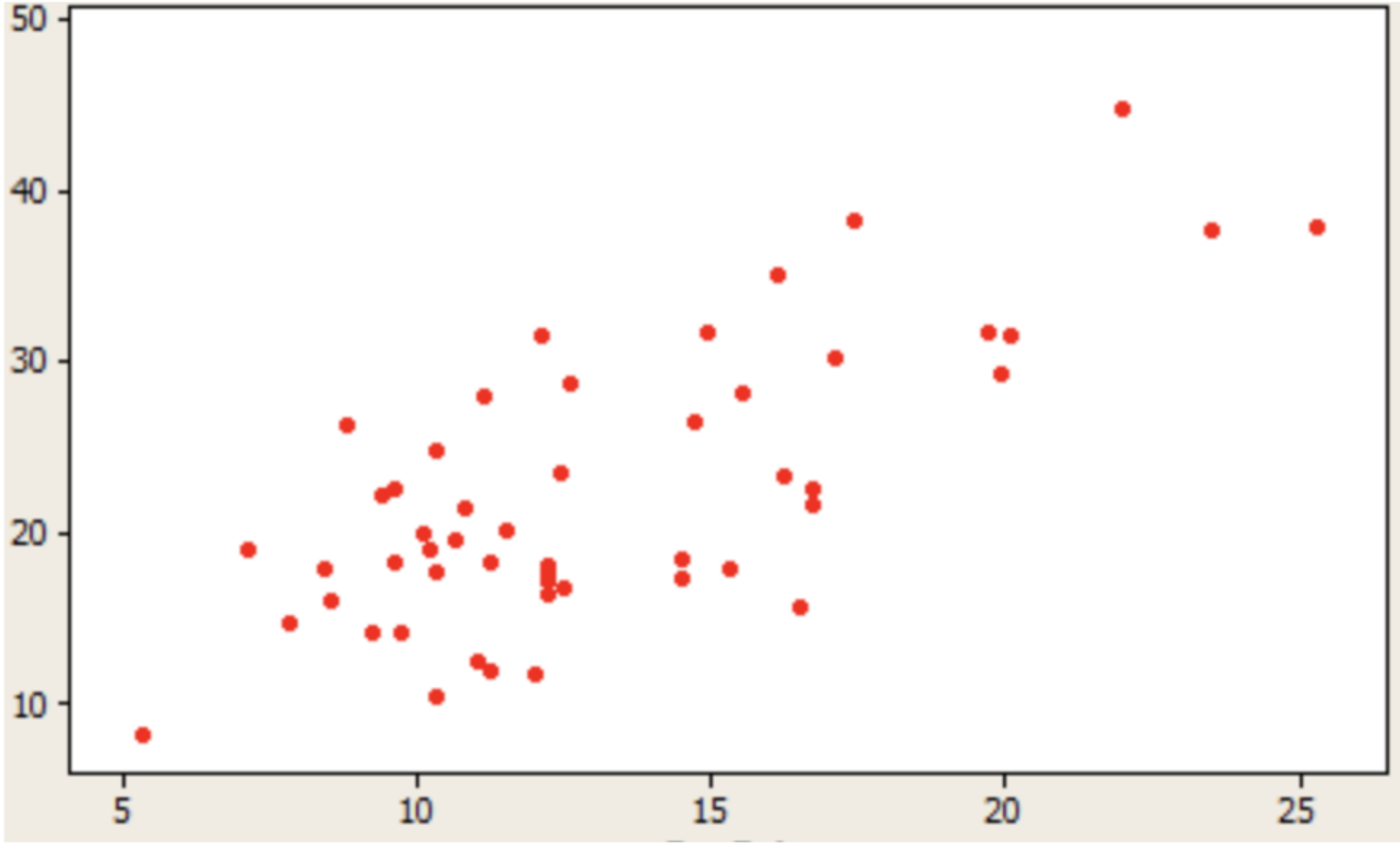
- Machine learning models
- Choosing parameters
- Decision boundaries
- Neural Networks
- Adversarial machine learning

# Why Machine Learning?

- From an earlier version of these slides
  - I assume I don't have to tell you why ML is important…
    - But I found it amusing that I did have to, at one time
    - And once again
      - I find myself wondering how small I can make this…
      - Apparently, just this small.  Hey, you have to amuse yourself somehow when making slides

# Let’s Consider Plain old Linear Regression

- We have some data: a collection of (x,y) pairs
- We’d like to fit a line to this data, so that we can capture the relationship between x and y.
  - And potentially use it to predict values of y when given x



# Simple Linear Regression

- Many of you have likely seen this at some point
- Assume the (x,y) pairs are given by

$$\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_n, y_n)\}$$

- Lines have the form

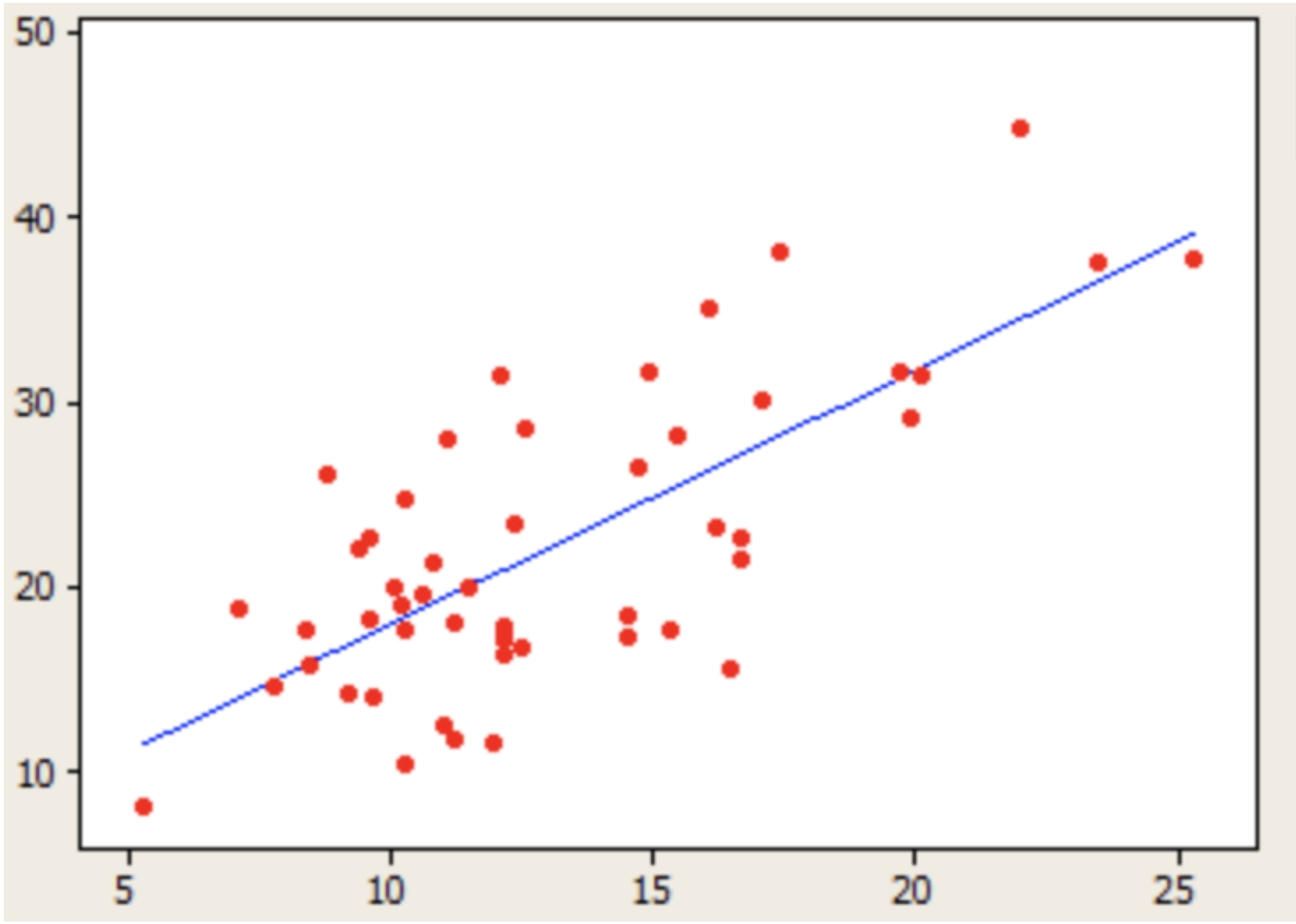
$$y = mx + b$$

- So finding the “best” line means finding the values of m and b that correspond to that best line
- Let’s denote these best values by $\hat{m}$ and $\hat{b}$

# Simple Linear Regression

- If you have seen this, then you know that the solutions are

$$\hat{m} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \qquad \hat{b} = \bar{y} - \hat{m}\bar{x}$$



# Simple Linear Regression

- Some questions:
  - What makes this line the “best” line?
  - Given that one agrees with he definition of “best”,  where did the equations for the best line come from?
  - What assumptions are we making about the relationship between x and y when we attempt to fit a line to the data?

# Some Answers

- What makes this the “best” line?
- Every choice of the pair (m,b) corresponds to a line. Some lines would, if graphed, be clearly a bad fit for the data. How can we quantify this?
- Let’s look at the point ($x_i$, $y_i$). We know that $y_i$ is the value that should correspond to input $x_i$. Given a specific choice of m and b, the resulting line, when fed $x_i$ will output

$$\hat{y}_i = mx_i + b$$

- The difference between what the line outputs, and what we know the line *should* output is a measure of error

# Some Answers

- What makes this the “best” line?
- That error we call “cost”. There are many ways to measure cost. The way regression does it is this

$$C(m,b) = \sum_{i=1}^{n} [y_i - (mx_i + b)]^2$$

- Every value of m and b is associated with a cost. The regression line is the one that minimizes this particular cost function.
- How is this solved? Lots of ways. Multivariable calculus is among the easiest. We won’t go through the details…

# How do you optimize multiple variable functions?

- The analytic method can be impractical if, especially if you have billions of variables
- Gradient descent works in that case, but it can be slow
  - Each iteration requires computing billions of partial derivatives
  - So “training” a large network like that can take quite a bit of time (days/weeks/more).
- We’ll get back to this notion a bit later. For now, understand that it’s the primary reason that deep neural networks did not exist until relatively recently.

# What does this have to do with machine learning?

- We had a relationship between variables that we wanted to understand  (how does x predict y)
- We chose a particular type of model (linear regression)
- That “type of model” was actually a parametrized family of functions
  - For each choice of m and b we get a line (a “linear model”). Different parameters give different lines, and thus different results.  There is a correspondences between parameters values and models
- We searched through the space of linear models and found the one that appeared to be “the best”

# What does this have to do with machine learning?

- Put another way, we “trained” the model
  - We were given data, so called “training data”
  - We used the training data to determine the “best” values for m and b
  - If we were to learn more data values, we could repeat the process, likely resulting in different m and b values (and hopefully generating better predictions, but that is a story for another time)

# What does this have to do with machine learning?

- This is an example of *supervised learning*
  - Training data, with correct output values (*target values*) included inform the best choice of parameters for our model
  - There is a single input variable (*feature*) for simple linear regression
    - But in general a vector of inputs, the *features*
  - Once we find the "best" model, we can use it to make predictions about the target value given a feature vector
- There is an implicit assumption here. Can you identify it?

# Classifiers

- The regression model we discussed is designed to output real numbers
- But far more machine learning models do classification
  - Given an image, is it a dog, cat, person, school bus, stop sign, etc
  - Given an image of a person in this room, who is it?
  - Given an audio sample…
    - Identify the speaker (from a specific set)
    - Determine the word, phoneme, or grapheme being spoken
  - Given a sample of computer code, determine whether it is malicious

# Classifiers

- It’s possible to build a simple classifier out of a model that outputs a single number
  - Let’s assume there are only two classes, say malicious code versus benign code
  - Assign one of those classes the value y = 1 (malicious), and the other the value y = 0 (benign).
  - So each feature vector in our training set will have a 0 or 1 as the last component
  - Build a linear regression model
  - For predicting class, use regression function.  If y > 1/2, call it malicious.  Else call it benign.

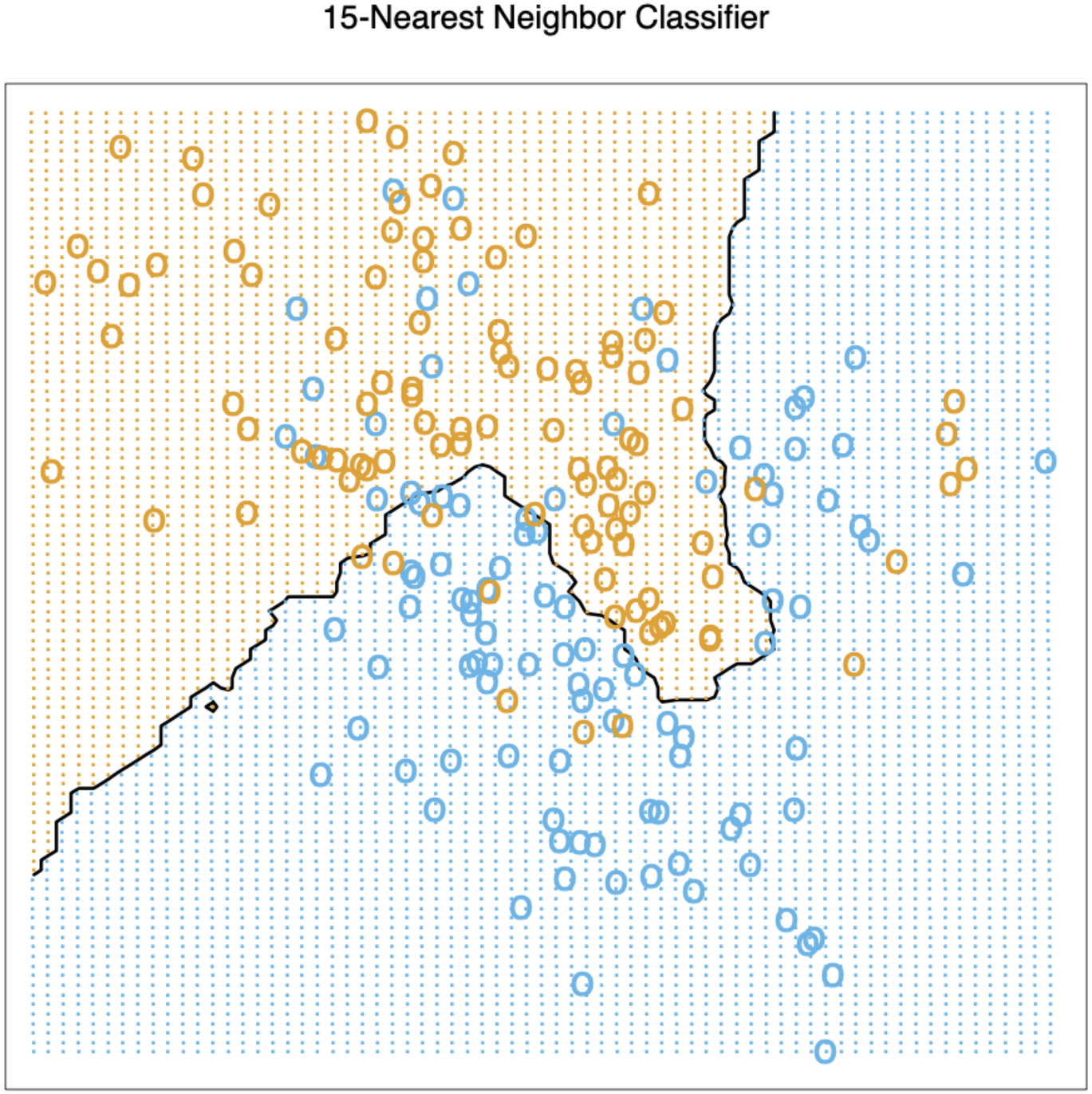
# Classifiers

- It's possible to build a simple classifier out of a regression model
  - For predicting class, use regression function. If y > 1/2, call it malicious. Else call it benign.
- In some contexts (not most), this will do fine
- But there's typically a better way
  - We won't go into the details of that (largely for time reasons)

# Decision Boundary

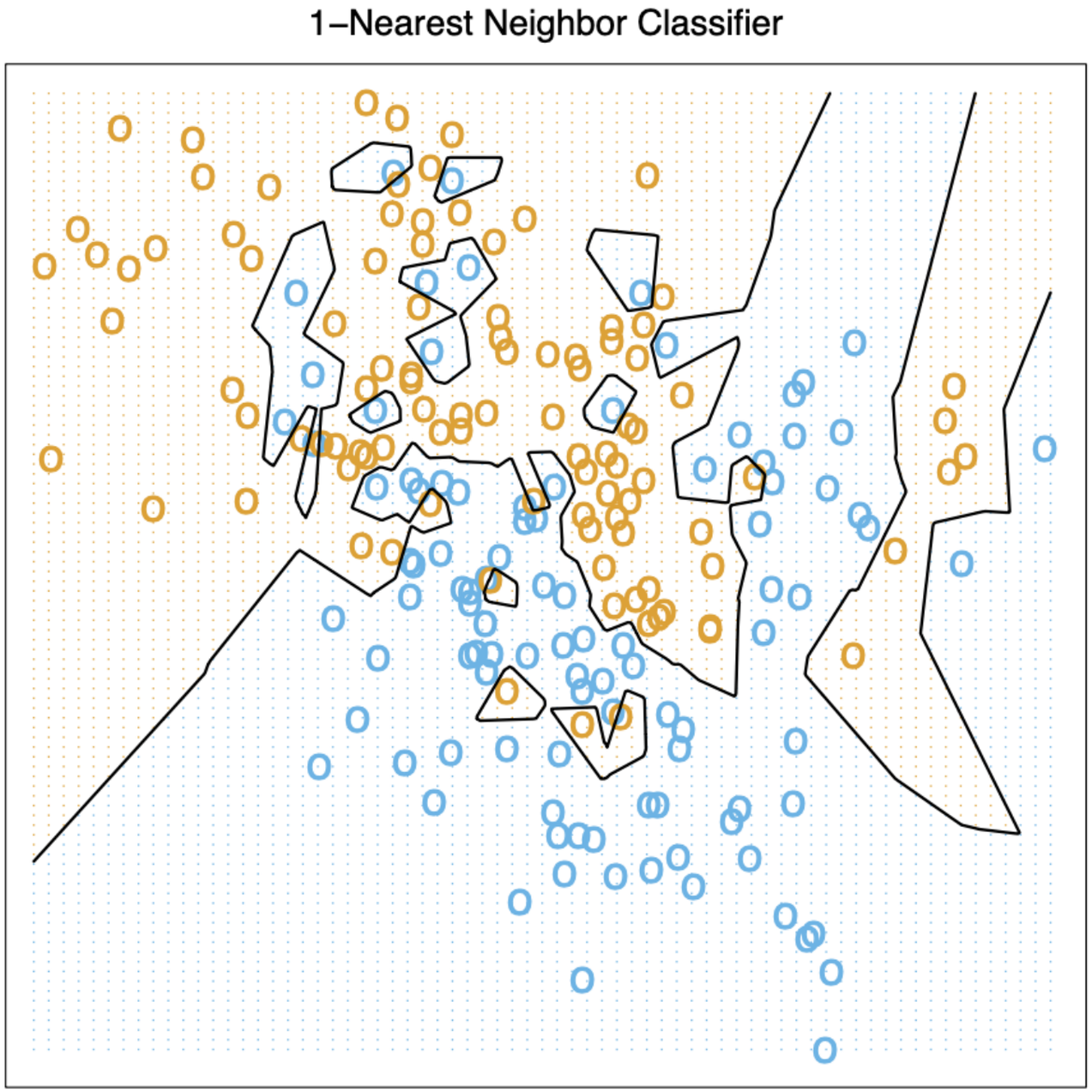
- BUT, there is an important notion that we need to discuss: the decision boundary
- The setting:  We have a binary classifier (only two classes):
  - Two classes:  Malicious code (orange), Benign code (blue)
  - We are given training data consisting of ordered (x,y) pairs, along with their correct class
- We then “train” our classifier: find the values of parameters that in some sense “best” fit the training data
- Once the model has been trained, we can use it on new data ( (x,y) pairs ).  Some of that new data will be classified as Malicious, some as Benign.  That divides the plane into two sets.  Boundary between them is the *decision boundary*

# Decision Boundary



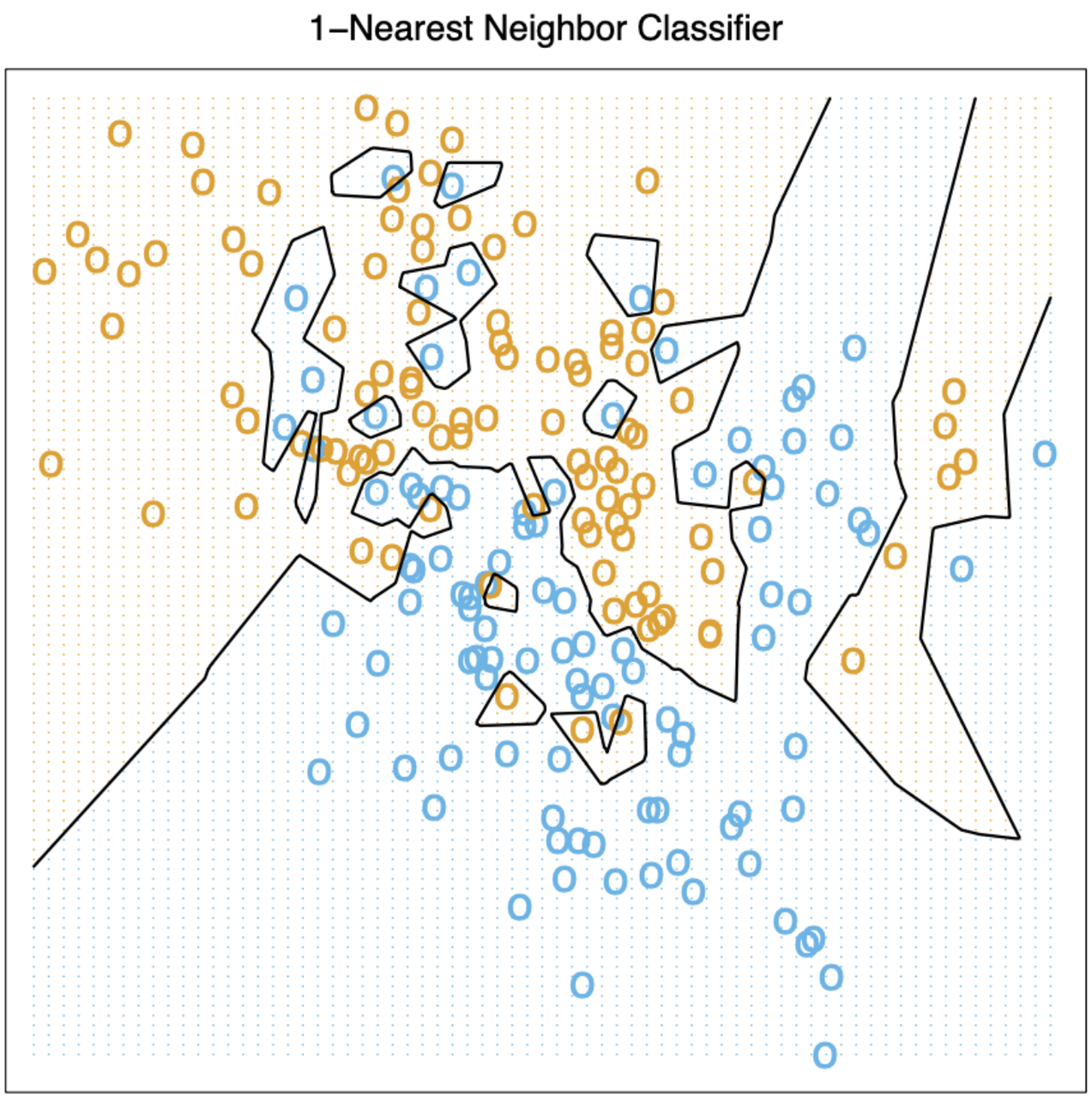

Note that the classifier here isn't perfect on the training data.
And we may not want it to be!

# Decision Boundary




And this is why: this classifier is perfect on the training data! But that's a problem! (Why?)

# Decision Boundary




This also shows something else: decision boundaries can be extremely complex (even in two dimensions). This has ramifications that we'll get to

# Classifiers

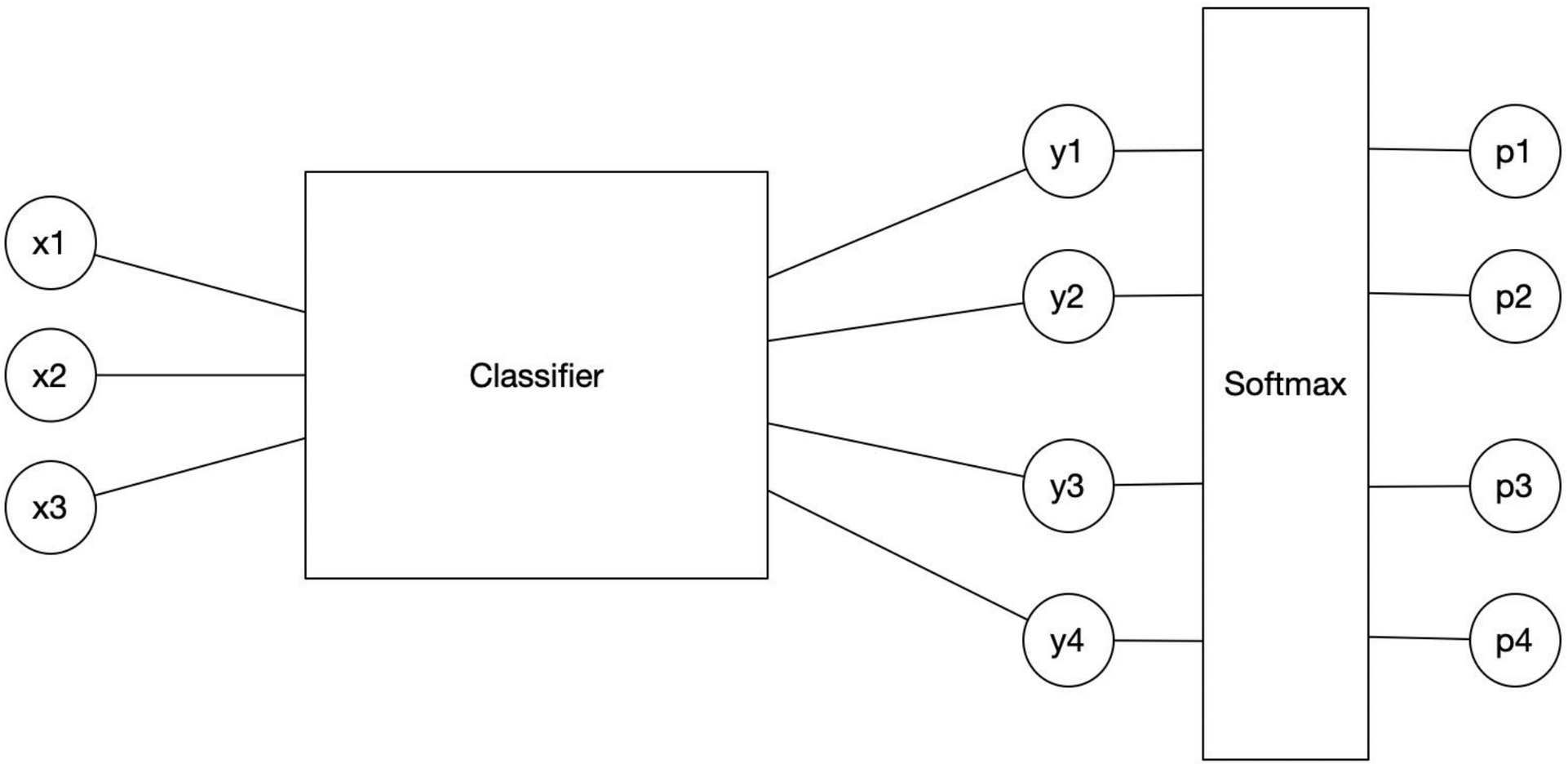
- It’s not usually the case that a classifier only identifies a single class
- The general idea is that instead of one output for the model, there are multiple outputs, one for each class
  - Of course this presents a problem: remember that our training data is of the form (features, output). So how does one handle multiple outputs? Answer: *one-hot encoding*
- The idea: the “single” output will be a tuple (vector)
  - Each class is assigned to a specific component (index)
  - The output indicates a specific class if the value of the component at the class’s assigned index is 1, but the values at other indices are 0

# One-Hot Encoding

| Colors in our data | Assignment of a number to each value | One-hot encoding of each color |
|---|---|---|
| red | red → 0 | red → [**1**, 0, 0, 0, 0, 0, 0, 0] |
| yellow | yellow → 1 | yellow → [0, **1**, 0, 0, 0, 0, 0, 0] |
| blue | blue → 2 | blue → [0, 0, **1**, 0, 0, 0, 0, 0] |
| green | green → 3 | green → [0, 0, 0, **1**, 0, 0, 0, 0] |
| orange | orange → 4 | orange → [0, 0, 0, 0, **1**, 0, 0, 0] |
| brown | brown → 5 | brown → [0, 0, 0, 0, 0, **1**, 0, 0] |
| purple | purple → 6 | purple → [0, 0, 0, 0, 0, 0, **1**, 0] |
| black | black → 7 | black → [0, 0, 0, 0, 0, 0, 0, **1**] |
| Class | Index (start at 0) | Vector |

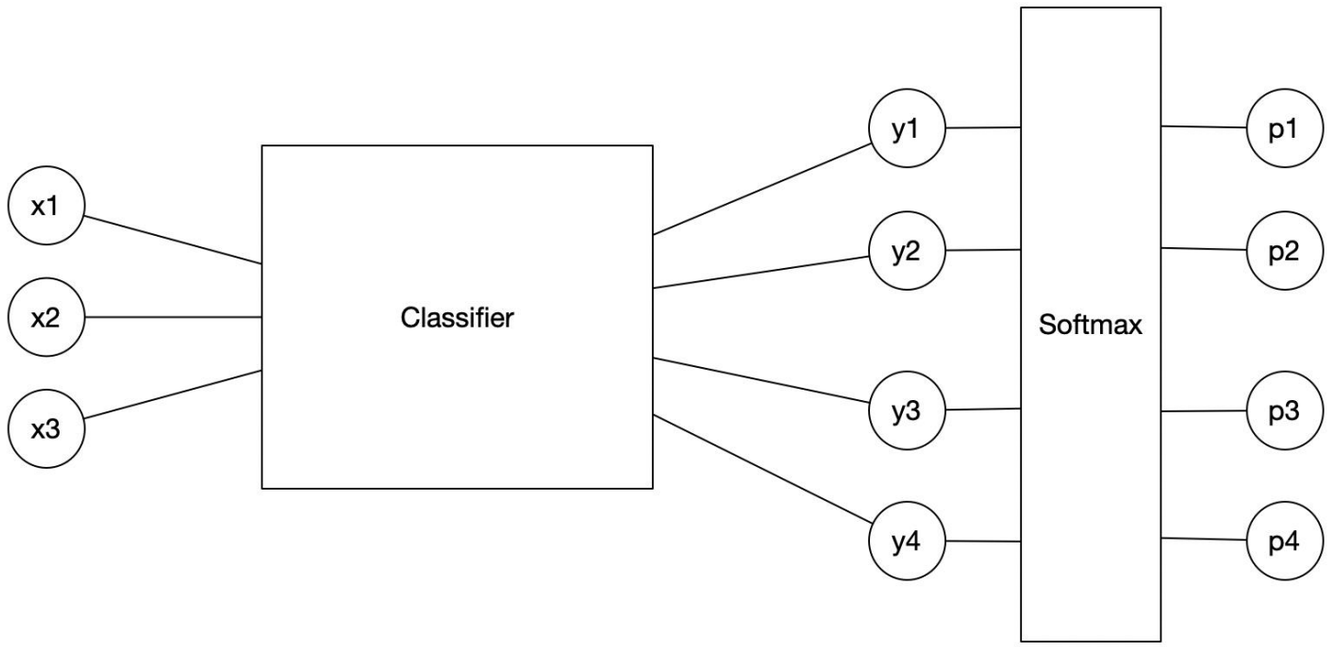
# Classifiers

- Given one-hot encoding, the general “structure” of a classifier looks something like this:
  - The y values make up the output vector

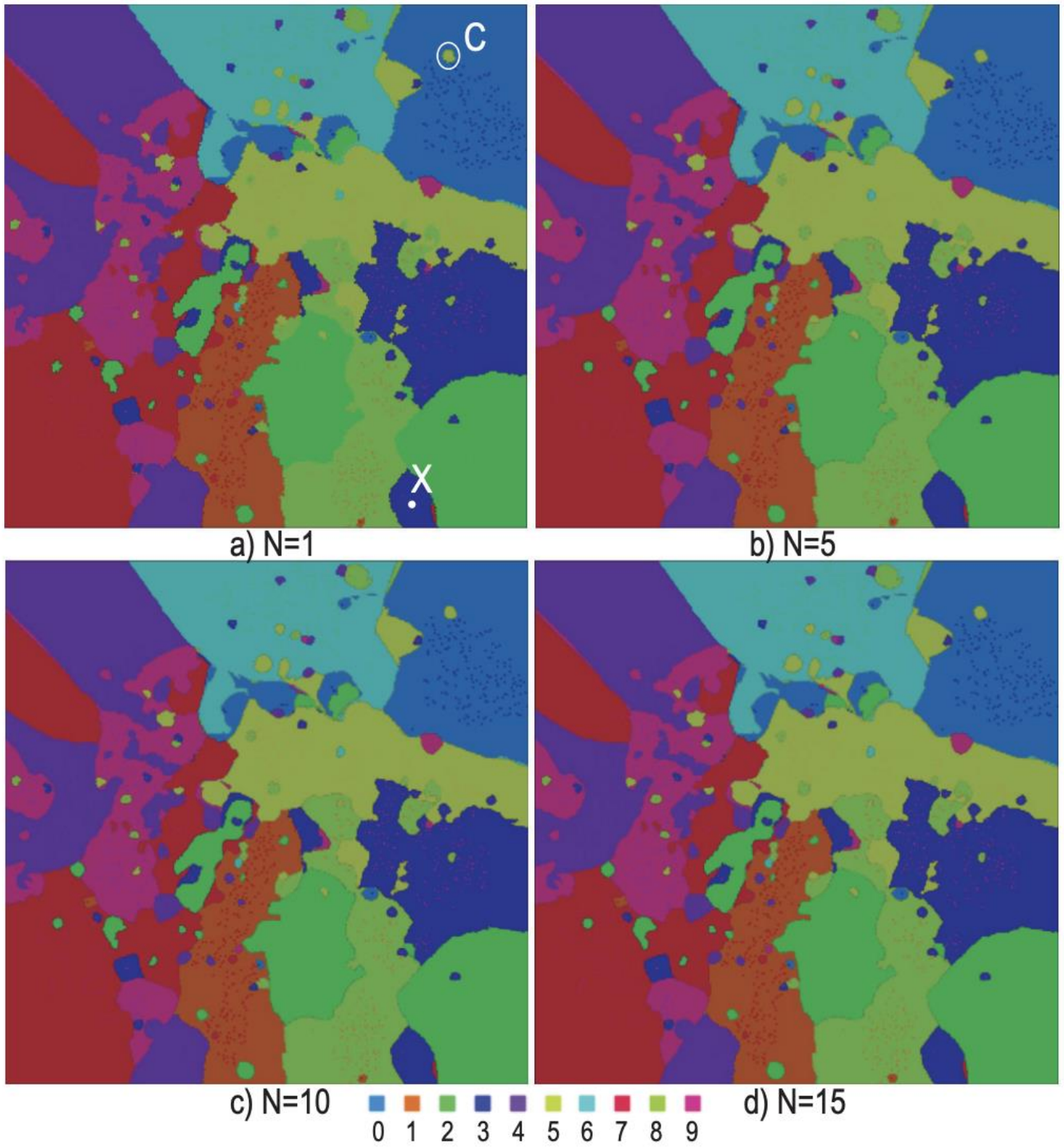


# Decision Boundaries?

- We saw some decision boundaries for a binary classifier
- What might they look like if you have multiple classes?



# Decision Boundary:  Multiclass example




Note that this is only in two dimensions. What if your feature vector has a million dimensions! Think that's not realistic? Think again!

# So now: Implications for Decision Boundaries

- Transparency: Though the theory is relatively clear, as in, you are finding parameters that in some way give the best results for your network, you have no real way of knowing why the network is making the choice it is
- And a curve ball:  Even if you use the exact same training data, and start with the exact same random assignments to all of the weights, you can end up with different end weights!
  - Which means two identical networks trained exactly the same way can potentially classify the same input differently!
  - Thanks to GPU multithreading

# Adversarial Machine Learning

- All of what we have discussed builds up to the primary issue for us: adversarial examples
- First, a question: who cares?
- Well, think about what these things are used for
  - How about a self driving car, which uses ML to identify and understand its surroundings
  - Suppose one could fool such a model simply by placing a few pieces of tape on a stop sign?

# Adversarial Machine Learning

- Well, think about what these things are used for
  - Suppose one could fool such a model simply by placing a few pieces of tape on a stop sign?


  - Well, it turns out you can. The ML model this was intended to fool interpreted this as a “Speed Limit 45 mph” sign

# Adversarial Machine Learning

- Well, think about what these things are used for
- Again, think about all of the ways in which ML models are used, and the damage one can do if the models are fooled
  - What can you do if you can fool siri or Amazon Alexa?
  - Or facial recognition software?
  - Or biometric authentication software?
  - Or any of the many other functions that ML models serve?

# Adversarial Machine Learning

- So now that we've seen the why, a few observations:
  - A general rule for machine learning: If a human expert can classify something, chances are you can build a ML classifier to do it
    - And often, if a human can't do it, neither can an ML classifier
  - But ML classifiers can identify patterns that a human can't, or generally wouldn't
    - Perhaps because we wouldn't even notice them
  - And even though some ML systems are built to model human perception, there remain differences in how humans and ML models "reason"
    - Which we might not (and generally do not) understand

# Adversarial Machine Learning

- So, the question: How can we fool ML classifiers?
- Can you think of anything you might do to get an ML model to perform poorly?
  - And what do you need to know or have access to in order to succeed at your task?
- Hint:  Think, at a high level, about how these things work

# Adversarial Machine Learning

- Can you think of anything you might do to get an ML model to perform poorly?
- Answer: Poison training data — insert a bunch of training examples that all have, say, a dark green pixel in the lower right hand corner of the image, and all have a fixed target class (say “duck”). Then regardless of image, if it has a dark green pixel in lower right corner, it’s likely to be classified as duck.
  - Might seem impractical. BUT, if you do something similar with computer code — put a specific string in every file, and label all those files as “benign”, then any code that has that string in it has a high probability of being classified as benign!

# Adversarial Machine Learning Attack Classes

- Two classes of errors:
  - *Targeted*: we fool the ML classifier so that it chooses the incorrect class *that we want it to*
  - *Untargeted:* we fool the ML classifier so that it chooses the incorrect class, but we don't care which incorrect class it chooses
  - Ex.  Automated speaker identification
    - Might want the ML model to identify speaker as a specific (incorrect) individual
  - Ex. Malware detection
    - Who cares why it says code is benign as long as it says it's benign!
      - Well, that's not entirely true…

# Adversarial Machine Learning Attack Classes


More on this later, if we have time

https://www.nytimes.com/2026/04/15/magazine/ai-black-box-interpretability-research.html?unlocked_article_code=1.bVA.7vwV.PSYoQJ1mbVP_&smid=url-share

# Adversarial Machine Learning Attack Classes

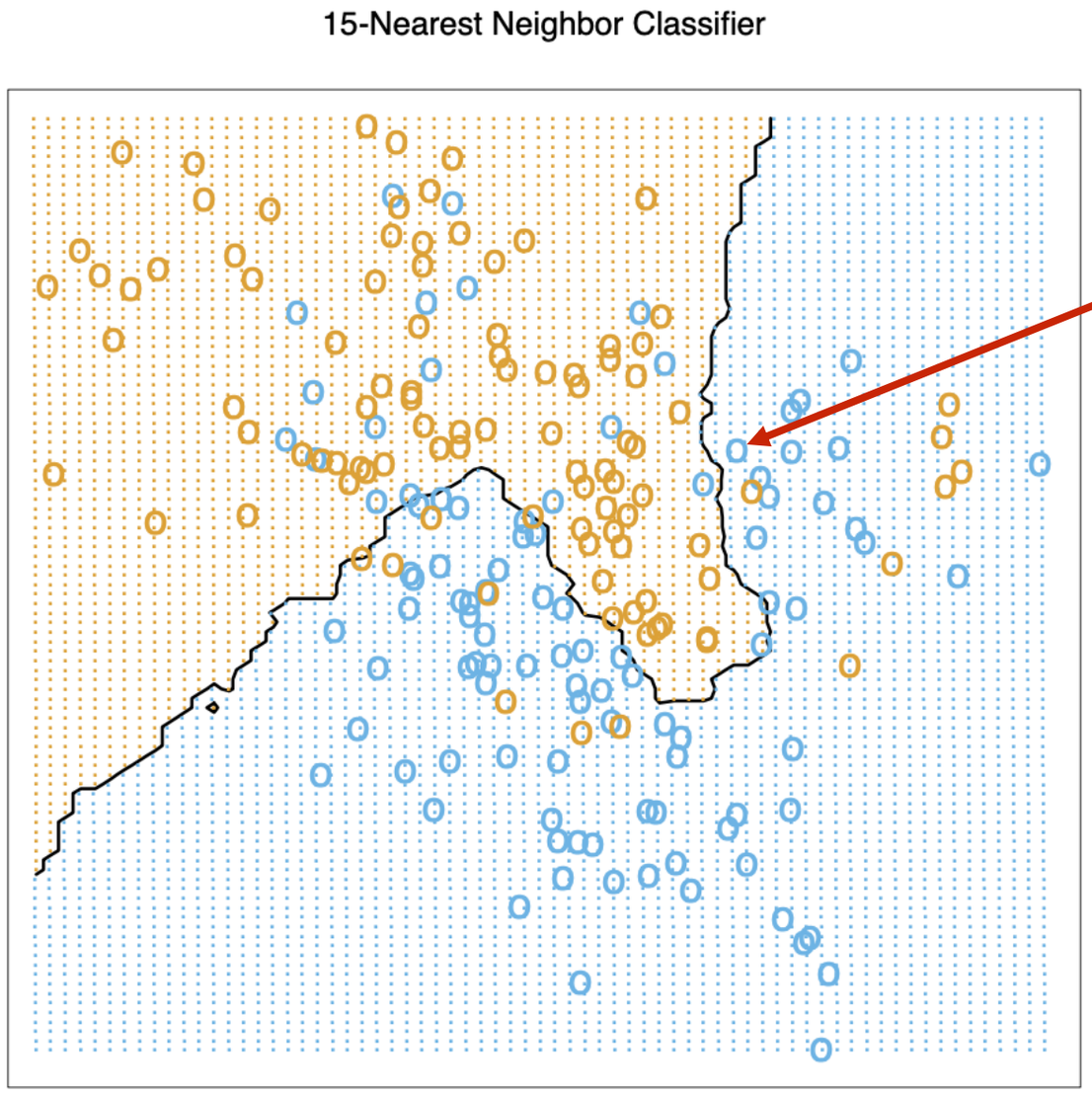
- *White box attack:* The adversary has access to all information about the model
  - Hyperparameters (what?) and all parameters
- *Black box attack*: The adversary has no access to any internal information or hyperparameters
  - They can use the model (feed it inputs and observe outputs), but that's it
  - The most realistic attack
- *Gray box attack*: The adversary has some information about structure of the model, but generally no information about parameter values or the like

# Adversarial Machine Learning Attack Classes

- Often one more consideration: similarity to original benign sample
- It's not hard to fool an ML classifier into thinking a picture of a school bus is actually an image of a duck if you modify the image so much that it effectively looks like a duck
- So the idea: with adversarial examples, we often want to modify the original benign example as little as possible
  - And generally only just enough so that the the example becomes misclassified
- In terms of mathematics, we want to *perturb* the original example just enough to nudge it over the decision boundary

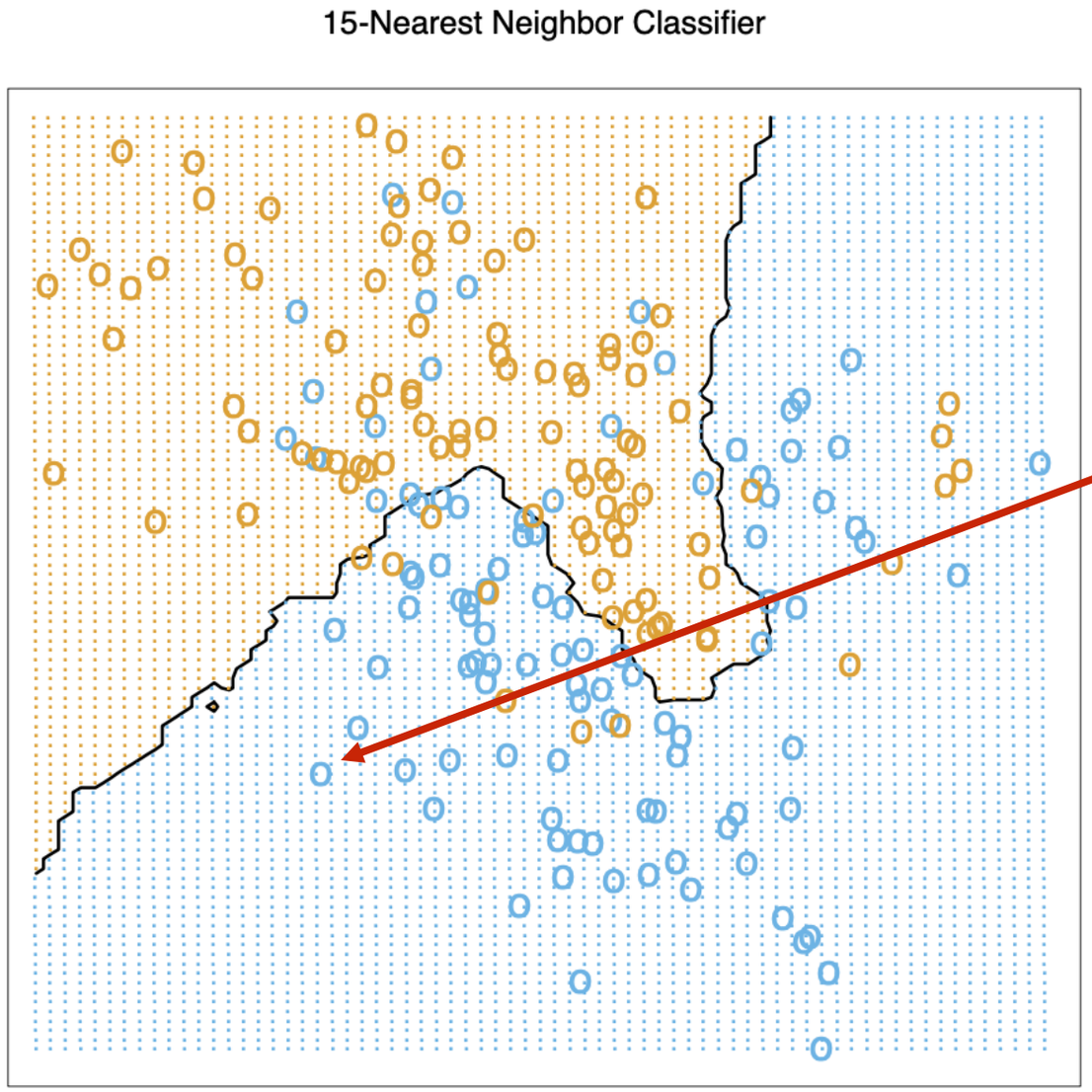
# A picture is worth a thousand words...

- Often one more consideration: similarity to original benign sample



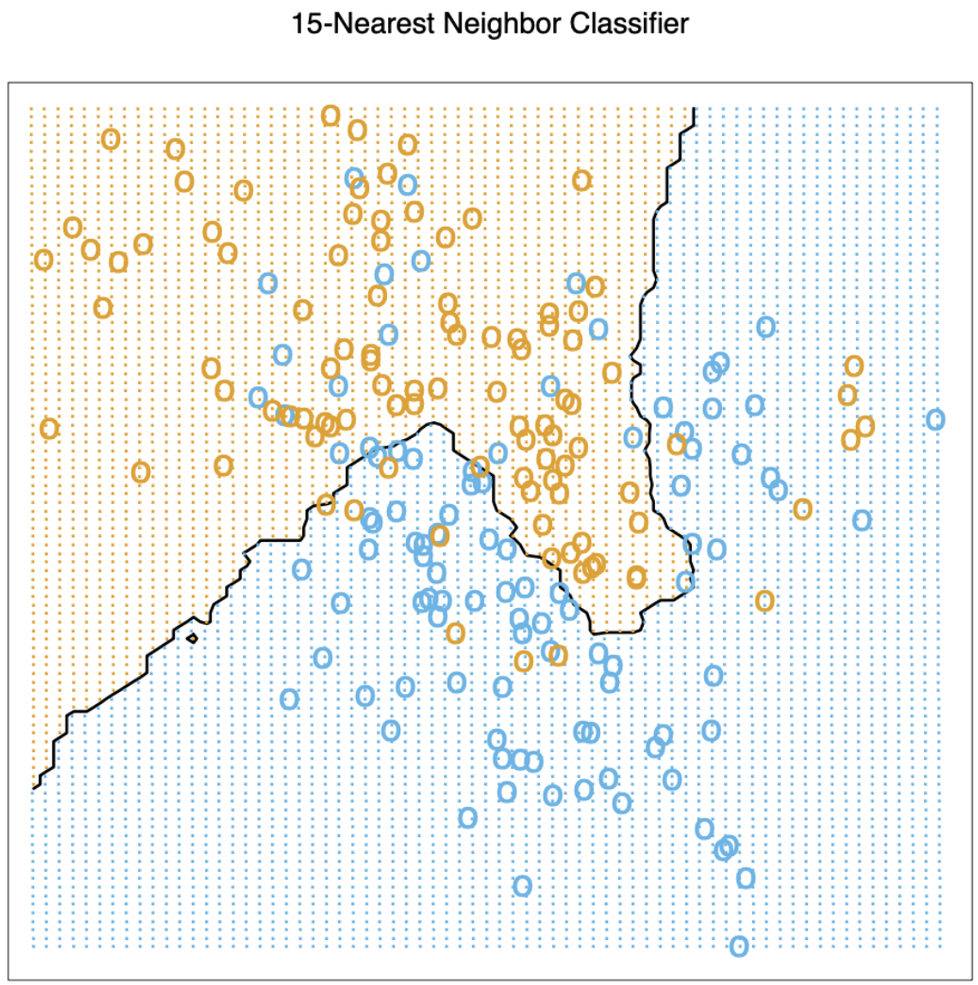
# A picture is worth a thousand words...

- Often one more consideration: similarity to original benign sample



# A picture is worth a thousand words…

- If this was what real decision boundaries looked like, you might think that in general creating adversarial examples is difficult



# A picture is worth a thousand words...

- But we know that in general this is not what real decision boundaries look like.  The look more like this:

- And remember, this is only *two* dimensions

# A picture is worth a thousand words...

data samples. Densely-packed white points effectively show the *confusion zones*, so one can use them to decide which kinds of samples need to be further added to the training set to improve accuracy.

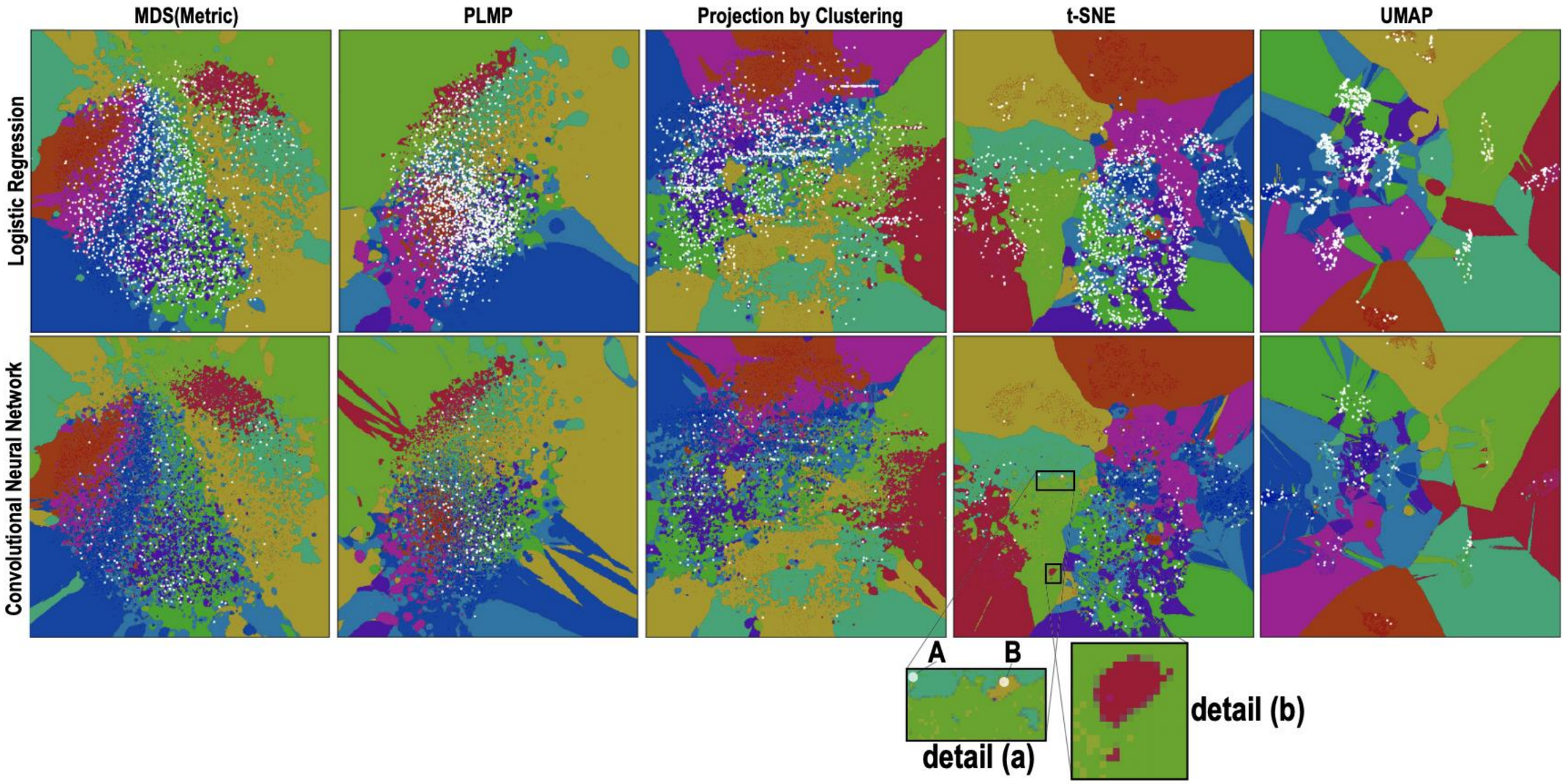



**Figure 5.** Classification errors (white dots) shown atop of the dense maps, logistic regression (LR) and CNN classifiers.

# We'll start with a white box attack

- Given the previous slides, it seems that a nice white box attack would be the following:
  - Take a benign image. Perturb (tweek) it so that it "moves" as little as possible toward the decision region of the class you want the image to be incorrectly classified as
  - It's a white box attack, since to do this you need to know the decision regions, which you only know if you have the parameters of the classifier
  - There is mathematics involved (multivariable calculus)
    - Generally, one looks at which inputs need to be tweaked to move classification toward the class you want, while simultaneously moving classification away from all of the other classes
    - Creates something called "saliency maps"

# We'll start with a white box attack

- Papernot, et. al., *The Limitations of Deep Learning in Adversarial Settings*, 2015, https://arx

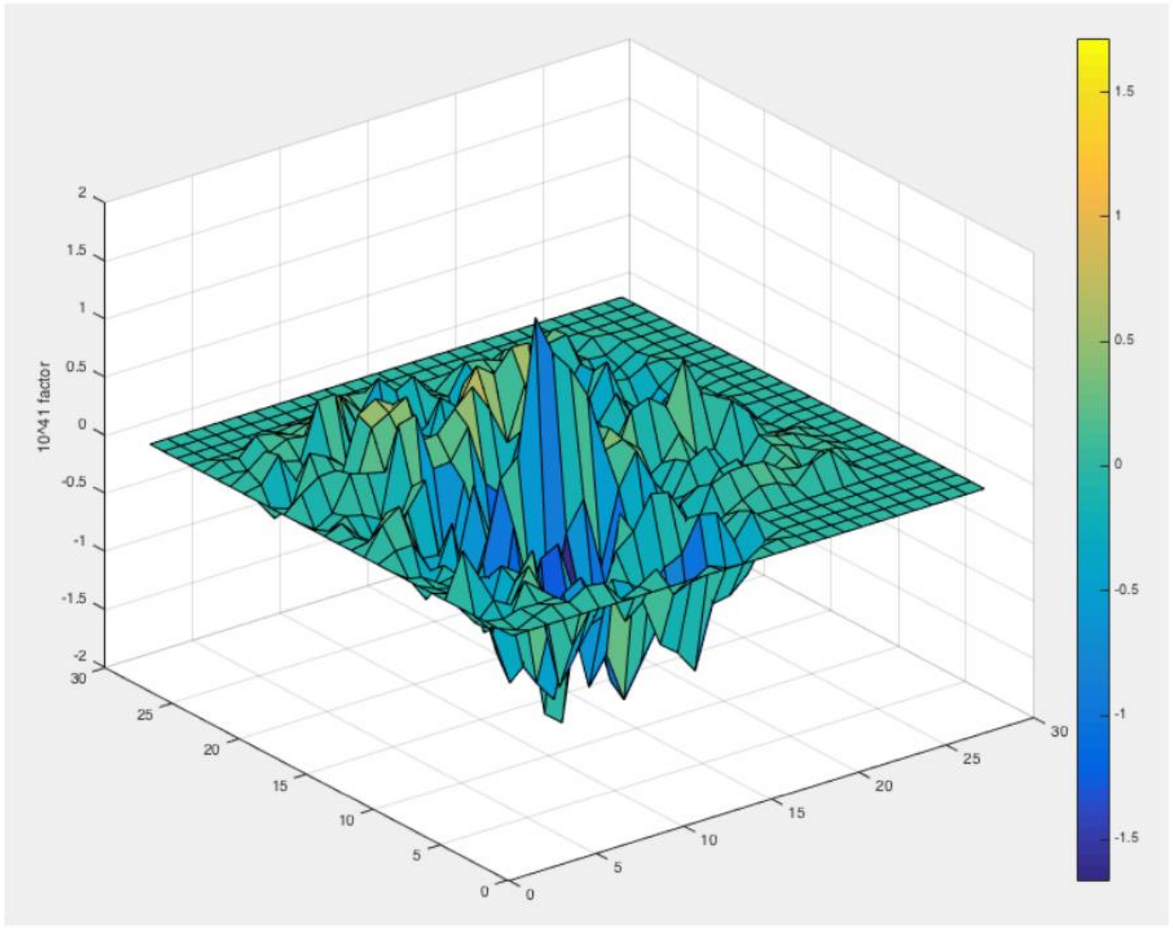



Fig. 7: Saliency map of a 784-dimensional input to the LeNet architecture (cf. validation section). The 784 input dimensions are arranged to correspond to the 28x28 image pixel alignment. Large absolute values correspond to features with a significant impact on the output when perturbed.

# We'll start with a white box attack

- Papernot, et. al., *The Limitations of Deep Learning in Adversarial Settings*, 2015, https://arx

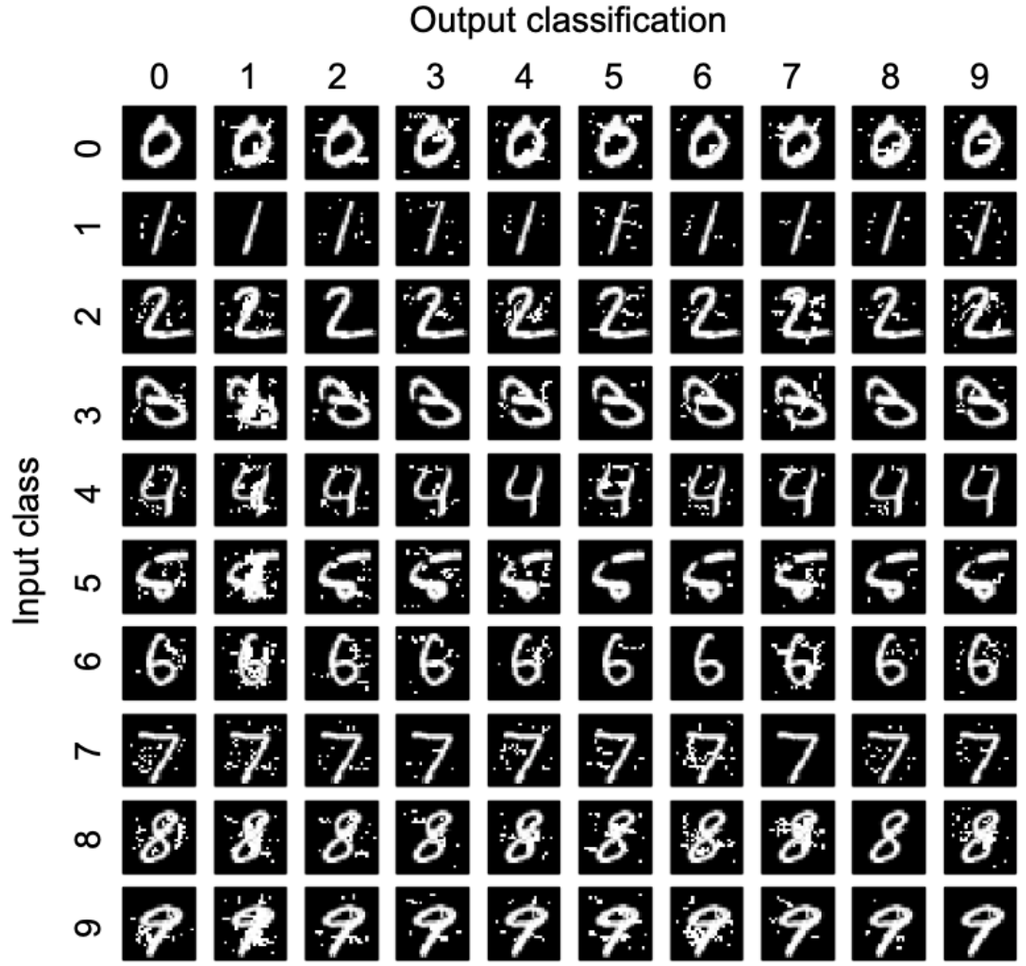
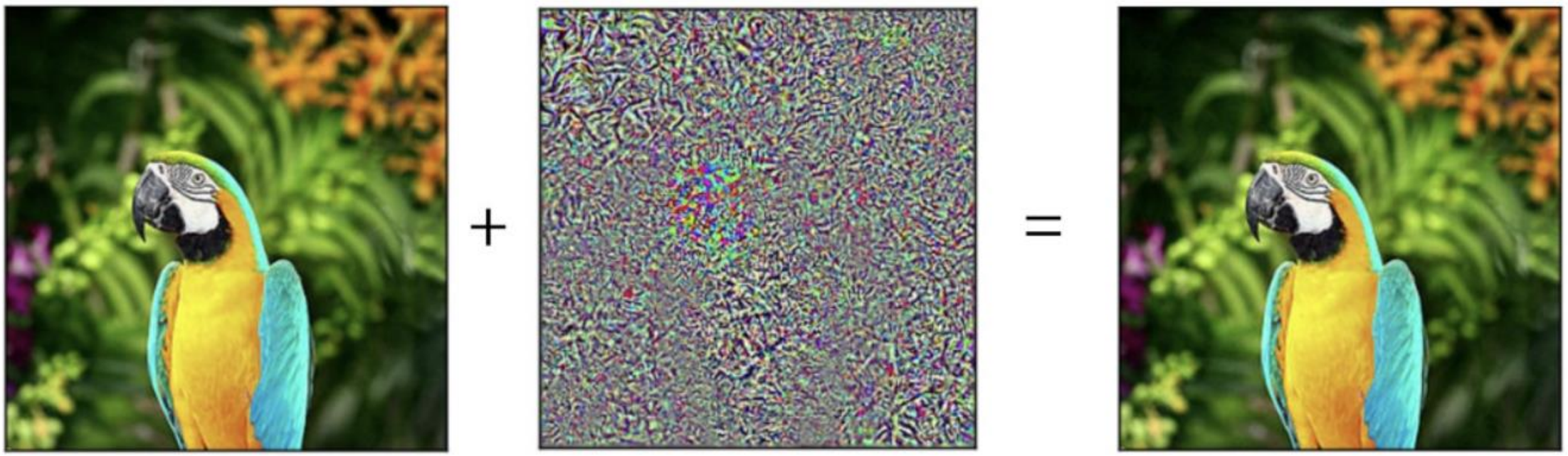



Fig. 1: *Adversarial sample generation* - Distortion is added to input samples to force the DNN to output adversary-selected classification (min distortion $= 0.26\%$, max distortion $= 13.78\%$, and average distortion $\varepsilon = 4.06\%$).

# We’ll start with a white box attack

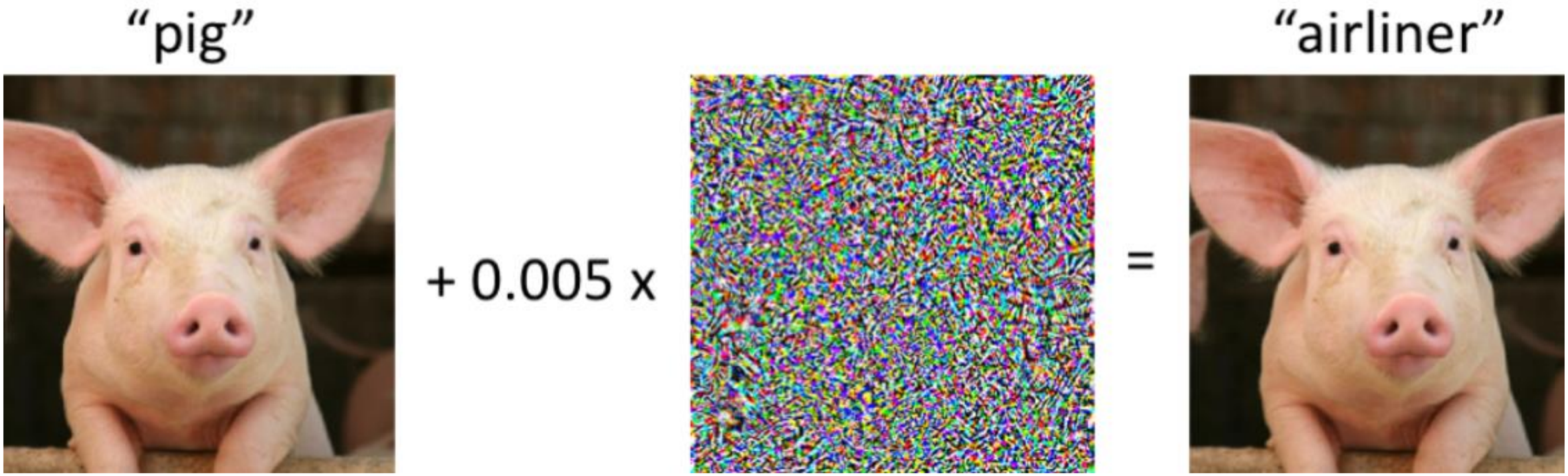
- On the previous images, you could see the changes in some. How about in these?



- Macaw + noise = Book case

# We’ll start with a white box attack

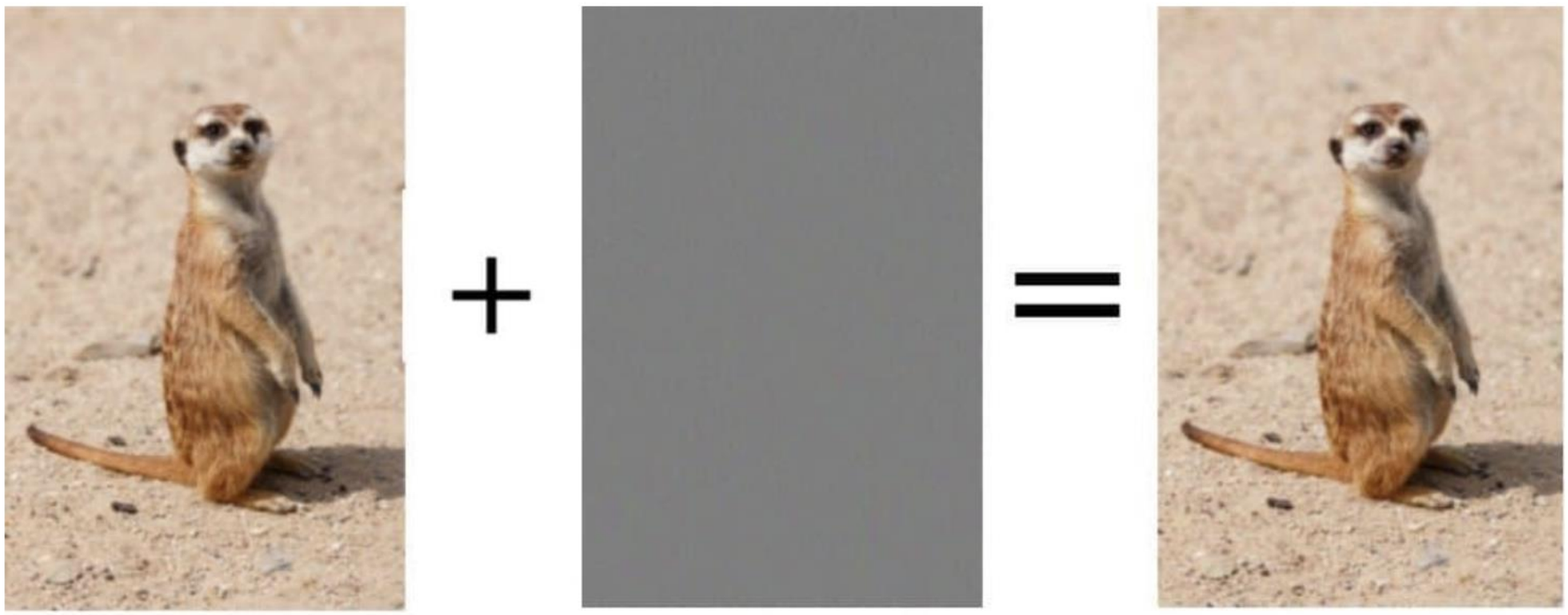
- Or these?




- Pig + noise = Airliner

# We'll start with a white box attack

- Or these?



meerkat, mierkat (score = 0.90021)
mongoose (score = 0.02666)
Windsor tie (score = 0.00072)
otter (score = 0.00069)
doormat, welcome mat (score = 0.00055)

kite (score = 0.07896)
bald eagle, American eagle, Haliaeetus leucocephalus (score = 0.04153)
bee eater (score = 0.03940)
parachute, chute (score = 0.02724)
hummingbird (score = 0.02334)

doormat, welcome mat (score = 1.00000)
prayer rug, prayer mat (score = 0.00000)
manhole cover (score = 0.00000)
miniature poodle (score = 0.00000)
palace (score = 0.00000)

- Meerkat + Noise = Welcome mat

# We'll start with a white box attack

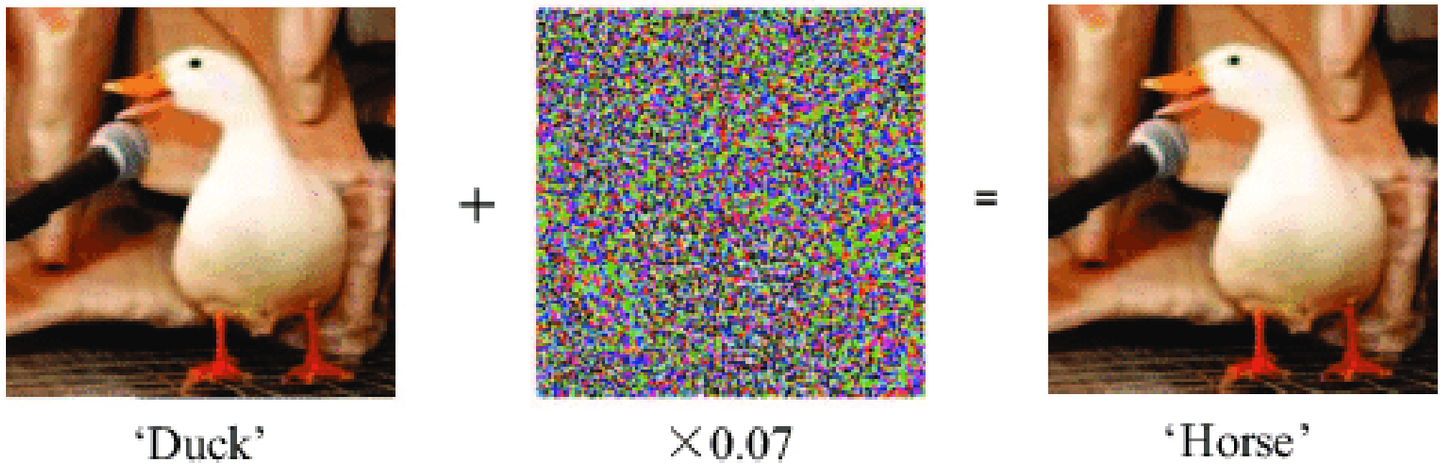
- Or all of these?

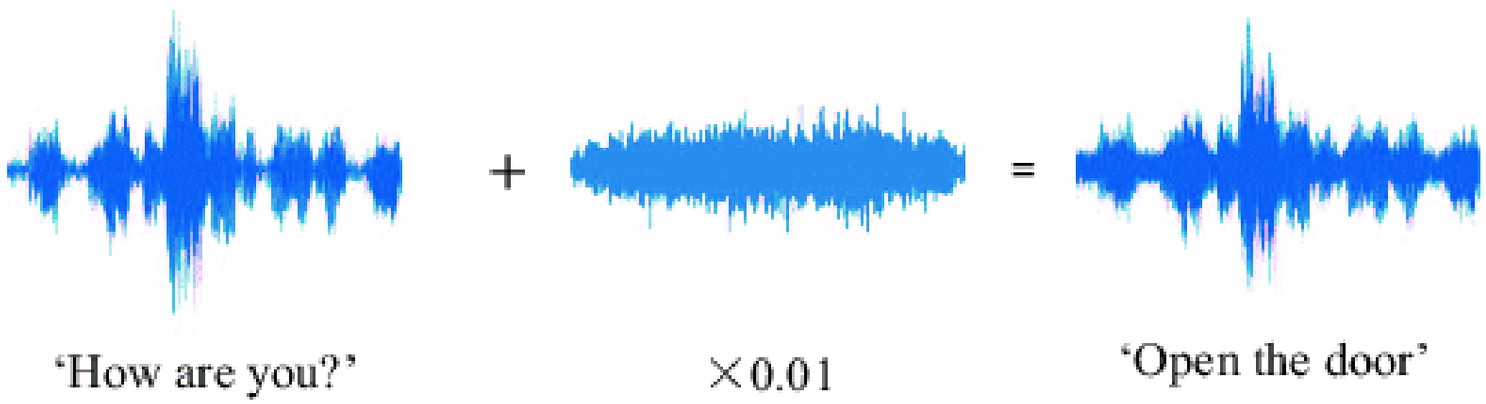
(a) (b)

Figure 1: (a): Left, the original image that was classified correctly, middle, the difference between the two images, right, the perturbed image that was classified incorrectly. (b) The same as (a), but the right column was categorized as an "ostrich, Struthiocamelus". [Szegedy et al. 2013]

# We'll start with a white box attack

- You might think, OK, so this works with images. But what about other things?




- Note the bottom example!

# We'll start with a white box attack

- And you might say, "OK, so this works with a white box attack, but how practical are they?"
- Well, as it turns out, in some cases (images being the most prominent), there is something called *transferability*
- Researchers had wondered whether one could infer the parameter values of a neural network given only black box access. Papernot showed (in another paper) that this isn't necessary!
  - Build your own substitute model, and train it using input-output pairs you get by running the original model. You have white-box access to the substitute network, so can generate adversarial examples on it. About 85% of the time, those examples will also work on the original neural network! That's transferability.

# Transferability

- So, as mentioned, it often works with image classification
- But it doesn't work at all with automated voice processing
  - Relativy recent work (since 2021) explains several of the factors that prevent transfer of adversarial examples in automated voice processing systems
  - One of the primary factors is that automated voice processing systems employs types of neural networks that image processing systems don't
    - Recurrent Neural Networks (RNNs) and now transformers are networks designed to handle sequential data and to "remember" context (which is important in speech)
    - These networks (and other speech-specific processing) seem to prevent transfer of adversarial examples.

# Black Box Attacks

- So are there any successful black box attacks?
- It turns out there are!
- Let's consider, for a second, automated voice processing, and in particular, speech-to-text
- Two kinds of attacks:
  - Fool the AVP system, but not humans
    - Two people one a cell phone call can understand each other, but the AVP system listening in is confused
  - Fool humans but not the AVP system
    - Humans in presence of audio signal either don't hear it or hear it as background noise, but AVP system understands "hidden" commands

# Black Box Attacks

- So are there any successful black box attacks?
- It turns out there are!
- Abdullah, et. al., *Hear "No Evil", See "Kenansville": Efficient and Transferable Black-Box Attacks on Automatic Speech Recognition and Voice Identification Systems*, Proceedings of IEEE Symposium on Security and Privacy, May, 2021.
- https://sites.google.com/view/transcript-evasion

# Black Box Attacks

- Given that there are all of these attacks, how can we defend against them?  Ideas?

# Explanation Methods

- It has become clear that we need a way to understand how classifiers reason
  - That is, understand why they classify specific examples the say they dod
- So recently, researchers have begun looking into explanation methods
- Let's begin our discussion of this with a return to an old friend:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots + \beta_k x_k$$

- If you know the beta values, do you know which feature(s) are more important than others?

# Explanation Methods

- If you know the beta values, do you know which feature(s) are more important than others?

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots + \beta_k x_k$$

- Of course, we know that models like neural networks are highly nonlinear, so we can't hope to end up with something like this linear model.

# Explanation Methods

- If you know the beta values, do you know which feature(s) are more important than others?

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots + \beta_k x_k$$

- Of course, we know that models like neural networks are highly nonlinear, so we can't hope to end up with something like this linear model.

- Or can we?

# Explanation Methods

- If you know the beta values, do you know which feature(s) are more important than others?

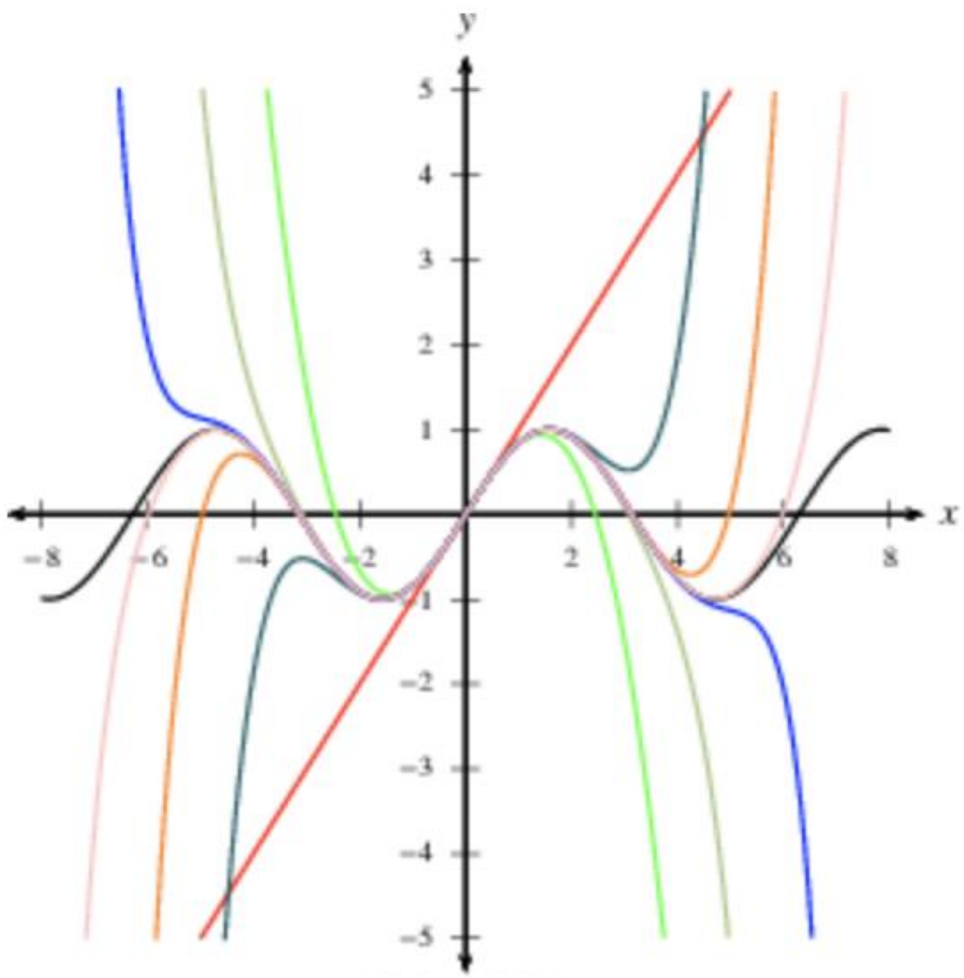
$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots + \beta_k x_k$$

- Of course, we know that models like neural networks are highly nonlinear, so we can't hope to end up with something like this linear model.

- Or can we?

# Explanation Methods

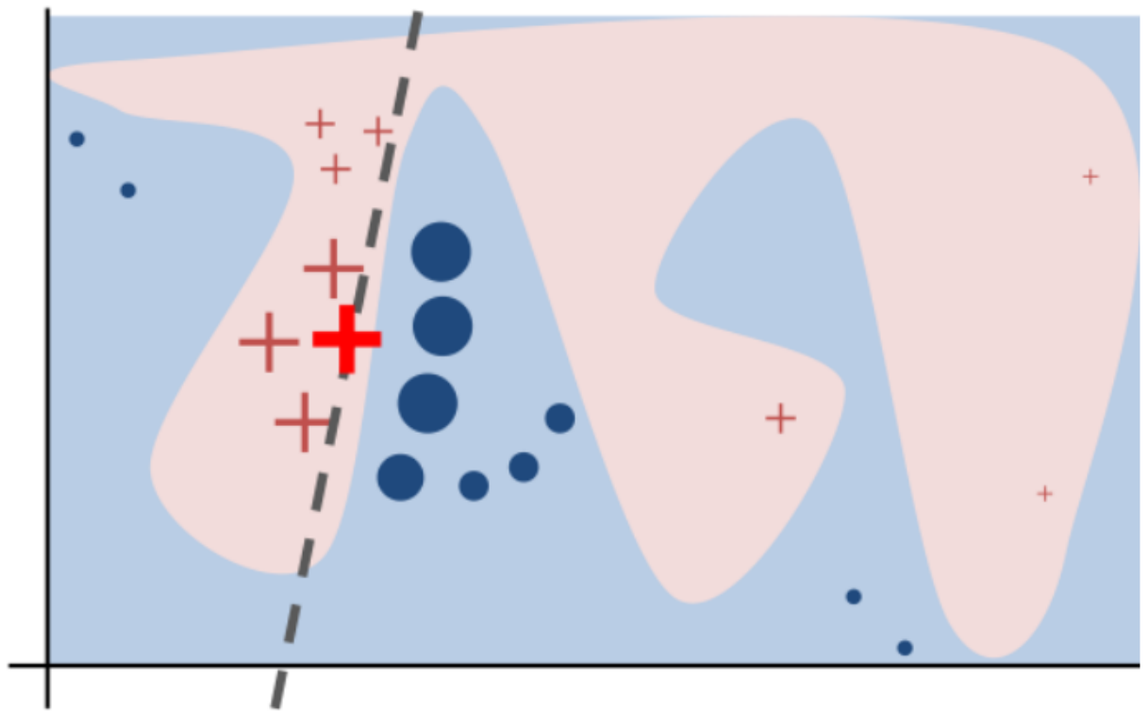
- We can’t approximate the entire functionality of most models with a linear model, but we can often approximate behavior in a very small neighborhood with a linear model




$f(x) = \sin(x)$

# Explanation Methods

- So the idea: if you want to know why a specific input is classified the way it is, approximate the decision boundary in the neighborhood of the input of interest with a linear model.



- How?

# Explanation Methods

- How?  Well, we know how to build linear regression models.  All we need is data.
- Where do we get that?
- Two examples of doing this: LIME and LEMNA
  - Google them if interested

# Explanation Methods

- Let’s look at an example of the results

(a) Input Image. (b) Explanation. (c) Deduc. test. (d) Augme. test. (e) Synthet. Test.

**Figure 5: We use an image classifier as an toy example to explain the fidelity test. Figure 5a is the original input image (“sweater”). Figure 5b is the explanation produced by LEMNA where important features (pixels) are highlighted in red. Figure 5c–5e are three testing instances we generated to test the fidelity of the explanation.**