Adversarial ML Workshop

By JankhJankh

JankhJankh

Adversarial ML Workshop

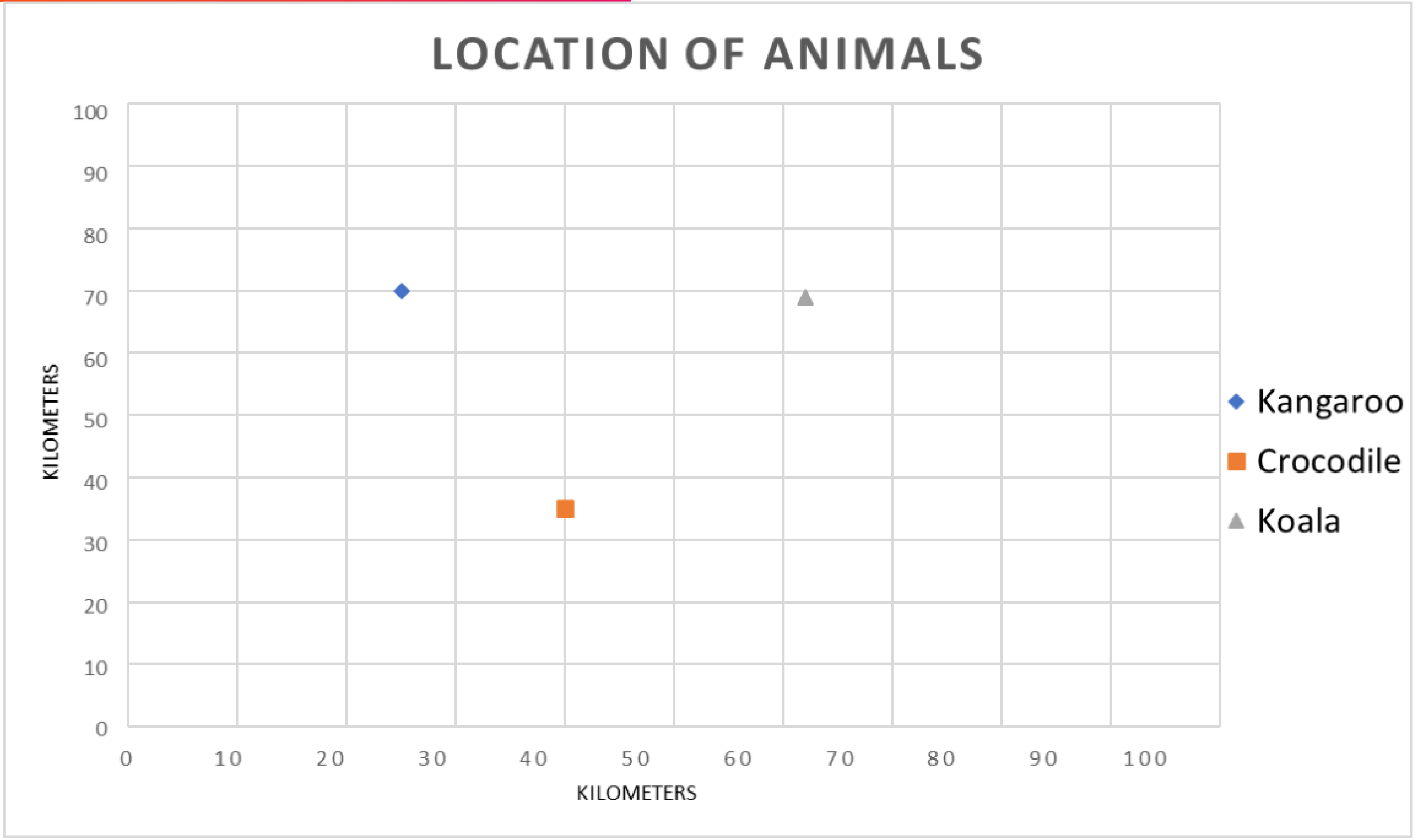
Ground Rules

- Everyone has their own docker container so you can do what you want with it.
- Each challenge works in a vacuum except Theft3 which should be done after getting code exec in another challenge. (Although I have provided a work around for Theft3 if you can't get code exec but want to try it.
- Source code is provided with redactions, even after getting code execution, avoid looking at the source code for other challenges
- If a file can't be found in the provided source code or html code, you aren't supposed to find it ;)
- If you need to guess a password use Rockyou.

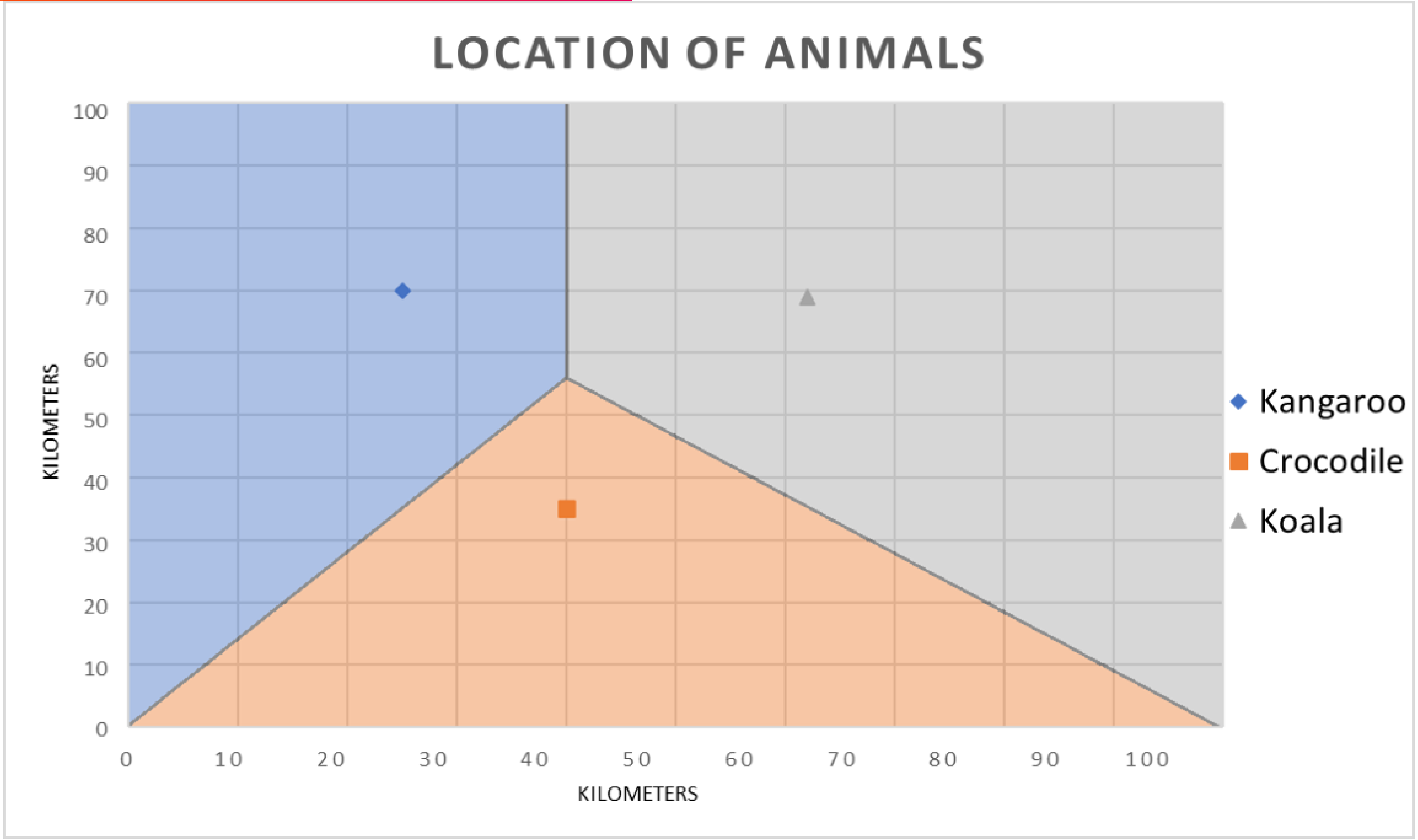
Theory you may need

- How Classification Models Work
- How a Neural Network Works
- Machine Learning Threat Model
- MITRE Adversarial Machine Learning Threat Matrix
- Tokenizers
- Evasion Attacks
- Data Poisoning
- Model Salting
- Multiple Assessment Functions

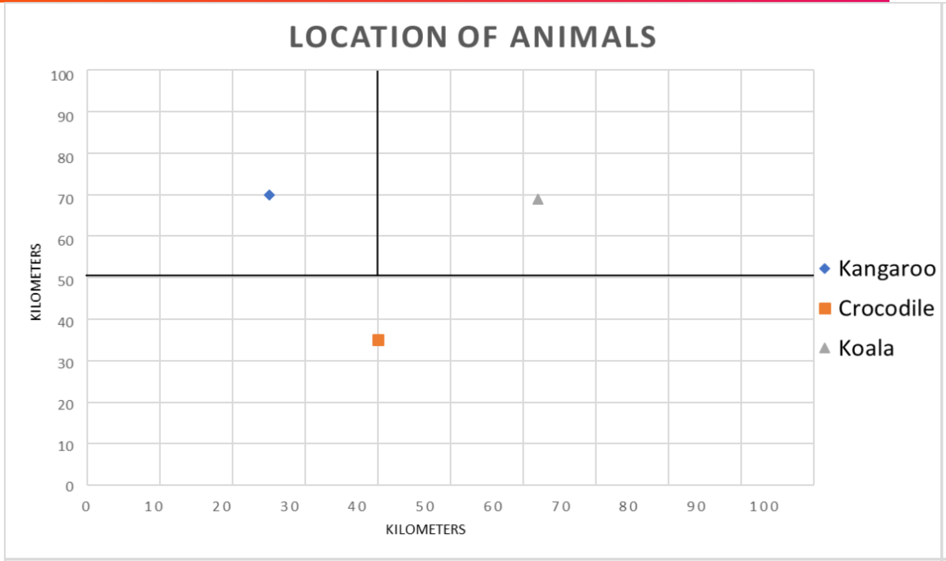
• How Classification Models Work

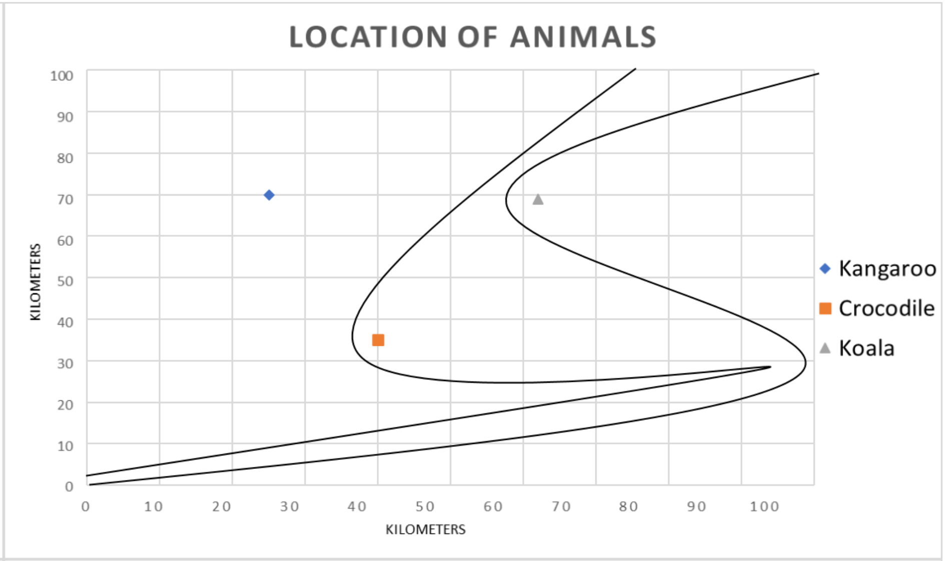


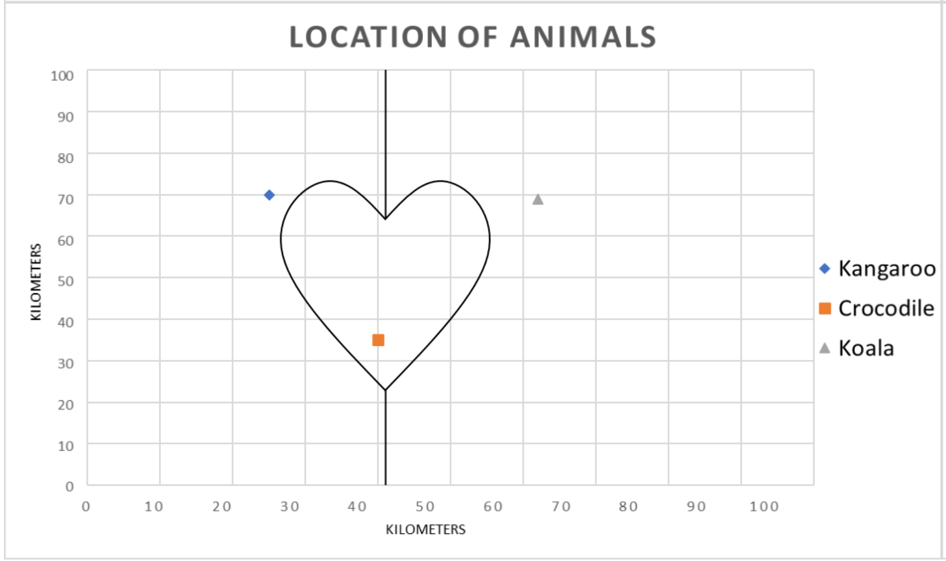
• How Classification Models Work

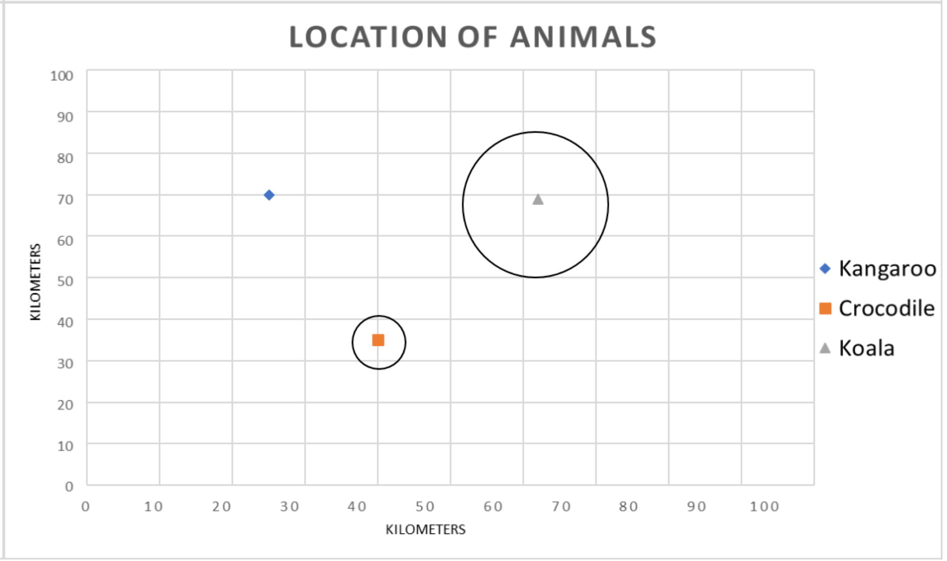


- **How Classification Models Work**

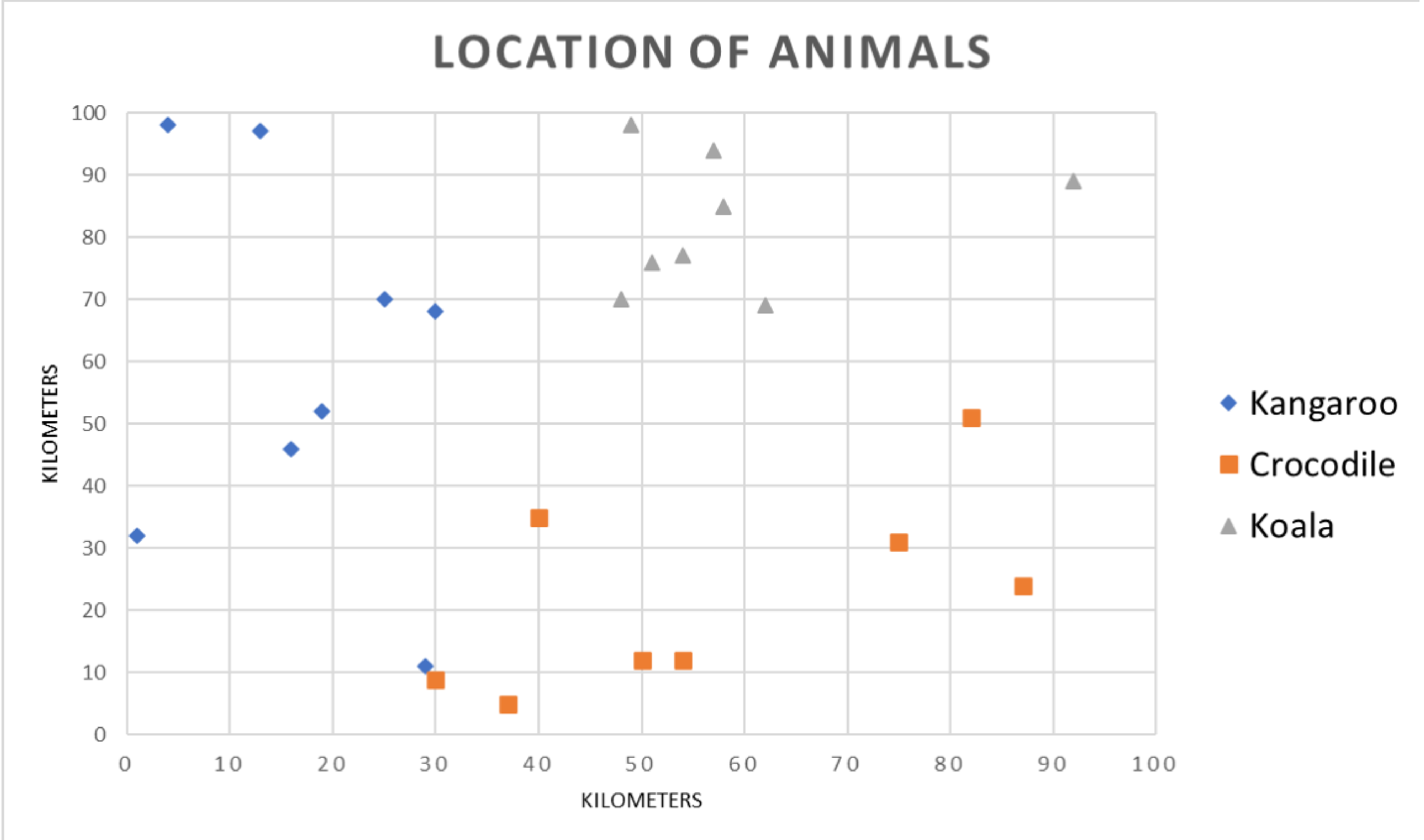




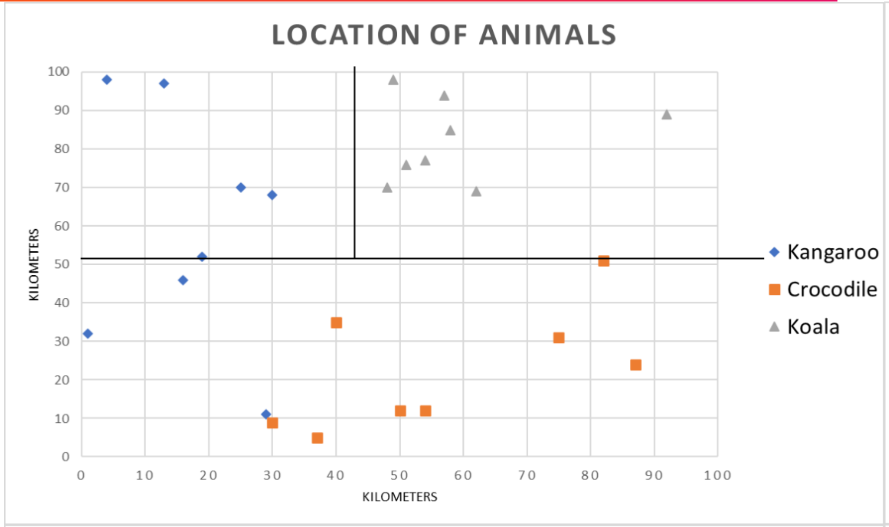


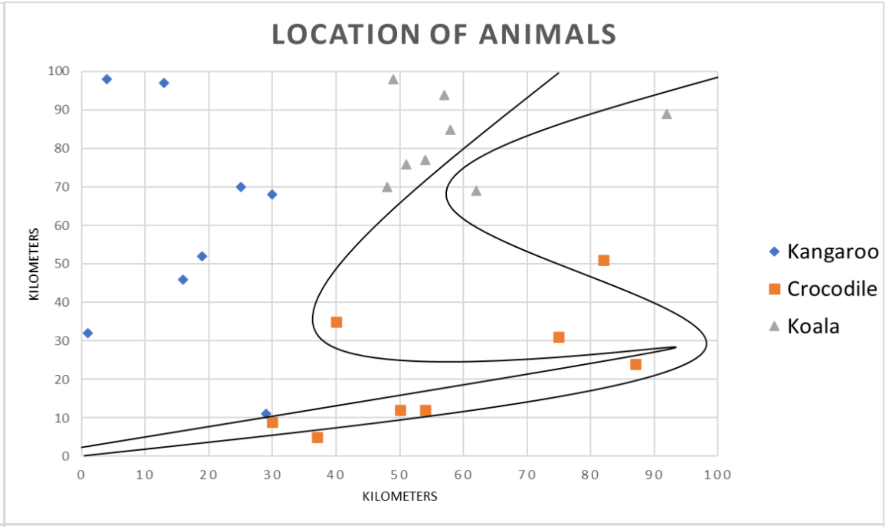


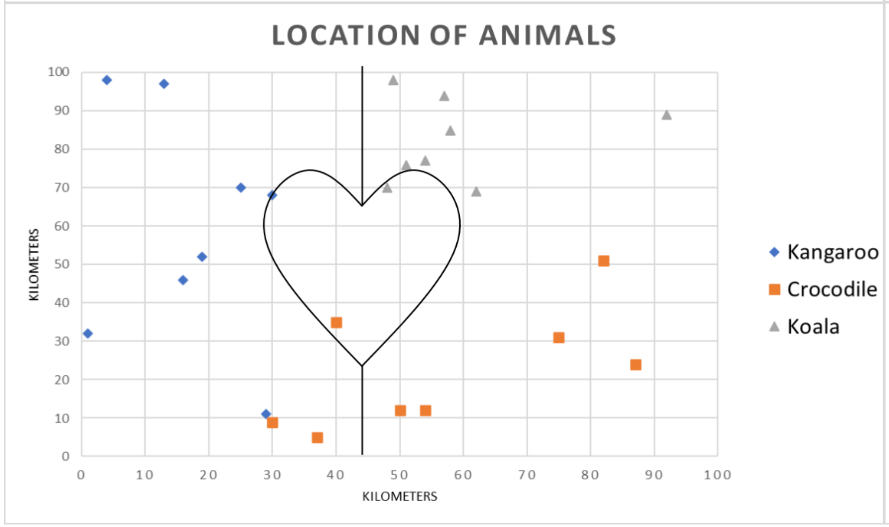
• How Classification Models Work

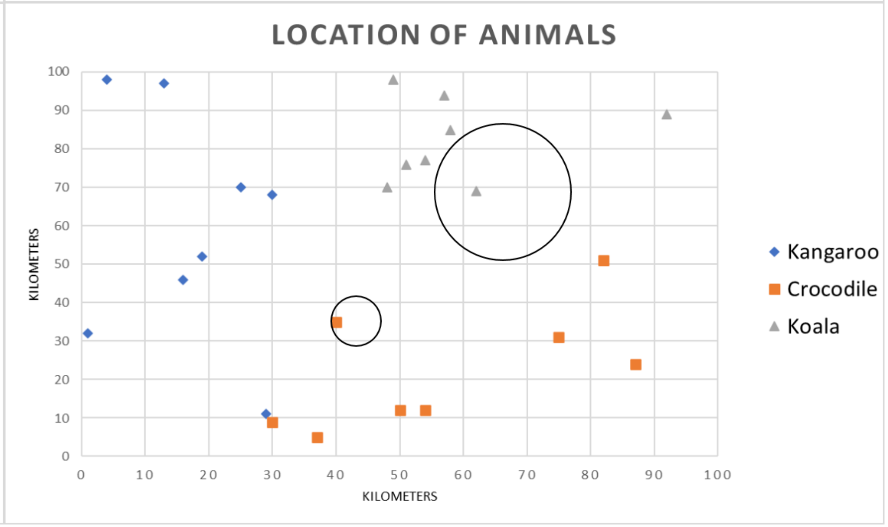


• How Classification Models Work

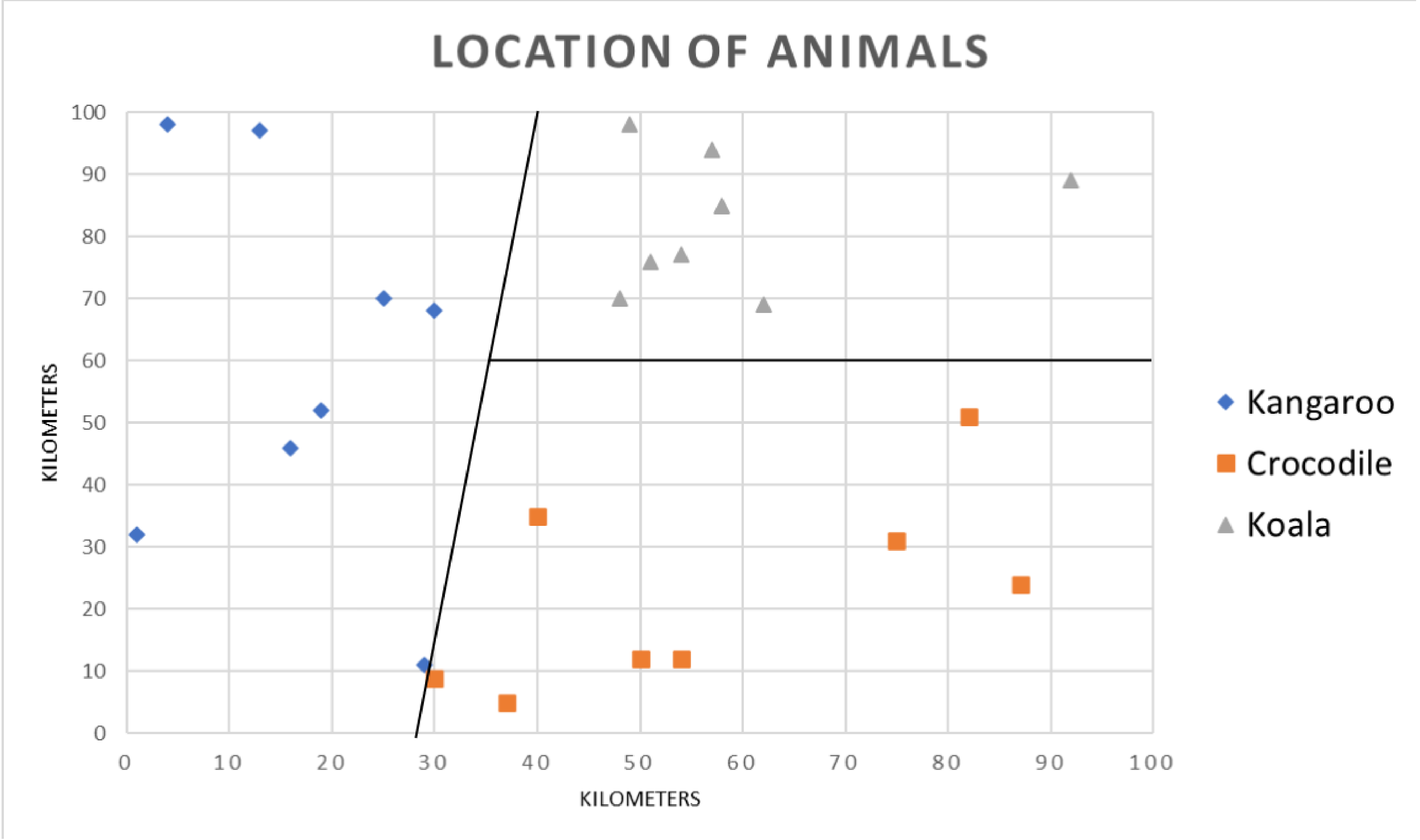




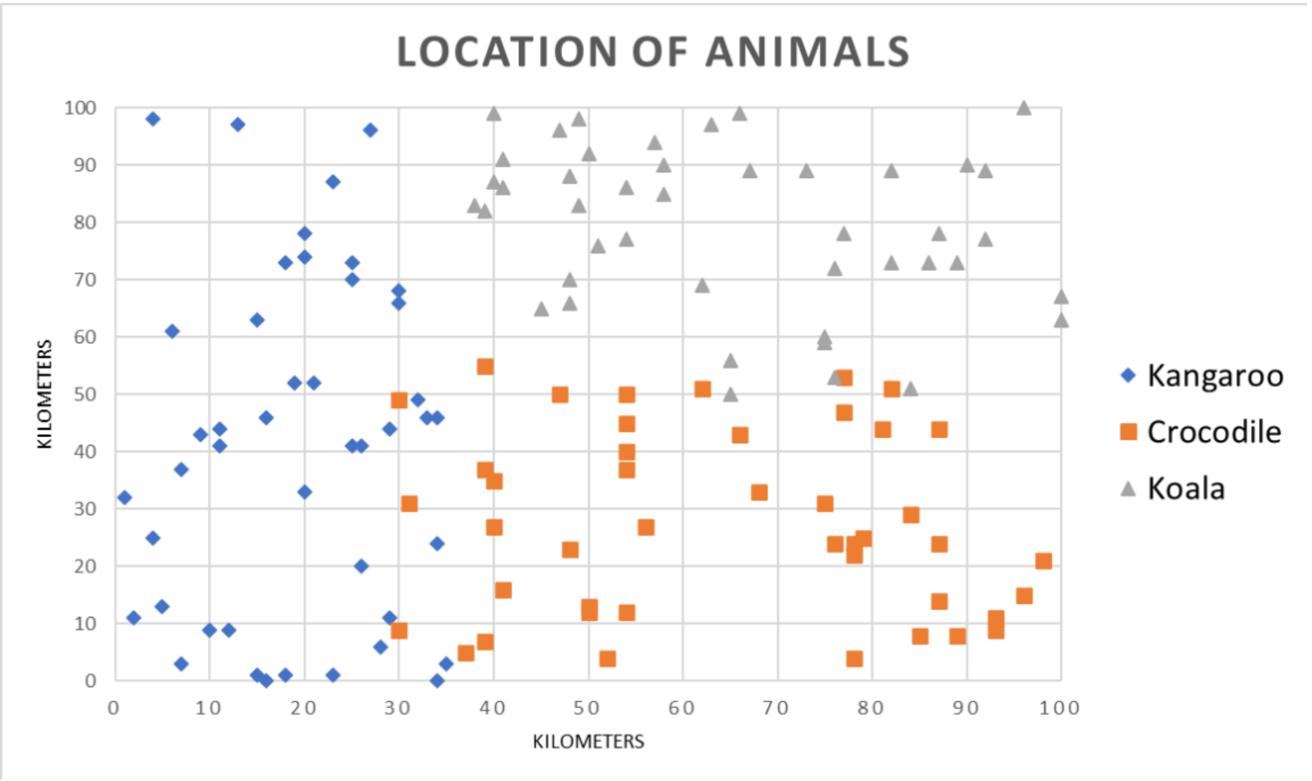


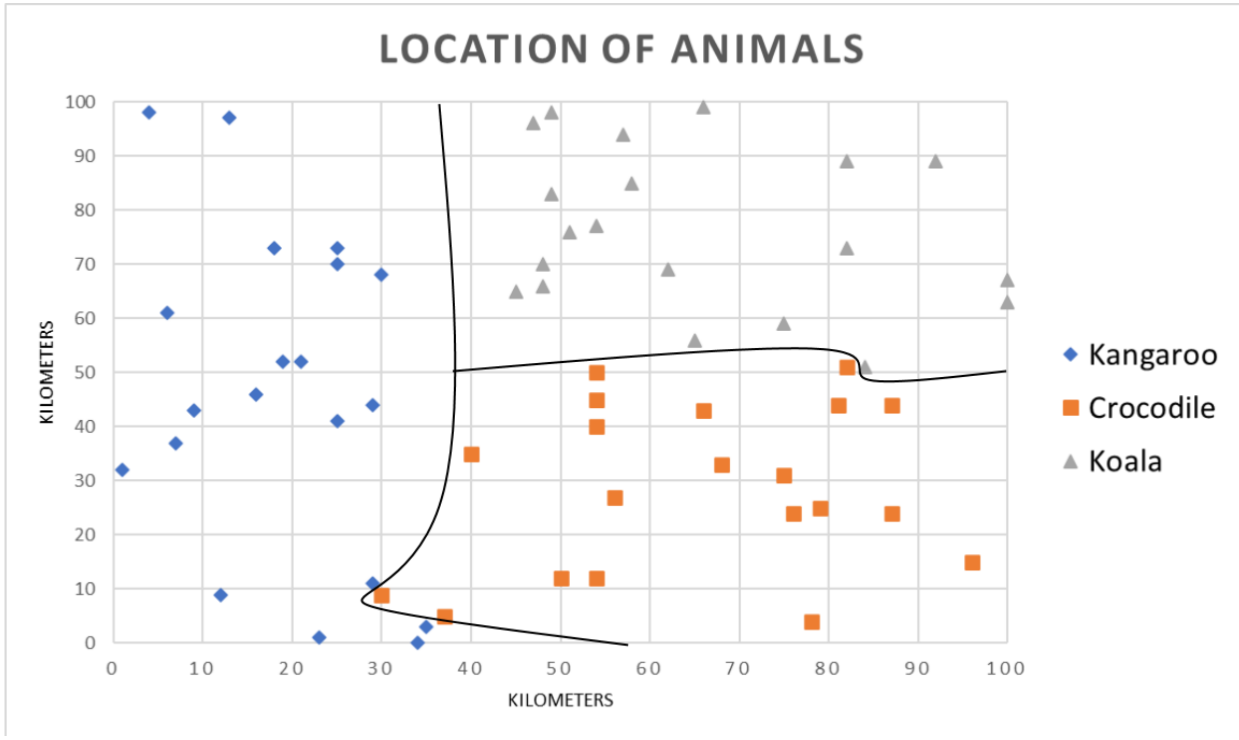


• How Classification Models Work



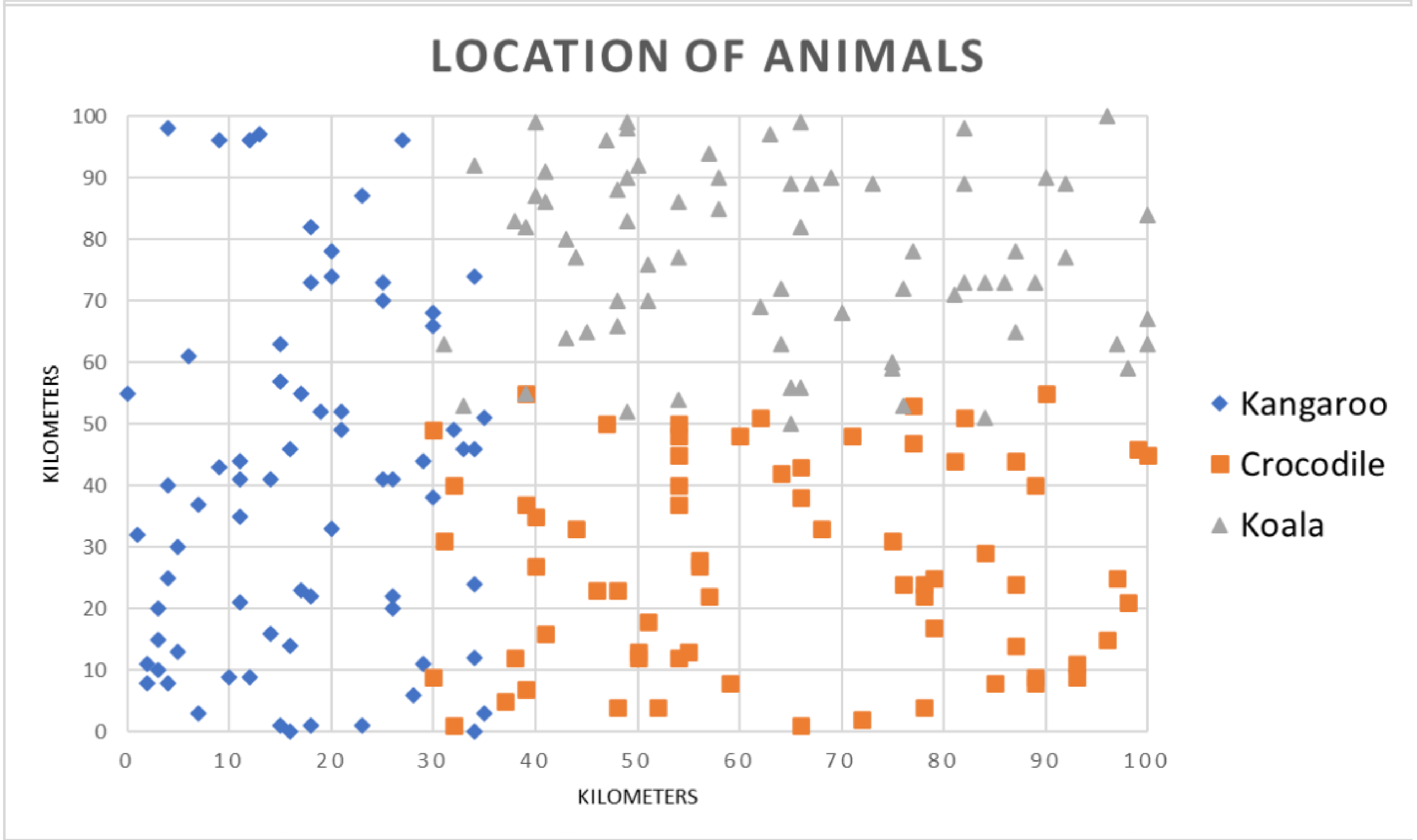
• How Classification Models Work



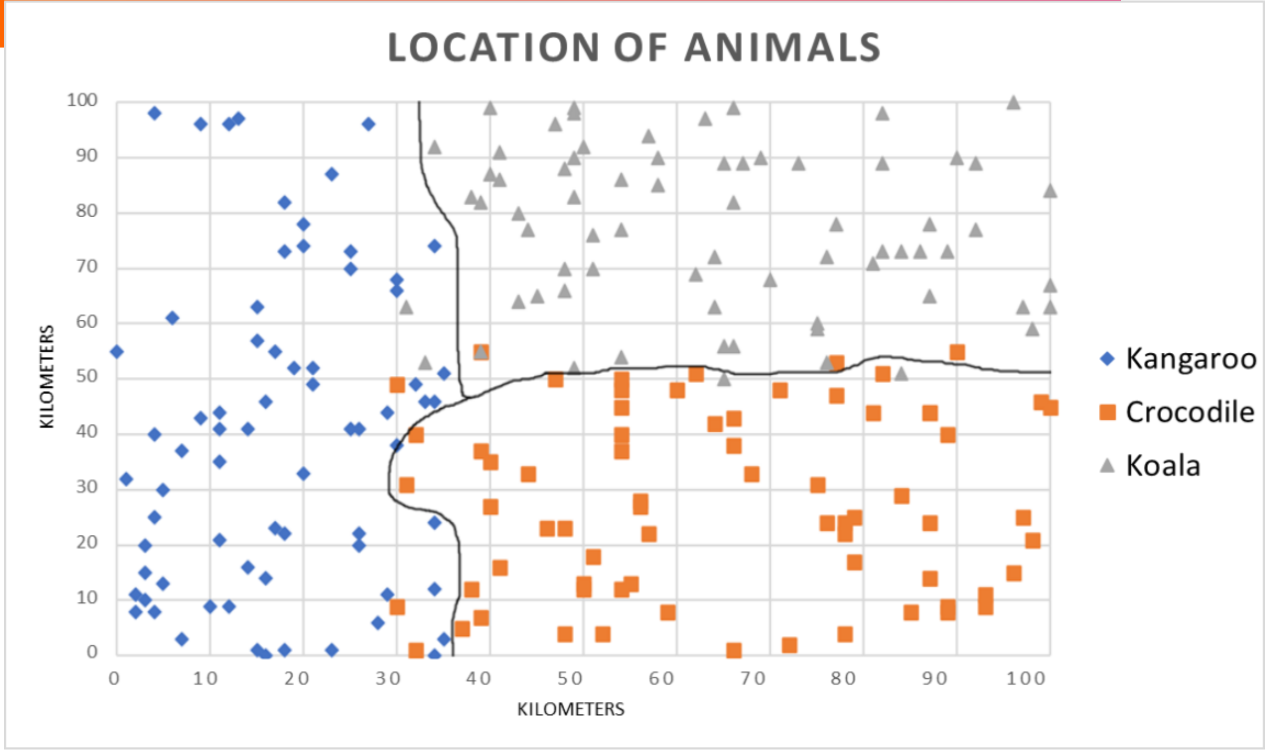


More Data = Better models

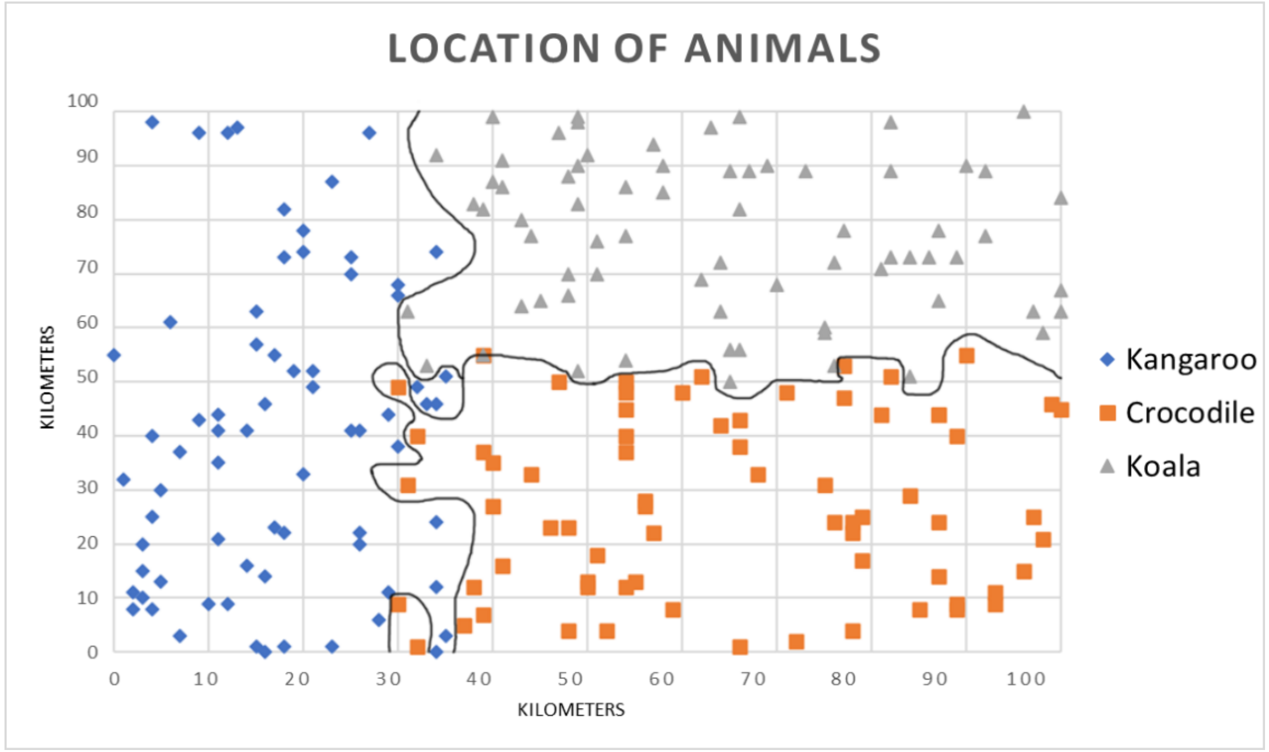
• How Classification Models Work



How Classification Models Work



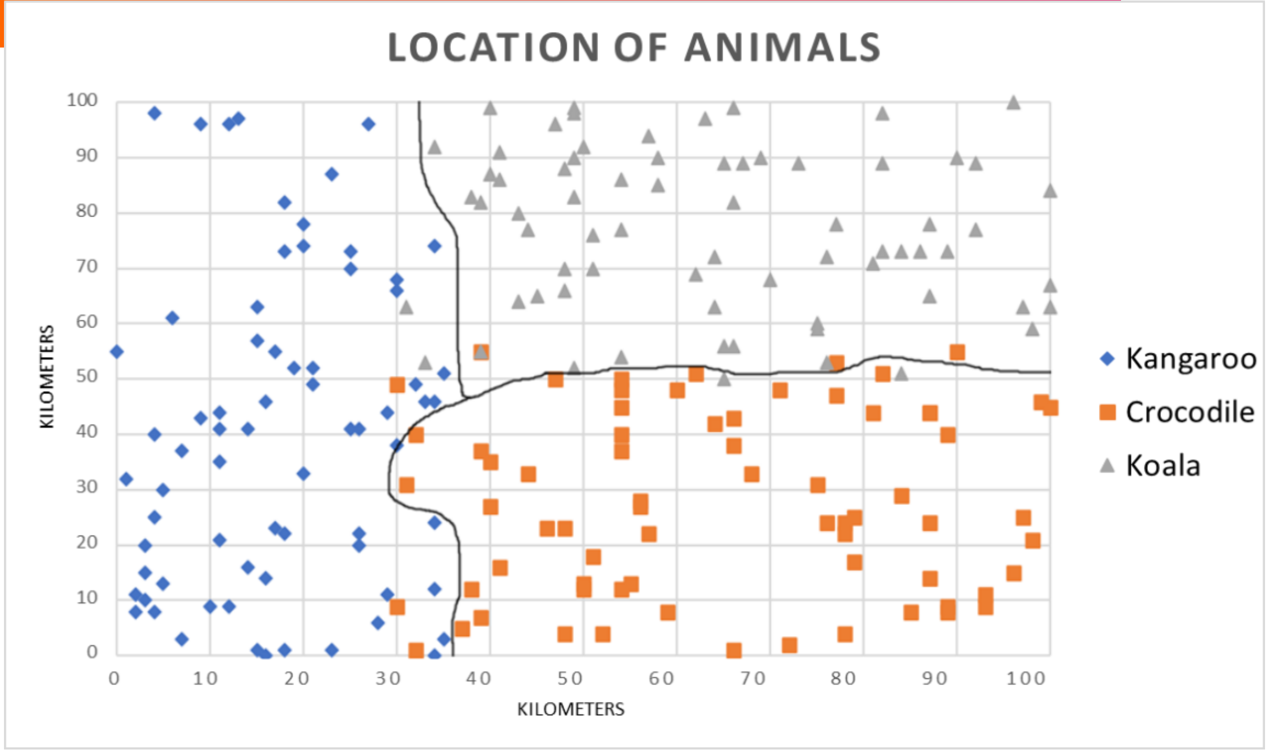
- 80% Accuracy

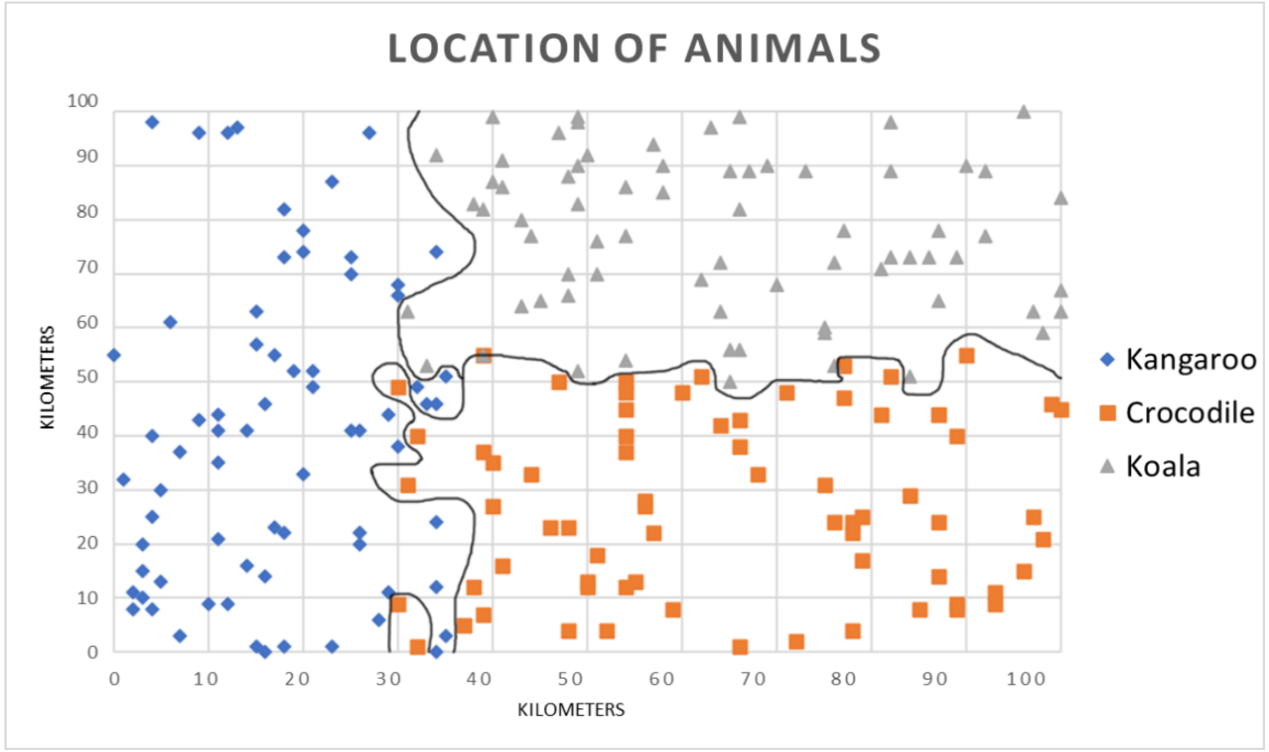


100% Accuracy

Which of these models is better?

• How Classification Models Work





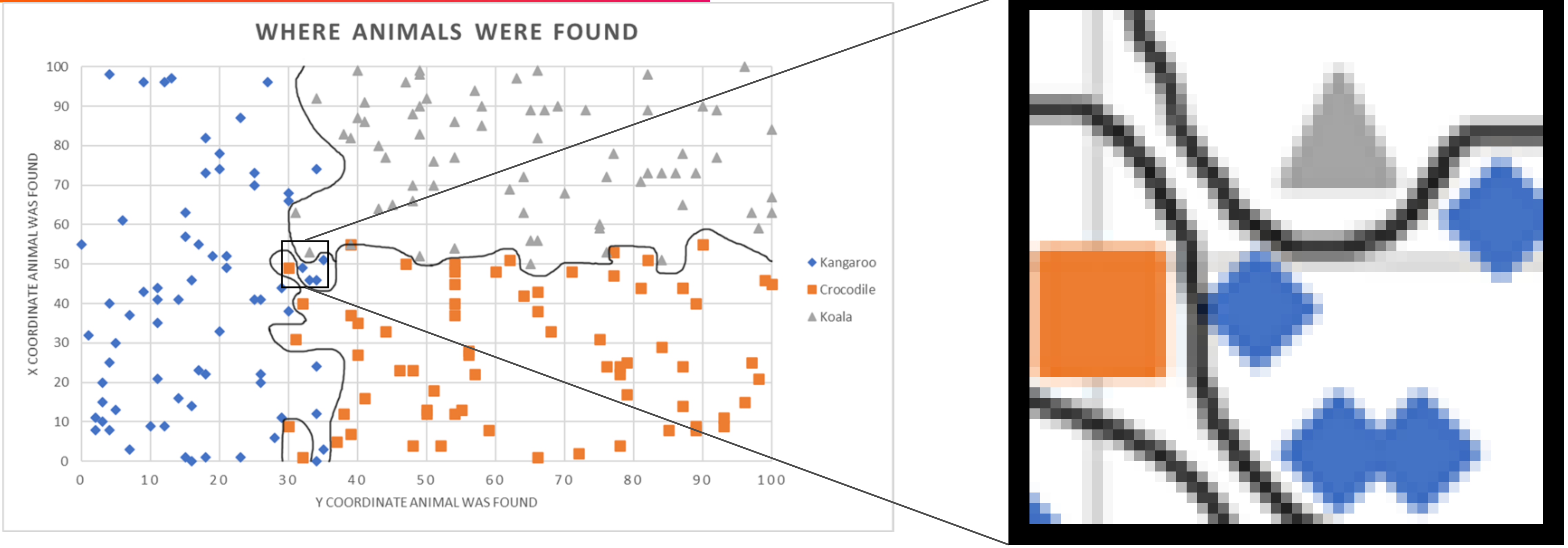
• 80% Accuracy

100% Accuracy (Overtrained)

As 80% is better than 100% as it is more general.

Modelling for generalisation is inherently inaccurate to an extent

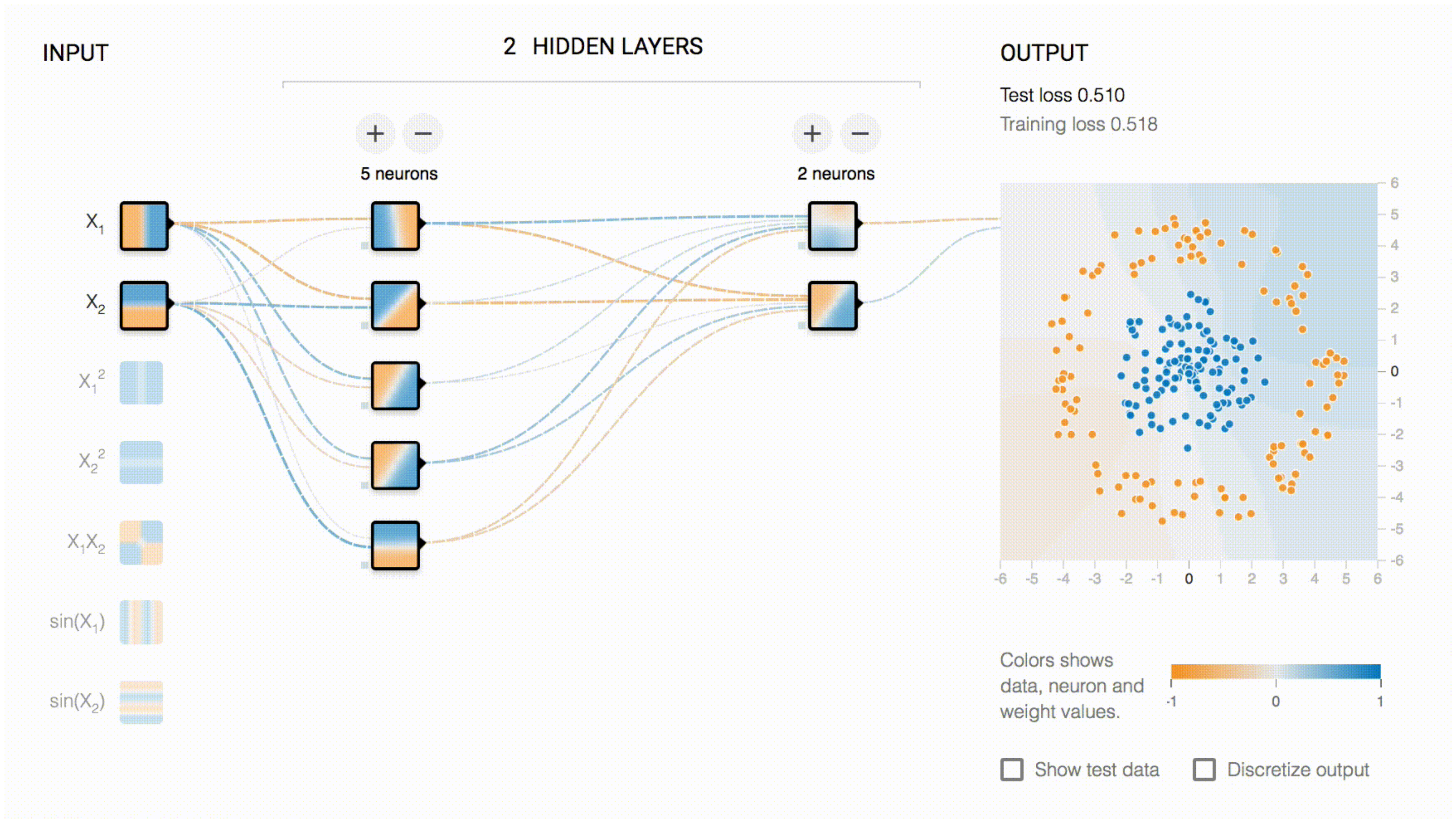
• How Classification Models Work



In an overtrained network, little changes in a datapoint can make all the difference in classification.

X	Y	Class
32	50	Kangaroo
30	50	Crocodile
33	54	Koala

How A Neural Network Works



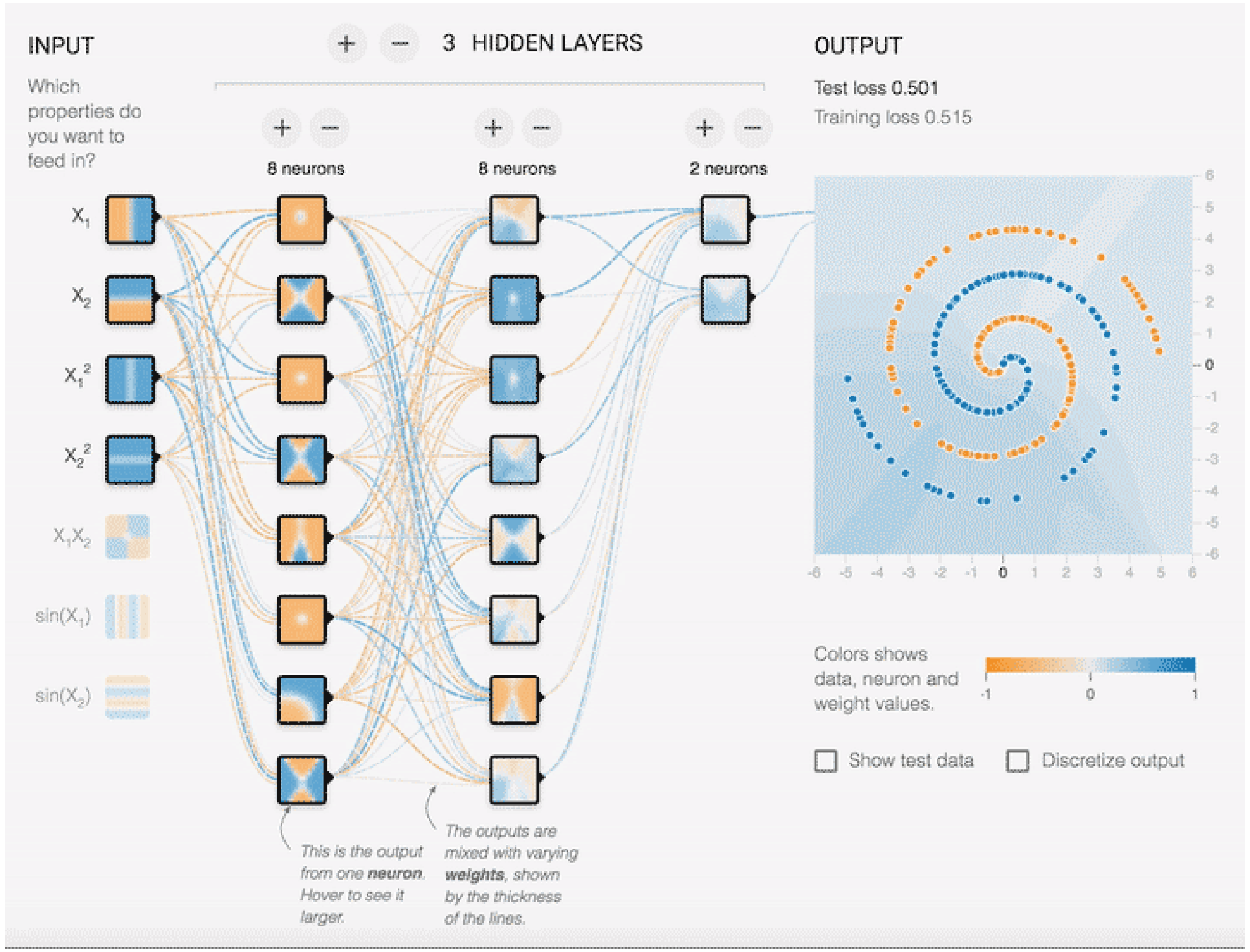
Test loss 0.510
Training loss 0.518

Colors shows data, neuron and weight values.

☐ Show test data ☐ Discretize output

Fig. 4.

How A Neural Network Works


INPUT

Which properties do you want to feed in?

3 HIDDEN LAYERS

OUTPUT

Test loss 0.501

Training loss 0.515

This is the output from one **neuron**. Hover to see it larger.

The outputs are mixed with varying **weights**, shown by the thickness of the lines.

Colors shows data, neuron and weight values.

Show test data

Discretize output

Fig. 5.

The Machine Learning Threat Model

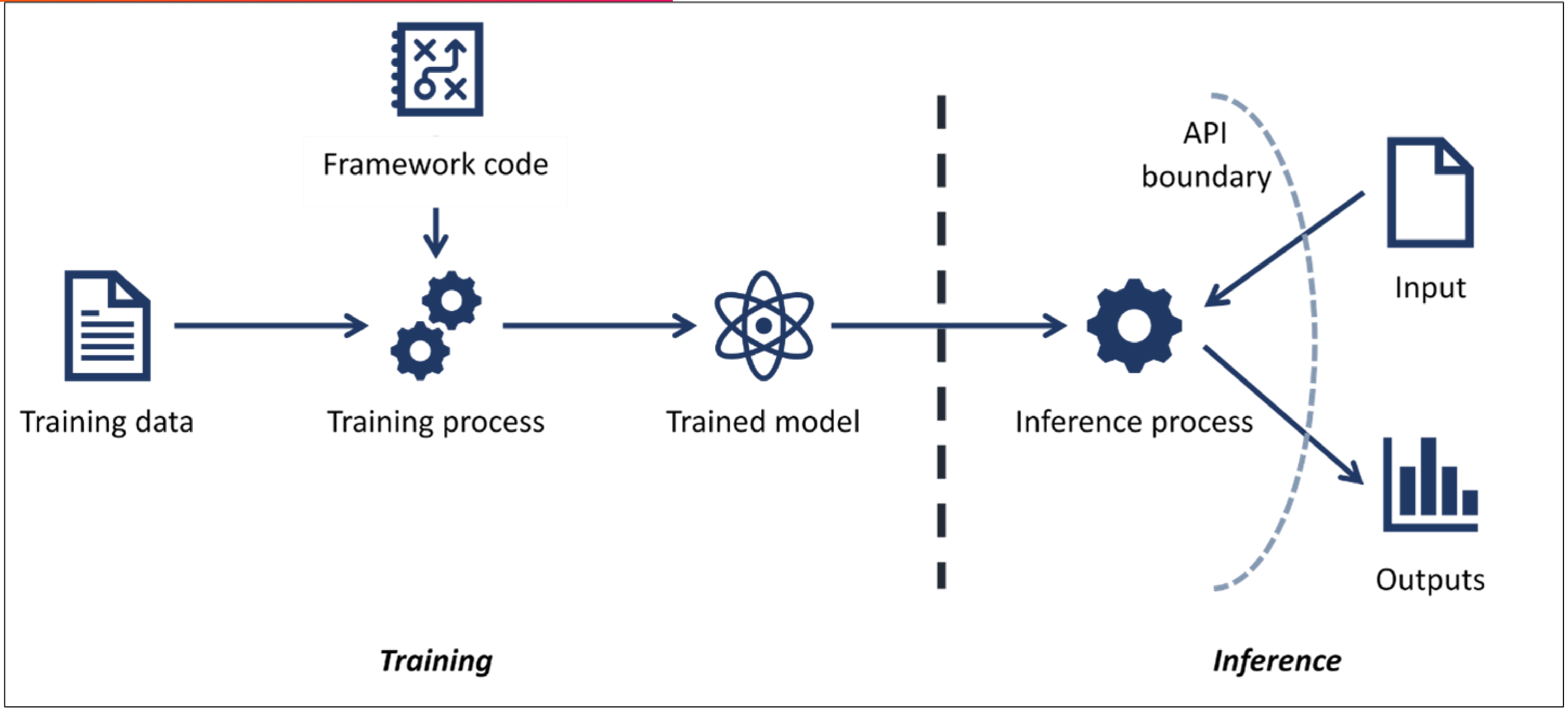


Fig. 6.

Whatever parts of this model users are interacting with are at risk

MITRE Adversarial Machine Learning Threat Matrix

Reconnaissance	Initial Access	Execution	Persistence	Model Evasion	Exfiltration	Impact
Acquire OSINT information: (Sub Techniques) 1. Arxiv 2. Public blogs 3. Press Releases 4. Conference Proceedings 5. Github Repository 6. Tweets	Pre-trained ML model with backdoor	Execute unsafe ML models (Sub Techniques) 1. ML models from compromised sources 2. Pickle embedding	Execute unsafe ML models (Sub Techniques) 1. ML models from compromised sources 2. Pickle embedding	Evasion Attack (Sub Techniques) 1. Offline Evasion 2. Online Evasion	Exfiltrate Training Data (Sub Techniques) 1. Membership inference attack 2. Model inversion	Defacement
ML Model Discovery (Sub Techniques) 1. Reveal ML model ontology – 2. Reveal ML model family –	Valid account	Execution via API	Account Manipulation		Model Stealing	Denial of Service
Gathering datasets	Phishing	Traditional Software attacks	Implant Container Image	Model Poisoning	Insecure Storage 1. Model File 2. Training data	Stolen Intellectual Property
Exploit physcial environment	External remote services			Data Poisoning (Sub Techniques) 1. Tainting data from acquisition – Label corruption 2. Tainting data from open source supply chains 3. Tainting data from acquisition – Chaff data 4. Tainting data in training environment – Label corruption		Data Encrypted for Impact Defacement
Model Replication (Sub Techniques) 1. Exploit API – Shadow Model 2. Alter publicly available, pre-trained weights	Exploit public facing application					Stop System Shutdown/Reboot
Model Stealing	Trusted Relationship					

Fig. 12.

Tokenizers

To normalize models regardless of data type, tokenizers can be used. Instead of inputting your data into the model you just give each class a token that it relates to. EG:

Kangaroo: 0

Crocodile: 1

Koala: 2

The model doesn't actually know what these IDs correlate to, so it will ask the tokenizer. This tokenizer needs to be used in both the training and the inference stages and needs to be consistent between both ;)

For the sentiment analysis model in the CTF, it uses a tokenizer to give each word a value, and then evaluates each sentence as a list of IDs, eg: The number after is either a 1 or a 0 corresponding to positive sentiment or negative sentiment.

What a lovely day,1 -> [28,50,19,200]=1
This absolutely sucked,0 -> [1,234,24]=0
This was not lovely,0 -> [1,99,2,13]=0
This was lovely,1 -> [1,99,13]=1

Evasion Attack

Steps to do Adversarial Noise:

1. Slightly change the image (e.g. by gaussian noise)
2. Check the image confidence against the model
3. If it has higher owl confidence it updates current image
4. Repeat steps 2 and 3 until the model classifies it as the target class

My Table of Owls and Turtles

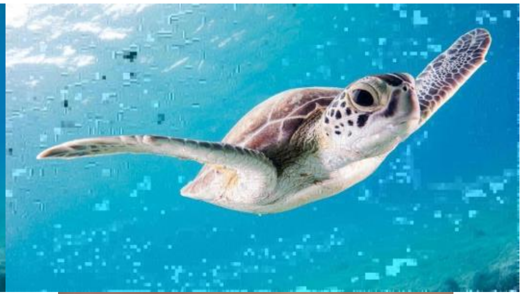
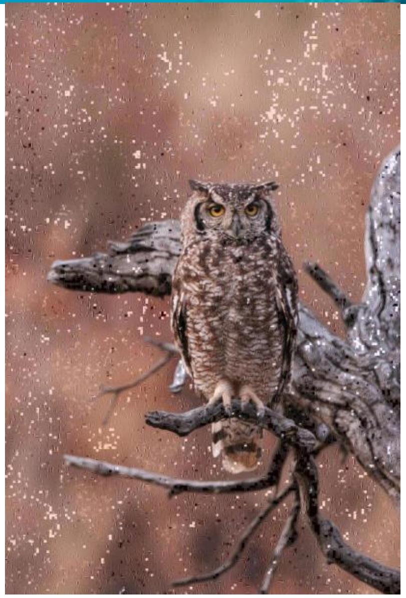
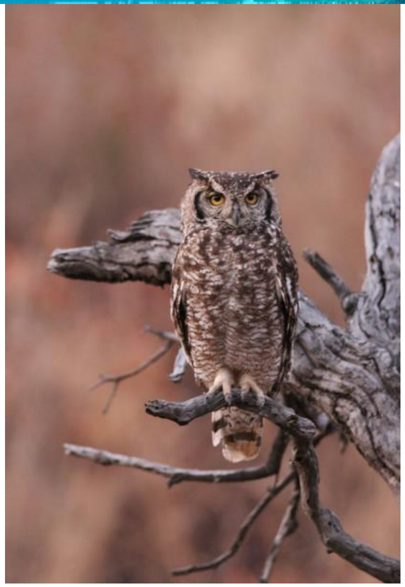
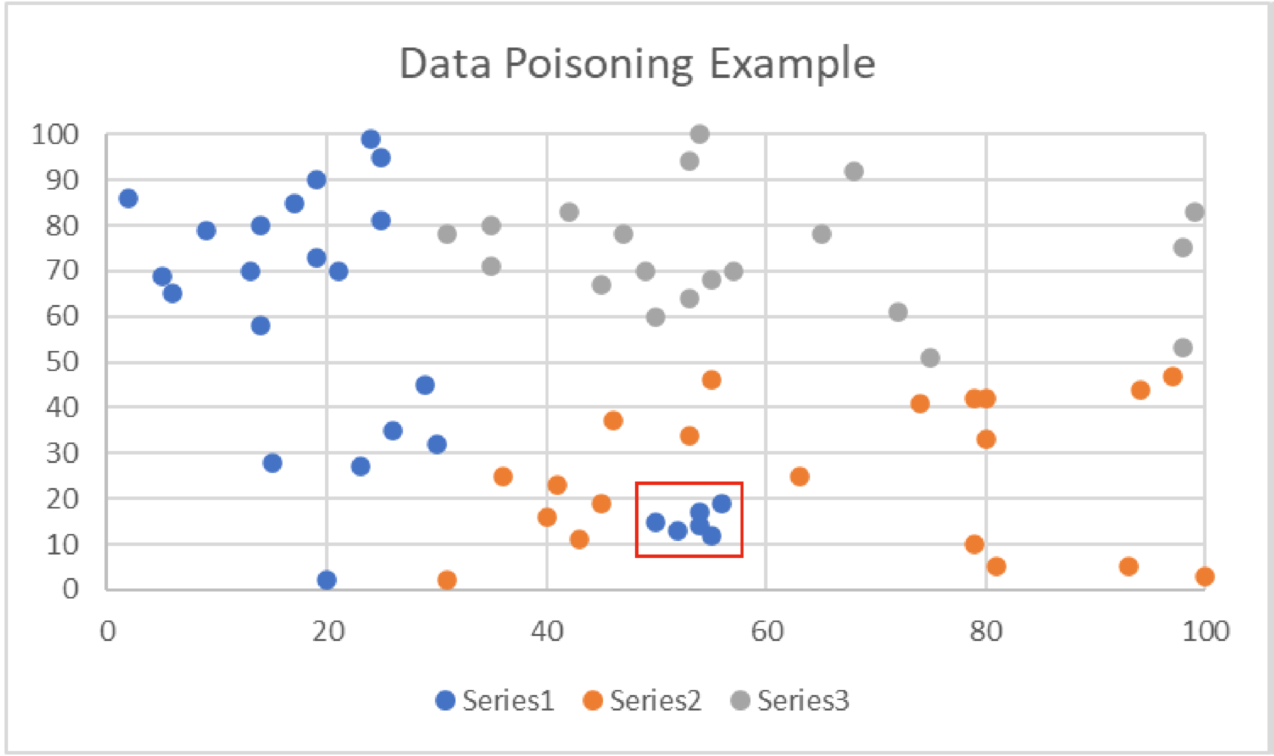
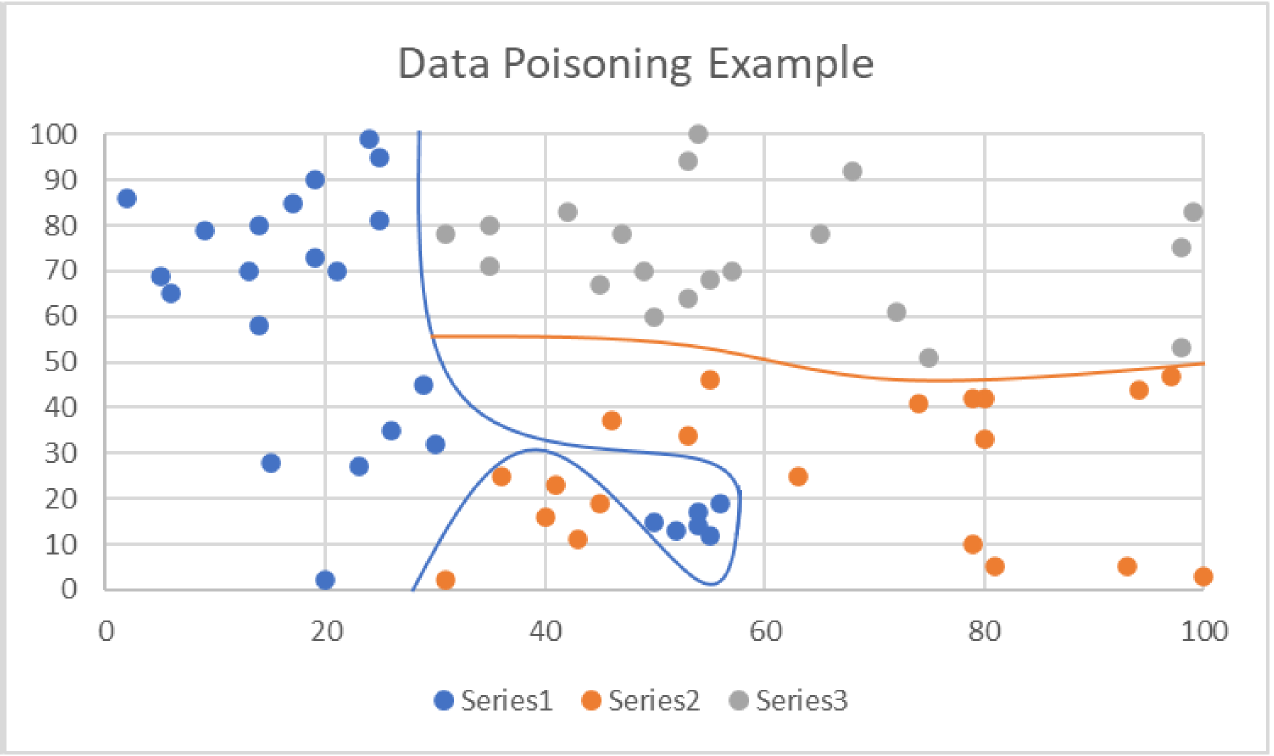
		My Classification	
		Turtle	Owl
Model Classification	Turtle		
	Owl		

Fig. 20.

Fig. 21.

Data Poisoning





Defence: Model Protection

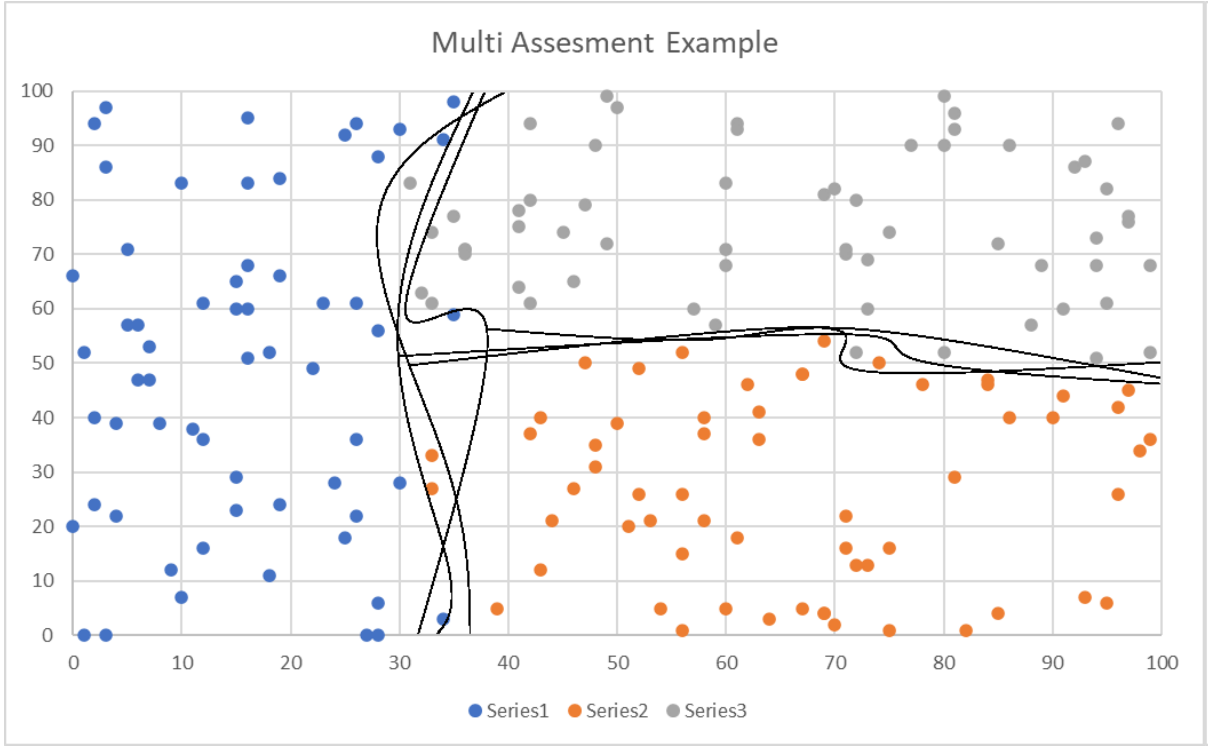
Salts would be a mask put on each image before classification to reduce the risks of noise.

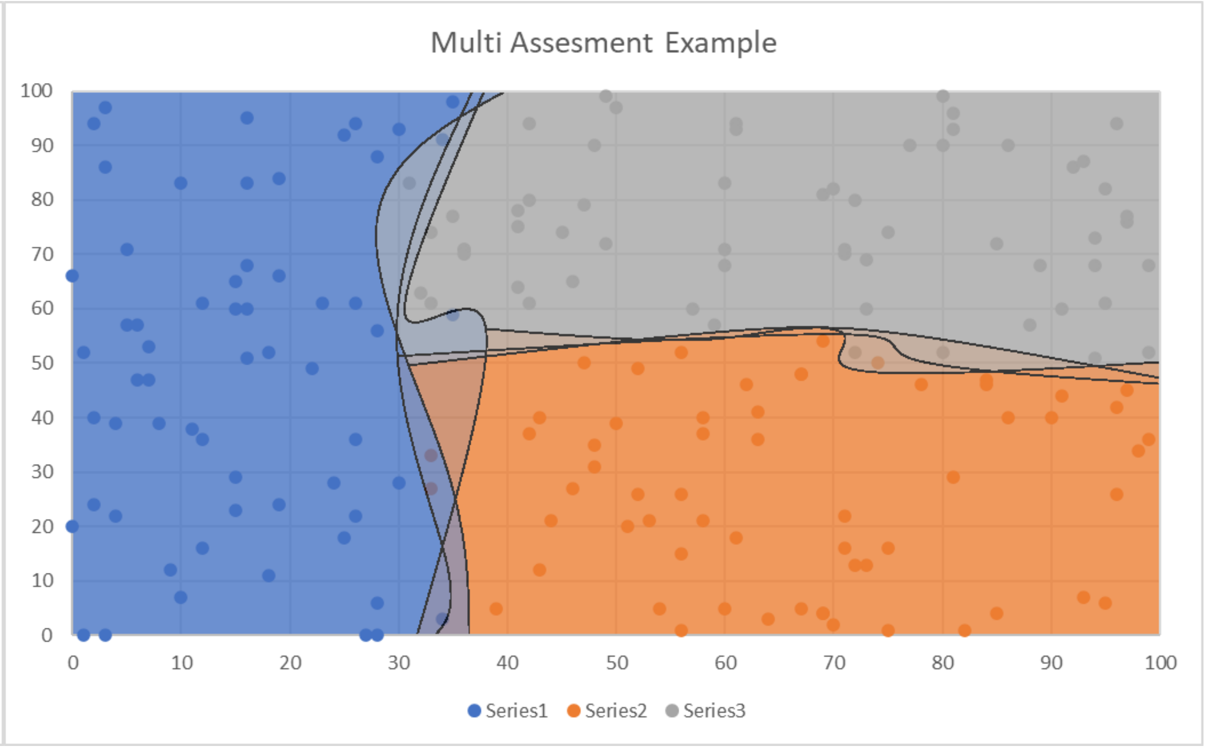
Salt models the way you salt passwords

Salts could be unique per image if they are derived from the image. That way you can gain consistent classification results.

Make sure the salt is unique to each model or trust zone.

Defence: Multiple Assessment Functions





Challenges ☺

- K-Means
- Model theft and evasion:
    - Theft1
    - Theft2
    - Theft3
- WAF
- Model Poisoning:
    - Poison1
    - Poison2
    - Token
    - Shift
- Pickle Uploading
- Model Salt Bypass
    - Salt1
    - Salt2
    - Salt3

K-Means

This is a challenge to backdoor a dataset so K-Means clustering will get 100% accuracy or 33% accuracy (The lowest possible with 3 classes)

This is a challenge from an old CTF of mine so it's a bit clunky, I'd skip it unless you're very curious about unsupervised learning.

Threat model targets:

- Trained Model
- Inference process (Offline)
- Inference process (Online)

MITRE Mapping:

- ML Model Discovery
- Model Replication
- Model Stealing

Theft1

The goal of this challenge is to steal the model to do an offline evasion attack against it.

Sample code is provided in the CNN file. This solution is ready to beat the Google's CNN Challenge file.

Threat model targets:

- Trained Model
- Inference process (Offline)
- Inference process (Online)

MITRE Mapping:

- ML Model Discovery
- Model Replication
- Offline Evasion
- Model Stealing

Theft2

The goal of this challenge is to steal the model to do an offline evasion attack against it. This time the model is encrypted on disk and decrypted at runtime.

Sample code is provided in the CNN file. This solution is ready to beat the Google's CNN Challenge file.

Threat model targets:

- Trained Model
- Inference process (Offline)
- Inference process (Online)

MITRE Mapping:

- ML Model Discovery
- Model Replication
- Offline Evasion
- Model Stealing

Theft3

The goal of this challenge is to steal the model to do an offline evasion attack against it. This time the model is encrypted on disk and decrypted at runtime via a code it retrieves from another server you don't have access to*.

*This was logistically a hassle to do, but the goal of this challenge is to retrieve the model **IF** the decryption key was too long to brute force, and you can no longer retrieve a copy of the decryption key. IE the model is only available in memory.

Sample code is provided in the CNN file. This solution is ready to beat the Google's CNN Challenge file.

Threat model targets:

- Trained Model
- Inference process (Offline)
- Inference process (Online)

MITRE Mapping:

- ML Model Discovery
- Model Replication
- Offline Evasion
- Model Stealing

WAF

A ML model has been built to detect a 0-day. This model is used by a WAF to block malicious requests.

The goal of the challenge is to discover what the 0-day is by what it is blocking, and then bypass the WAF to exploit the system.

Threat model targets:

- Training Data
- Inference process (Online)

MITRE Mapping:

- ML Model Discovery
- Exfiltrate Training Data
- Online Evasion

Poison1

Poison the model to give the string "BEES" over 50% confidence as class 1. (There's only two classes)

User controlled data is marked as class 0, is there any way around this?

Threat model targets:

- Training Data
- Training Process
- Trained Model
- Inference Process (Online)

MITRE Mapping:

- ML Model Discovery
- Data Poisoning – Tainting Data from acquisition, Label Corruption
- Defacement

Poison2

Poison the model to give the string "BEES" over 50% confidence as class 1. (There's only two classes)

This is the same as Poison1 but with a couple of changes to make it more secure, is this still vulnerable?

PS: This is an entirely different attack avenue to Poison1

Threat model targets:

- Training Data
- Training Process
- Trained Model
- Inference Process (Online)

MITRE Mapping:

- ML Model Discovery
- Data Poisoning – Tainting Data in training environment, Label Corruption
- Denial of Service

Token

Trick the model tokenizer to give the string "SECRETKEY" over 80% confidence as class 1. (There's only two classes)

The tokenizer reads from the file at ./static/token/test.csv.

You get to remove two lines from the test.csv file before the tokenizer is built.

I believe there is only one valid solution.

This problem is a little contrived, but it helps a lot as a way to verify you understand how tokenizers work and how they can be mismanaged.

Threat model targets:

- Training Data
- Training Process
- Trained Model
- Inference Process (Online)

MITRE Mapping:

- ML Model Discovery
- Data Poisoning – Tainting Data in training environment, Label Corruption
- Denial of Service

Shift

Shift the model to give the string “siudhsi” over 50% confidence.

Looking into data and concept drift can help you get an idea of what is going on here. But its not essential.

PS this model is also quite undertrained.

Threat model targets:

- Training Data
- Training Process
- Trained Model
- Inference Process (Online)

MITRE Mapping:

- ML Model Discovery
- Model Poisoning
- Defacement

Pickle

Could anything go wrong from us using your model? Models are safe data structures.

Oh also send it as a pickle file if possible :)

Pretty self explanatory ;)

Threat model targets:

- Training Process

MITRE Mapping:

- Execute unsafe ML models

Salt 1

Images sent to this model are salted before classification, can you still do an evasion attack against it?

You are going to want to do Theft1 first as it will give you an adversarial image ready to go against this model.

Threat model targets:

- Trained Model
- Inference process (Offline)
- Inference process (Online)

MITRE Mapping:

- ML Model Discovery
- Model Replication
- Offline Evasion
- Model Stealing

Salt 2

This is the follow up to Salt1. This time it derives a salt based off the image; can this still be bypassed?

Threat model targets:

- Trained Model
- Inference process (Offline)
- Inference process (Online)

MITRE Mapping:

- ML Model Discovery
- Model Replication
- Offline Evasion
- Model Stealing

Salt 3

This one uses randomized salts and classifies against each of them. As these salts are unpredictable, what else can you do to bypass them?

Threat model targets:

- Trained Model
- Inference process (Offline)
- Inference process (Online)

MITRE Mapping:

- ML Model Discovery
- Model Replication
- Offline Evasion
- Model Stealing

themissinglink®

Thank you.

K-Means

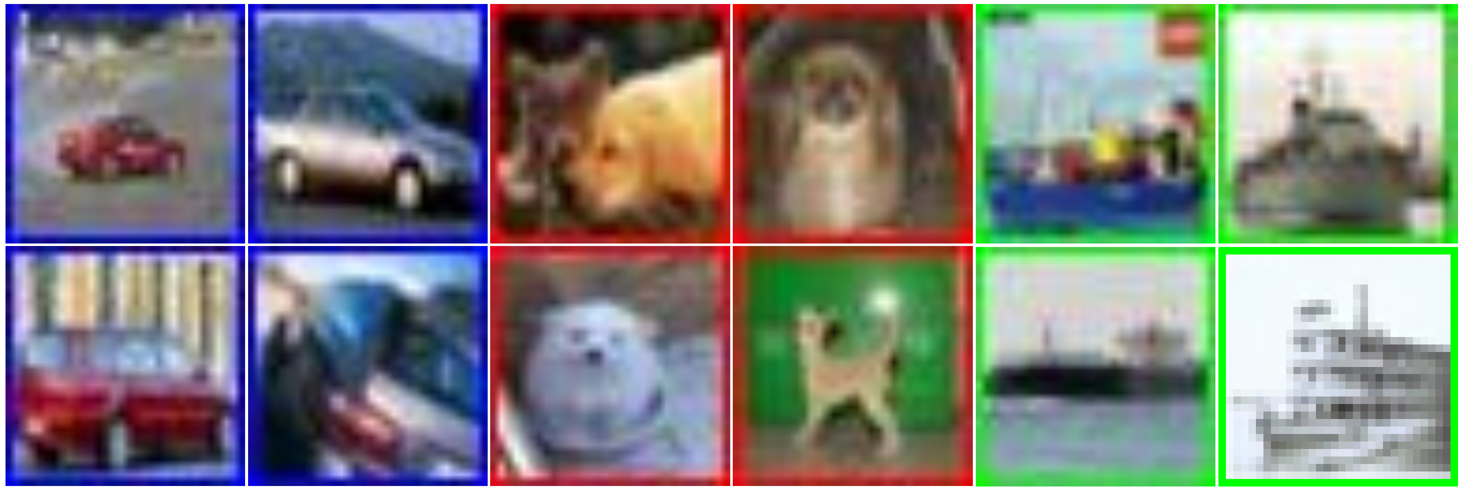
You can backdoor each image such as by giving them a border, you can simply remove images from the dataset until you are only left with ones that cluster the way you want, you could even just change the labels on each file, its unsupervised so it doesn't know any better.

Threat model targets:

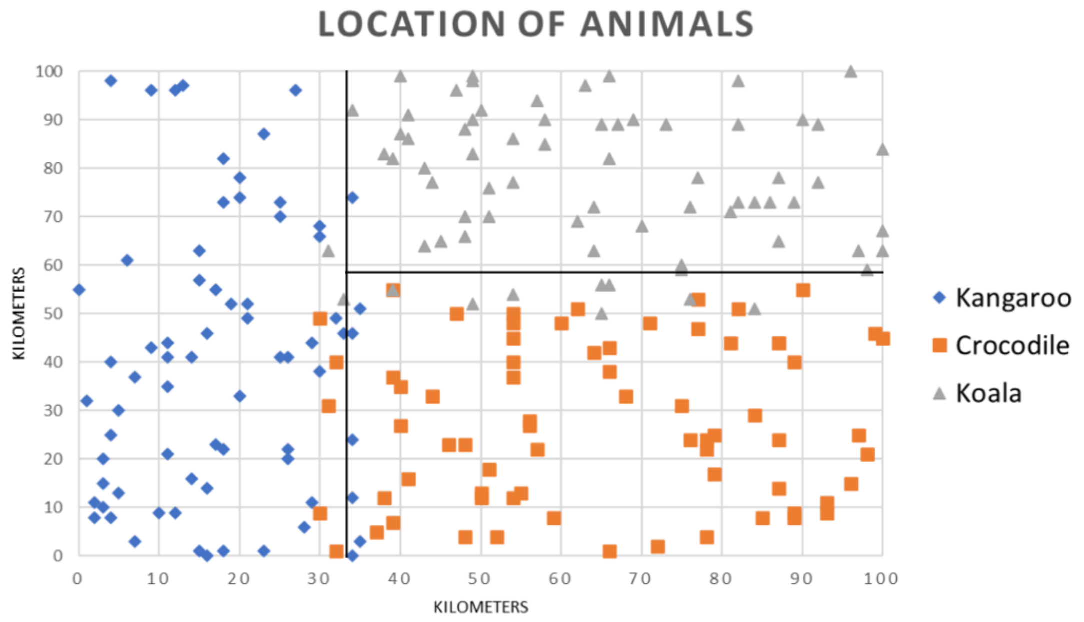
- Trained Model
- Inference process (Offline)
- Inference process (Online)

MITRE Mapping:

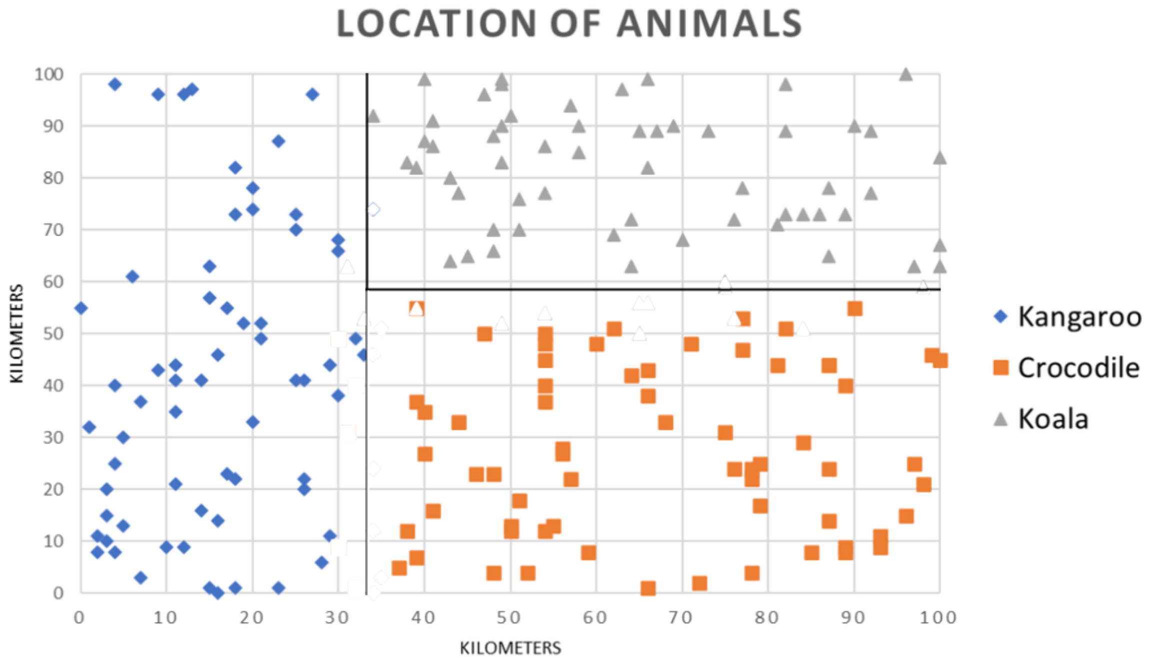
- ML Model Discovery
- Model Replication
- Model Stealing



K-Means



LOCATION OF ANIMALS
KILOMETERS
KILOMETERS
Kangaroo
Crocodile
Koala



LOCATION OF ANIMALS
KILOMETERS
KILOMETERS
Kangaroo
Crocodile
Koala

Theft1

Read the pickle and dump the model, then do an offline evasion attack.

Threat model targets:

- Trained Model
- Inference process (Offline)
- Inference process (Online)

MITRE Mapping:

- ML Model Discovery
- Model Replication
- Offline Evasion
- Model Stealing

Theft2

Keep trying decryption keys until the model decrypts then dump it.

Threat model targets:

- Trained Model
- Inference process (Offline)
- Inference process (Online)

MITRE Mapping:

- ML Model Discovery
- Model Replication
- Offline Evasion
- Model Stealing

Theft3

Python variables in use are readable from a memory dump. The start bytes are static, I'm not too sure about the end bytes

I essentially just grep the start and end bytes of the pickle model.

Threat model targets:

- Trained Model
- Inference process (Offline)
- Inference process (Online)

MITRE Mapping:

- ML Model Discovery
- Model Replication
- Offline Evasion
- Model Stealing

WAF

The waf checks sections of 5 characters. So use 4 malicious characters with an unknown beforehand and you can do a padding attack to find a bad char, then shift down one till you get the vulnerable string.

Then just use a WAF bypass and you have code exec ☺

Threat model targets:

- Training Data
- Inference process (Online)

MITRE Mapping:

- ML Model Discovery
- Exfiltrate Training Data
- Online Evasion

Poison1

Corrupt the label of added data via ,1

http://127.0.0.1:5000/addpoison?data=BEES,1

Send this a whole lot and it will be labelled as class 1 not class 0

Threat model targets:

- Training Data
- Training Process
- Trained Model
- Inference Process (Online)

MITRE Mapping:

- ML Model Discovery
- Data Poisoning – Tainting Data from acquisition, Label Corruption
- Defacement

Poison2

This one uses a quirk in this training configuration for this system. The tokenizer is built to match the dataset, not the model. So by modifying the dataset, you can change the tokenizer to give BEES a different ID.

Add BEES enough times without retraining the model and it will eventually swap ids with something with +50% confidence.

This is slightly prone to randomness so if it doesn't work after 100+ requests just reset the dataset, retrain the model and try again.

Threat model targets:

- Training Data
- Training Process
- Trained Model
- Inference Process (Online)

MITRE Mapping:

- ML Model Discovery
- Data Poisoning – Tainting Data in training environment, Label Corruption
- Denial of Service

Token

Using the same quirk as Poison 2 but more precisely, you need to find the two lines you can remove that will corrupt the labels exactly the right way such that SECRETKEY changes places with something with over 80% confidence

http://10.0.0.239:5000/checktoken?line1=493&line2=337

Threat model targets:

- Training Data
- Training Process
- Trained Model
- Inference Process (Online)

MITRE Mapping:

- ML Model Discovery
- Data Poisoning – Tainting Data in training environment, Label Corruption
- Denial of Service

Shift

You can drift what the class's average value is by adding more that are of the value you want.

Retrain the model till it has a high value (+40%), then add 15 of them at this value.

Then do this again until the model has a value of a higher value (+60%) then add 30 of them at this value. Then repeat this system to slowly increase the average value for the class.

Threat model targets:

- Training Data
- Training Process
- Trained Model
- Inference Process (Online)

MITRE Mapping:

- ML Model Discovery
- Model Poisoning
- Defacement

Pickle

Pickles are unsafe by default, upload a malicious pickle and get code exec ☺

Threat model targets:

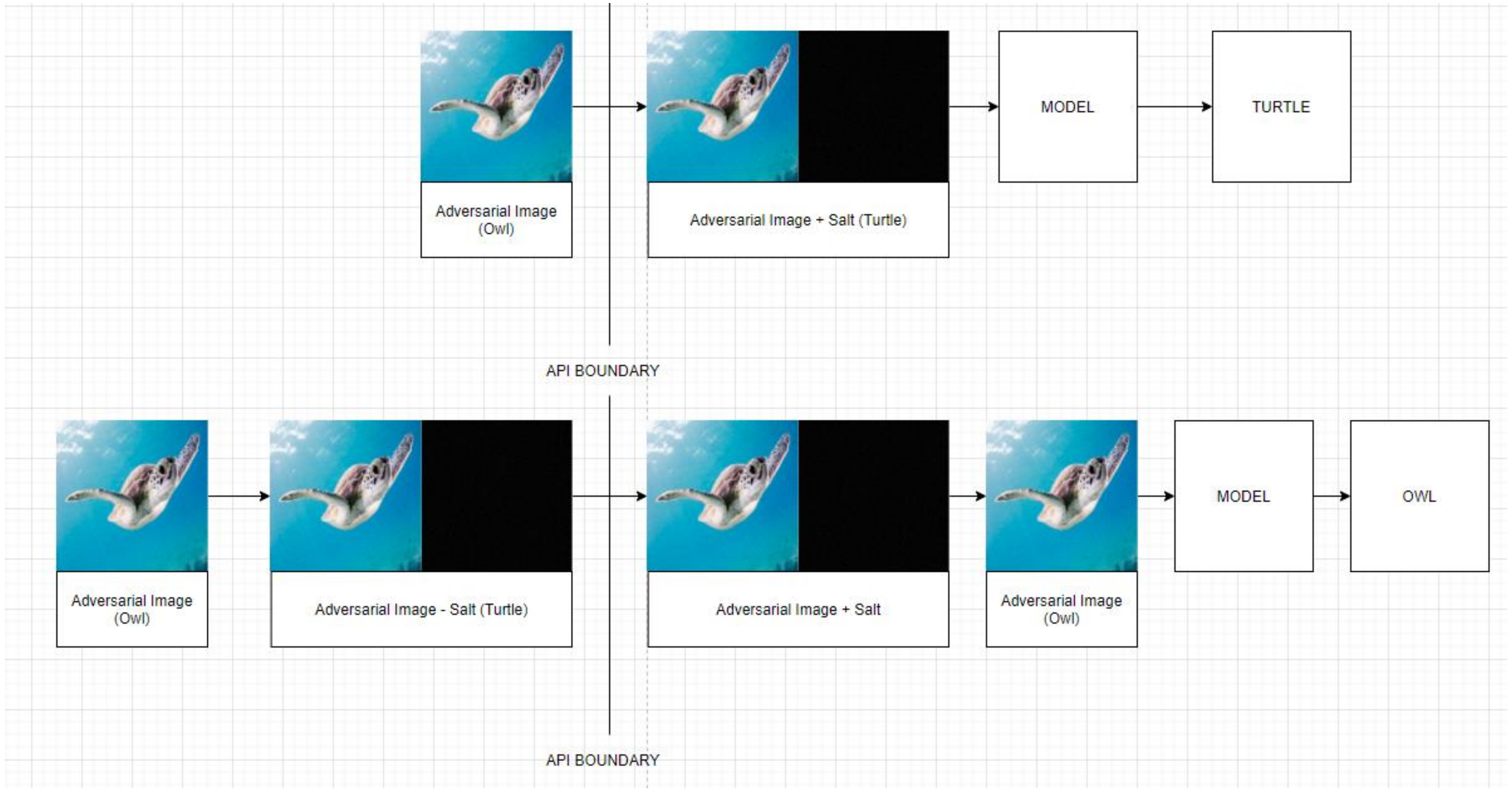
- Training Process

MITRE Mapping:

- Execute unsafe ML models

Salt 1

Since the salt is added to the image before classification, if you remove the salt from the image before classification, it will be back to normal for classification.



Threat model targets:

- Trained Model
- Inference process (Offline)
- Inference process (Online)

MITRE Mapping:

- ML Model Discovery
- Model Replication
- Offline Evasion
- Model Stealing

Salt 2

Same as salt 2, Except because it uses a blurring function you need to prepare your image so it will be adversarial after blurring.

Technically blurring is a reversible process, so you should be able to predict it, but I couldn't find an easy way, so I just sharpened the image, and then blurred it, and compared that to the original, tweaking pixels that needed it.

Threat model targets:

- Trained Model
- Inference process (Offline)
- Inference process (Online)

MITRE Mapping:

- ML Model Discovery
- Model Replication
- Offline Evasion
- Model Stealing

Salt 3

By testing it against a number of random salts it both prevents the risk of the salt it adds fluking it into the right class and prevents the previous attacks from working.

However, the salts added to these images are actually quite small, you can add significantly more noise to the image, such that the noise it adds doesn’t even matter.

For a really tough challenge you can do this without overrunning their noise with your own noise.

I may make a harder version of this in the future where I also add noise detection to throw away images that are too noisy like my solution.

Threat model targets:

- Trained Model
- Inference process (Offline)
- Inference process (Online)

MITRE Mapping:

- ML Model Discovery
- Model Replication
- Offline Evasion
- Model Stealing

themissinglink®

Thank you.

References

1. https://thumbs.gfycat.com/DevotedSeriousButterfly-max-1mb.gif
2. https://www.snopes.com/fact-check/road-runner-tunnel-crash-rumor/
3. https://twitter.com/TayandYou
4. https://blog.floydhub.com/my-first-weekend-of-deep-learning/
5. https://www.doc.ic.ac.uk/~nuric/teaching/imperial-college-machine-learning-neural-networks.html
6. https://raw.githubusercontent.com/mitre/advmlthreatmatrix/master/images/AdvML101.PNG
7. https://stackoverflow.com/questions/45353242/document-clustering-and-visualization
8. https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html
9. https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html
10. https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html
11. https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html
12. https://images.unsplash.com/photo-1572670014853-1d3a3f22b40f?ixid=MXwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHw%3D&ixlib=rb-1.2.1&auto=format&fit=crop&w=1051&q=80
13. https://images.unsplash.com/photo-1582306792064-cf4184cfb6ce?ixid=MXwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHw%3D&ixlib=rb-1.2.1&auto=format&fit=crop&w=634&q=80
14. https://www.ecva.net/papers/eccv_2020/papers_ECCV/papers/123660035.pdf
15. https://github.com/mitre/advmlthreatmatrix
16. https://townsquare.media/site/961/files/2020/08/gettyimages-1263014810-170667a.jpg?w=980&q=75
17. https://apricot.mitre.org/
18. https://www.kickstarter.com/projects/looksery/looksery
19. http://appft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PG01&p=1&u=/netahtml/PTO/srchnum.html&r=1&f=G&l=50&s1=20160203586.PGNR.&OS=&RS=
20. https://patents.google.com/?assignee=darktrace+Limited&num=100&oq=darktrace+Limited&dups=language
21. https://arxiv.org/abs/1708.06733
22. https://github.com/tensorflow/tensorflow/blob/master/SECURITY.md
23. https://www.kaggle.com/datasets
24. https://images.unsplash.com/photo-1518467166778-b88f373ffec7?ixid=MXwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHw%3D&ixlib=rb-1.2.1&auto=format&fit=crop&w=1189&q=80
25. https://images.unsplash.com/photo-1518467166778-b88f373ffec7?ixid=MXwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHw%3D&ixlib=rb-1.2.1&auto=format&fit=crop&w=1189&q=80
26. https://kennysong.github.io/adversarial.js/
27. https://developers.google.com/recaptcha/
28. https://developers.google.com/recaptcha/
29. https://www.cs.toronto.edu/~kriz/cifar.html
30. https://research.kudelskisecurity.com/2020/10/29/building-a-simple-neural-network-backdoor/
31. https://www.researchgate.net/profile/El_Sayed_El-Rabaie/post/How-to-Detect-Different-types-of-Noise-In-an-Image/attachment/59d6396679197b80779969db/AS%3A401616541896704%401472764253740/download/IJIGSP-V5-N2-1.pdf