Supervised problems

Classification & Regression

Given a dataset with label, find a function that can assign labels to new unlabelled data

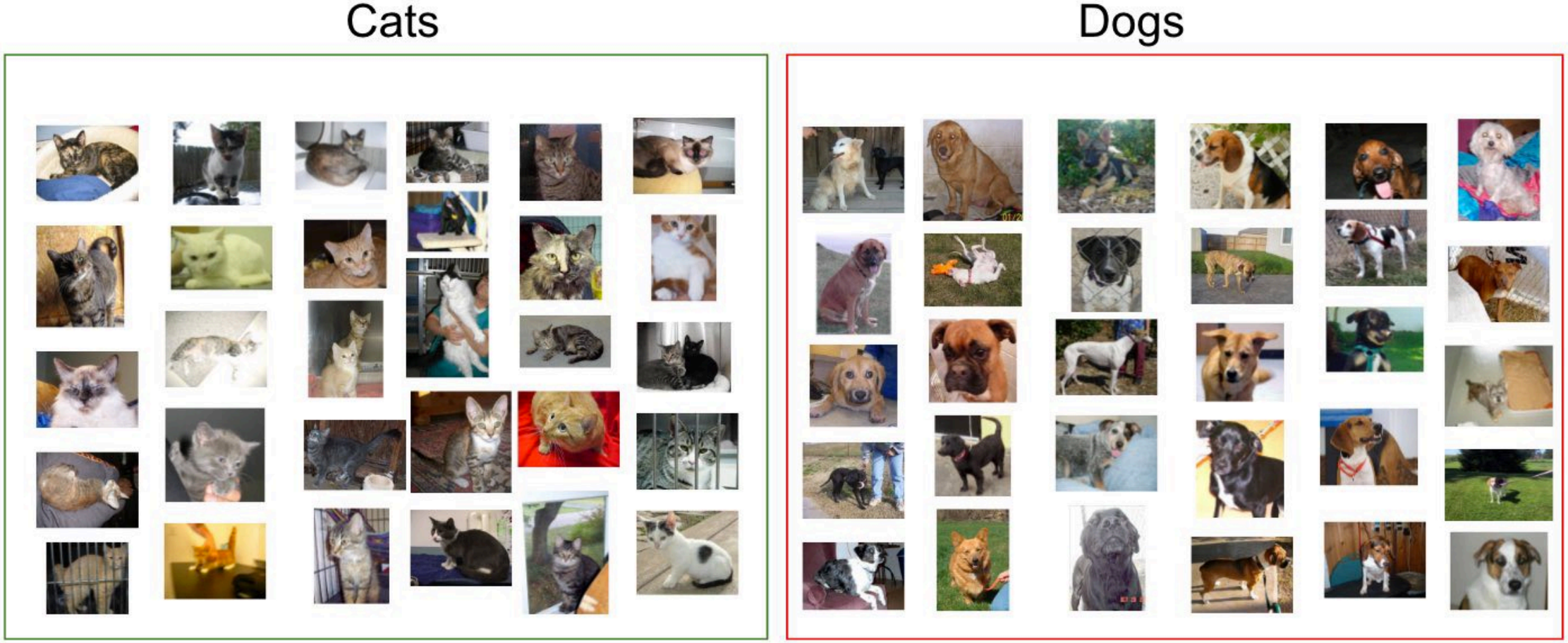
Regression

Classification

Labeled data: $\begin{cases} \{\vec{x}_1, \vec{x}_2, \vec{x}_3, \ldots, \vec{x}_n\} & \vec{x} \in \mathbb{R}^d \\ \{y_1, y_2, y_3, \ldots, y_n\} & y \in \mathbb{R} \quad \text{or} \quad \mathrm{y} \in \mathbb{N} \end{cases}$

Goal: Find a function $f_W(\vec{x})$ that outputs the right class/value for an object $\vec{x}$

Cats vs dogs classification

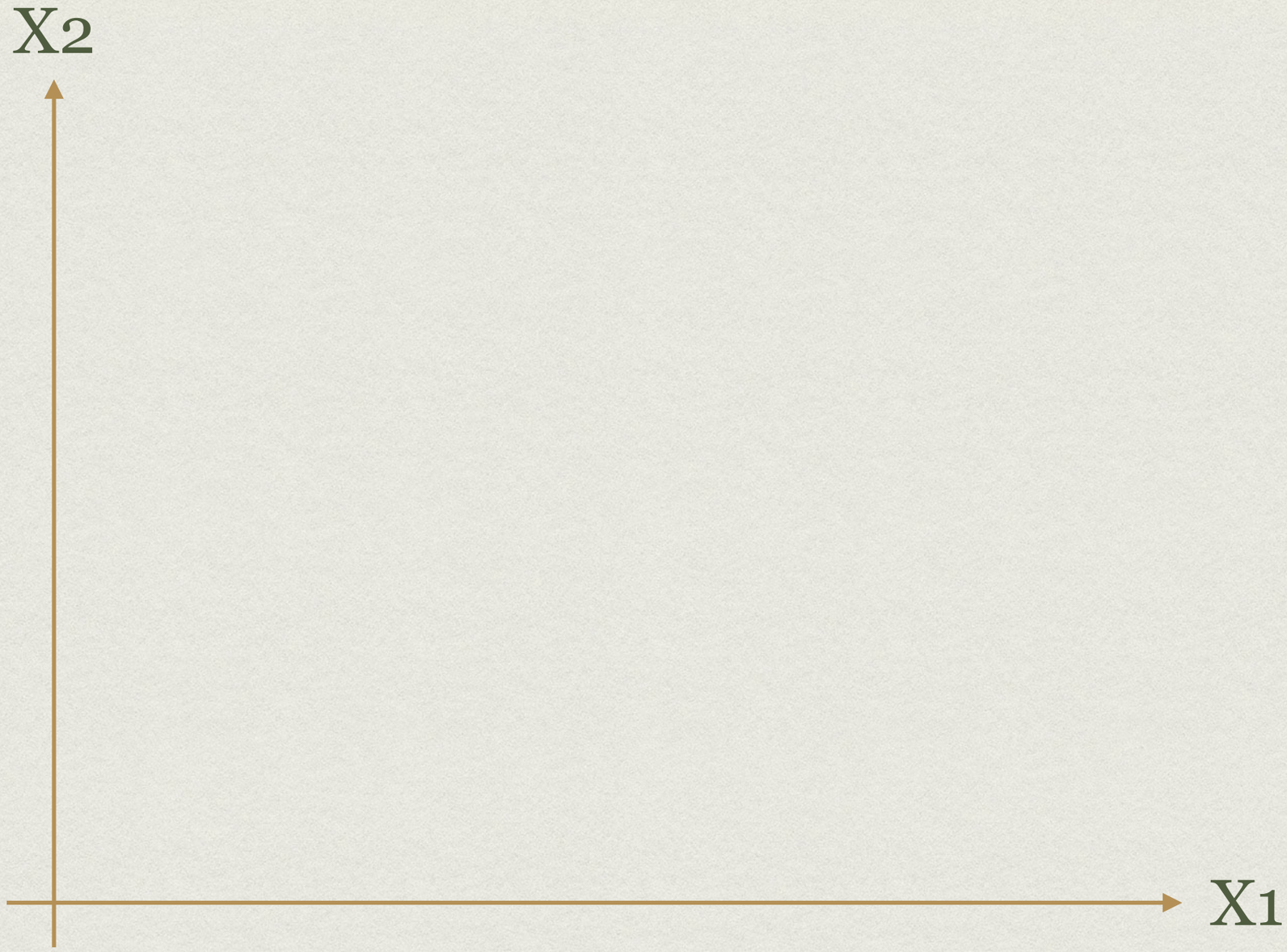


Sample of cats & dogs images from Kaggle Dataset

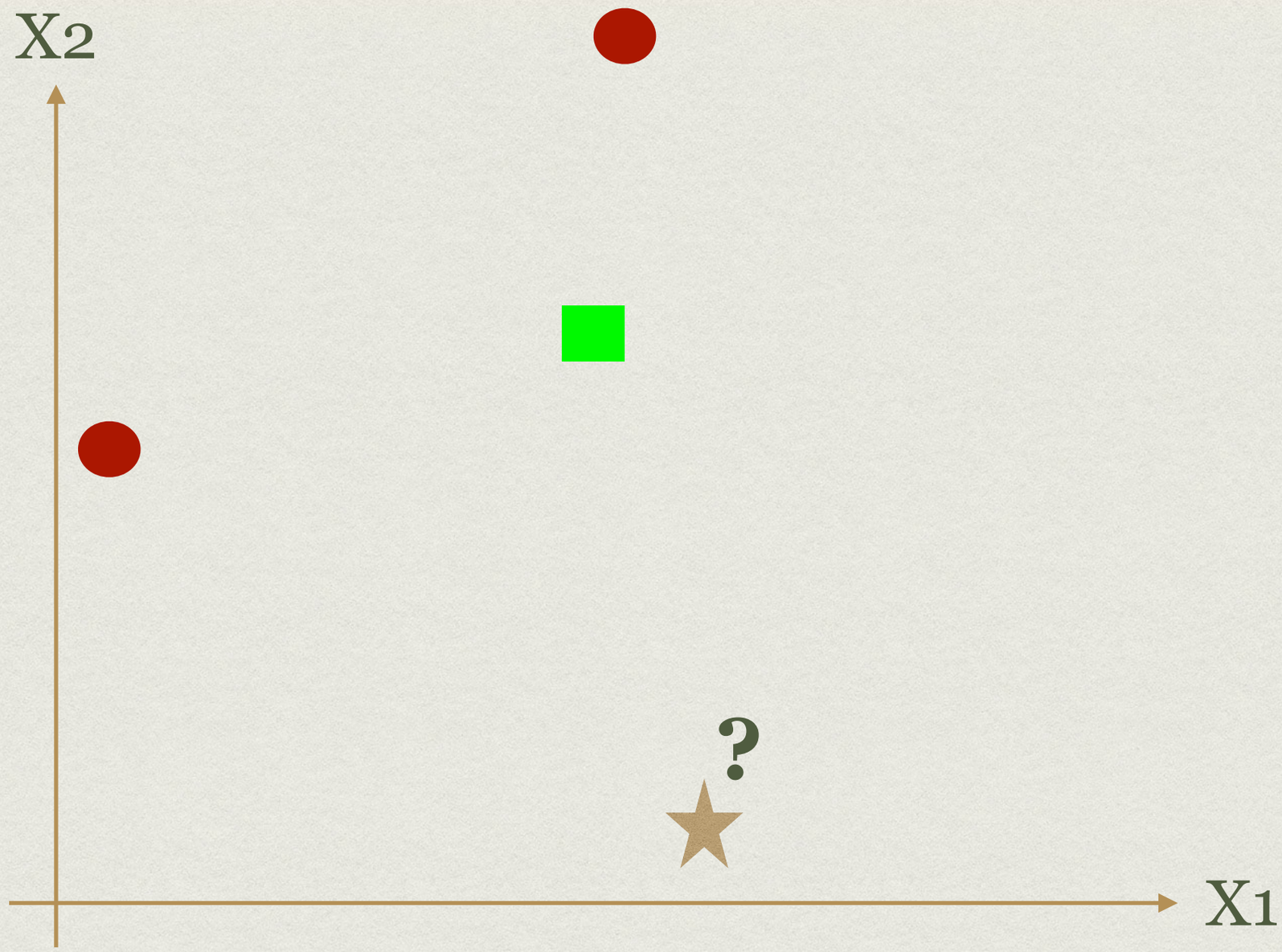
?

0 = dog
1 = cat

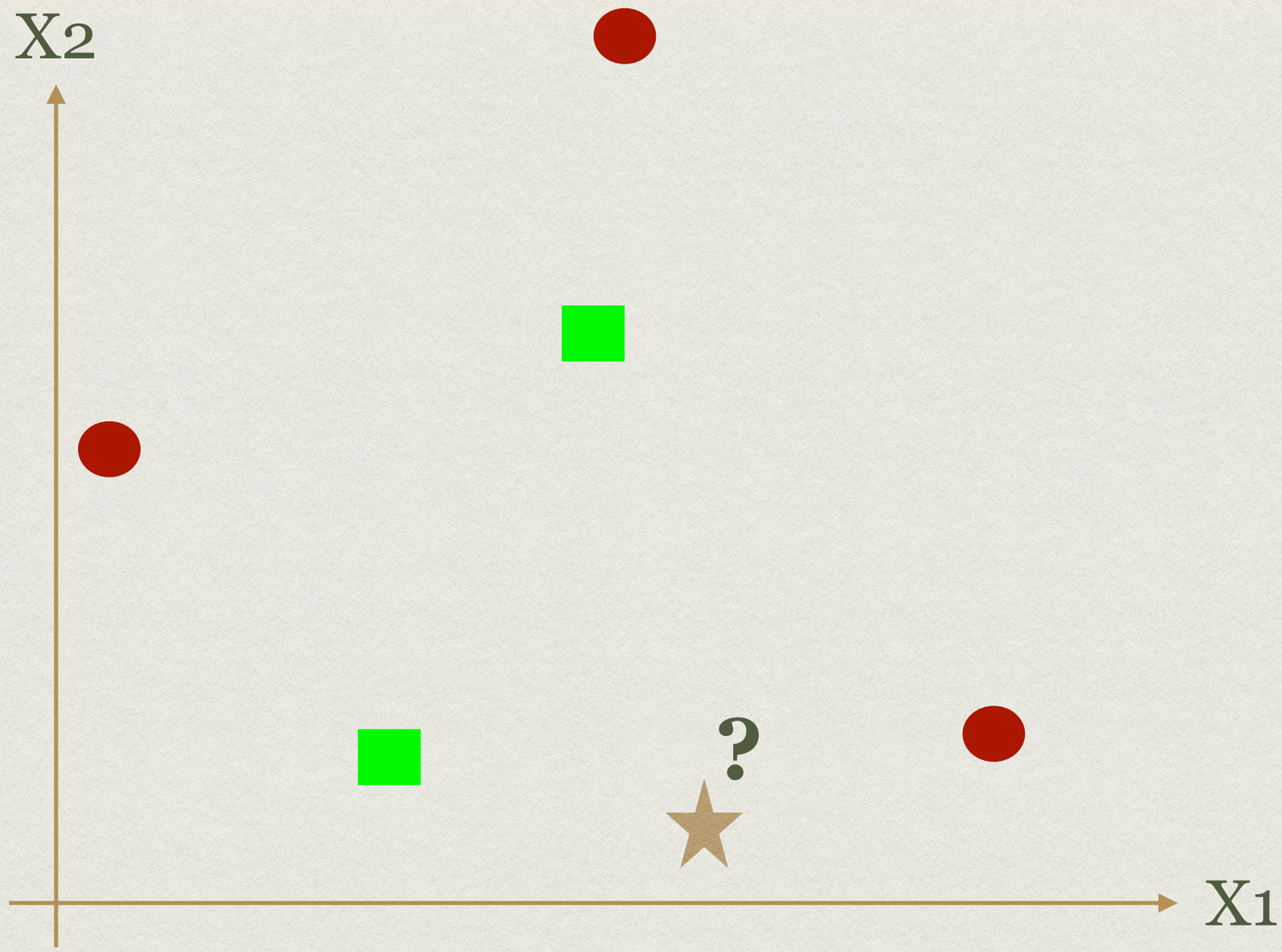
WHAT IS THE RULE?



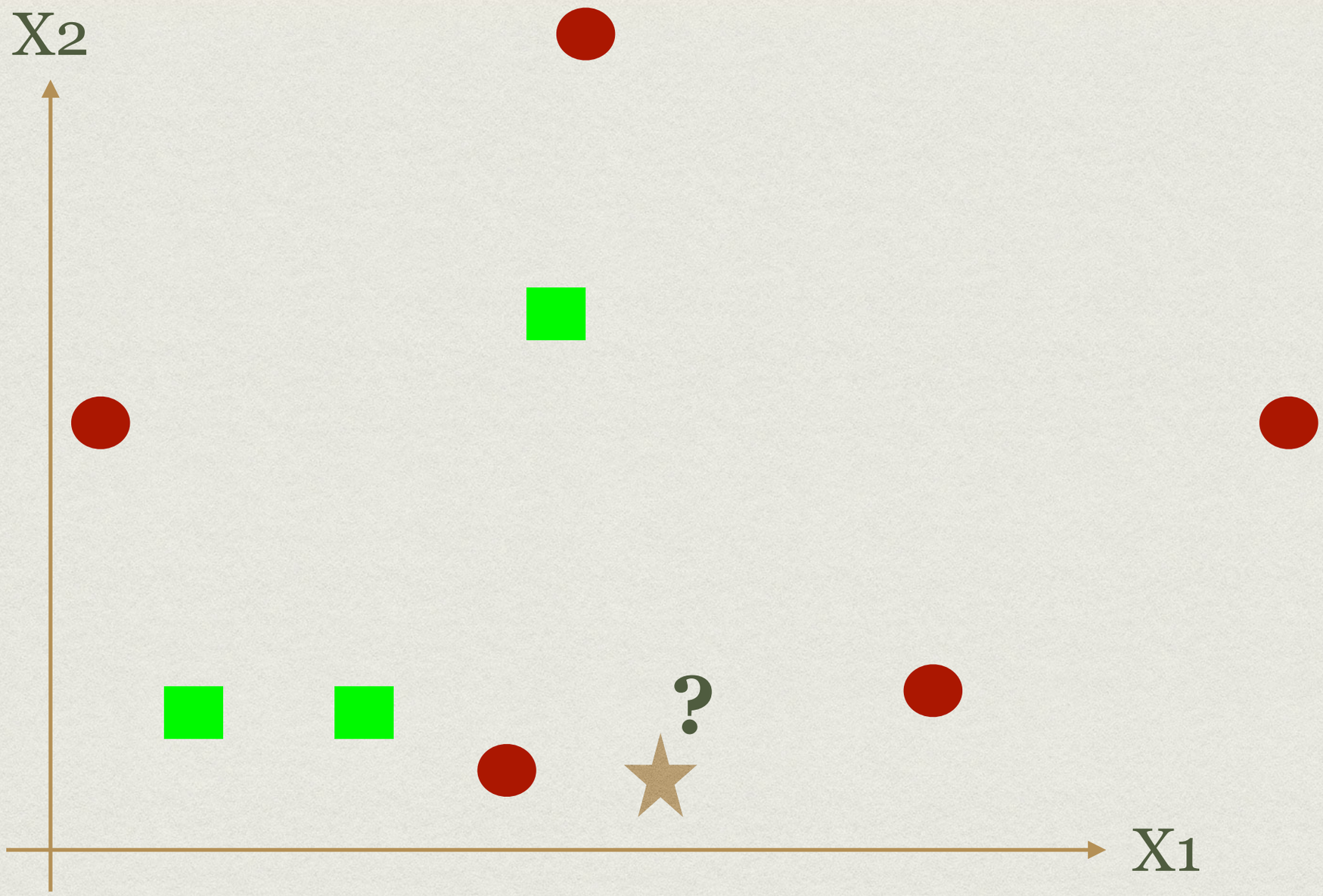
WHAT IS THE RULE?



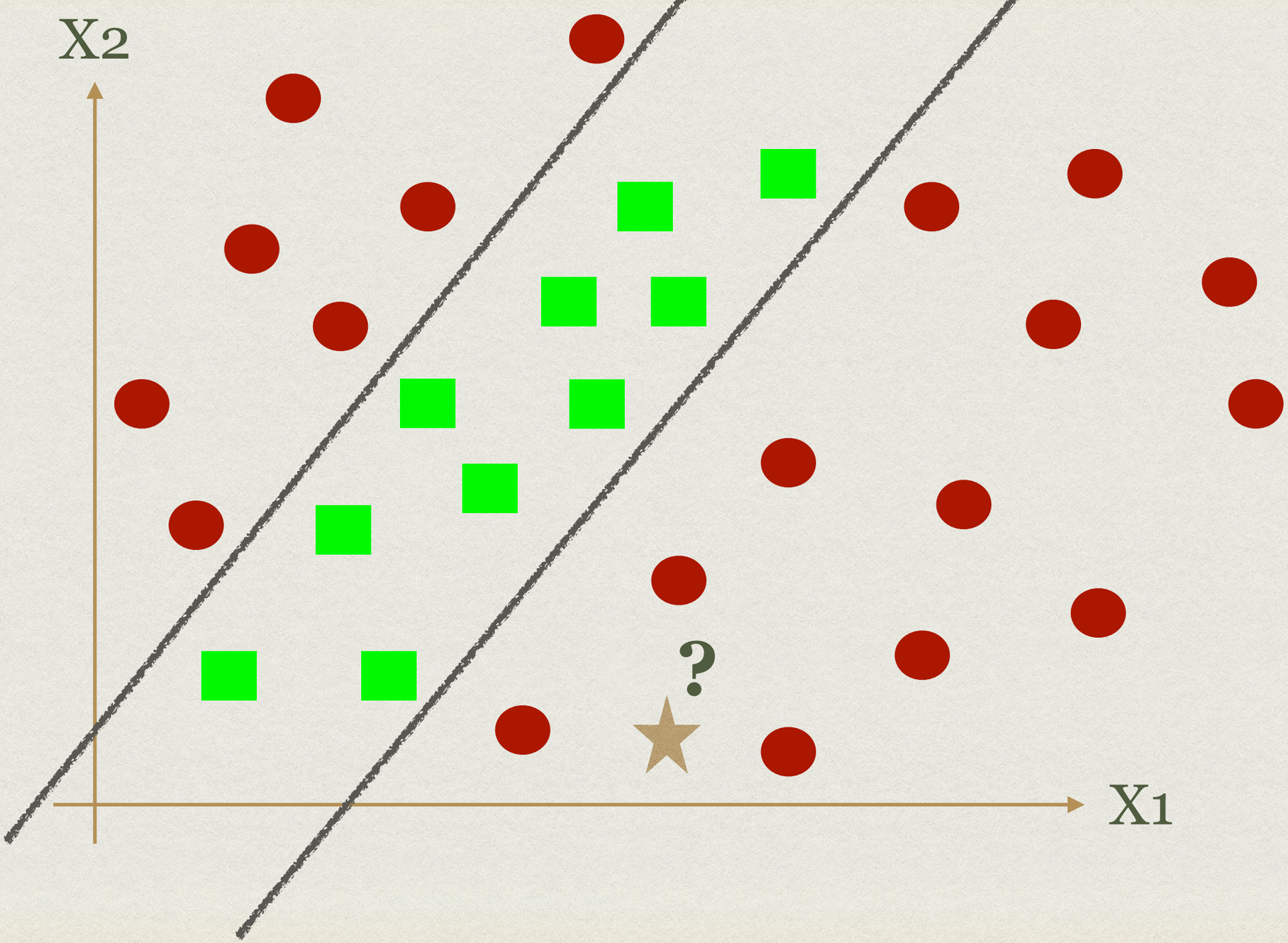
WHAT IS THE RULE?



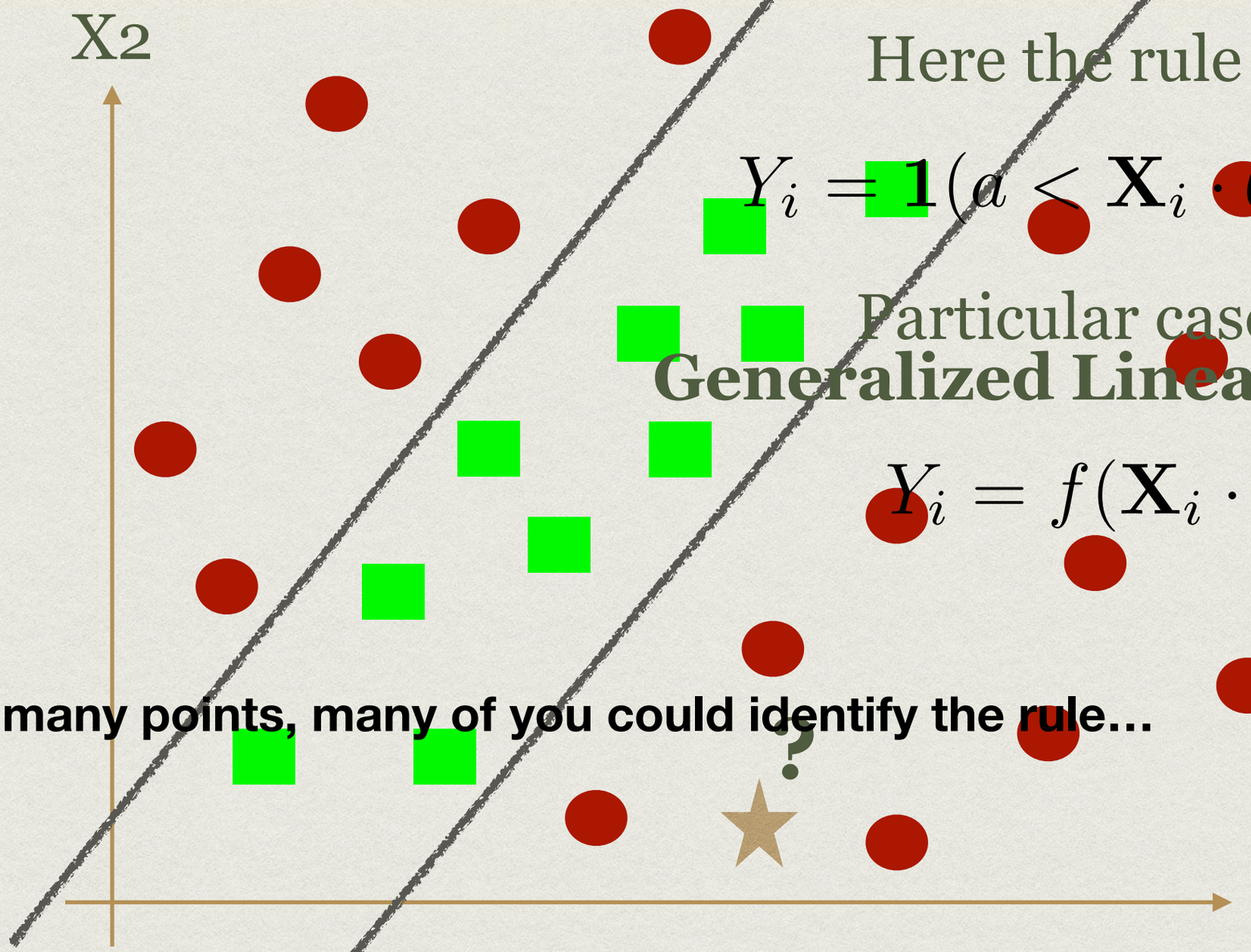
WHAT IS THE RULE?



WHAT IS THE RULE?



WHAT IS THE RULE?



Here the rule is

$$Y_i = \mathbf{1}(a < \mathbf{X}_i \cdot \theta < b)$$

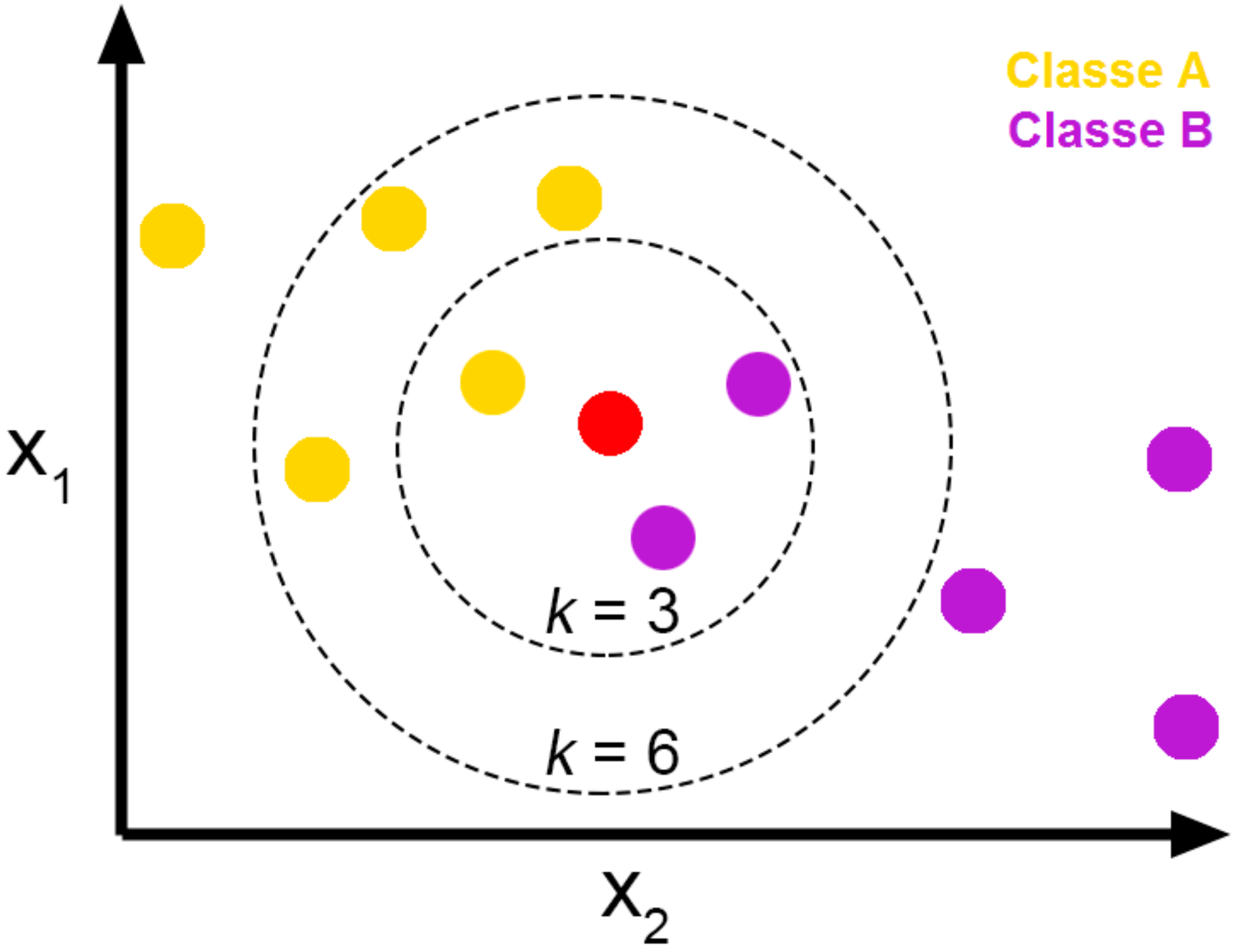
Particular case of

Generalized Linear Models

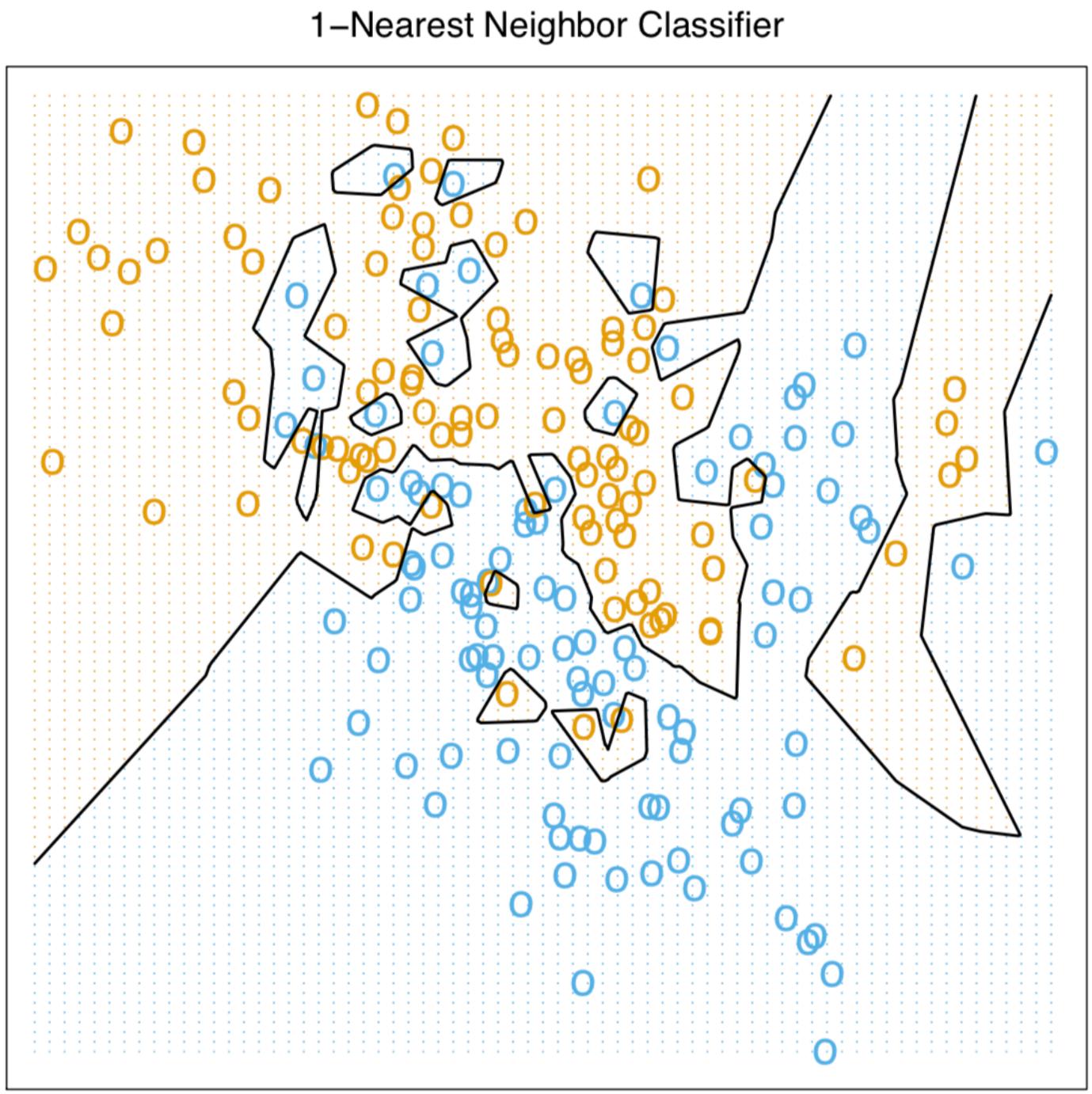
$$Y_i = f(\mathbf{X}_i \cdot \theta)$$

With many points, many of you could identify the rule…

… but before this, you probably used the nearest neighbour algorithm


Classe A
Classe B
x_1
$k = 3$
$k = 6$
x_2

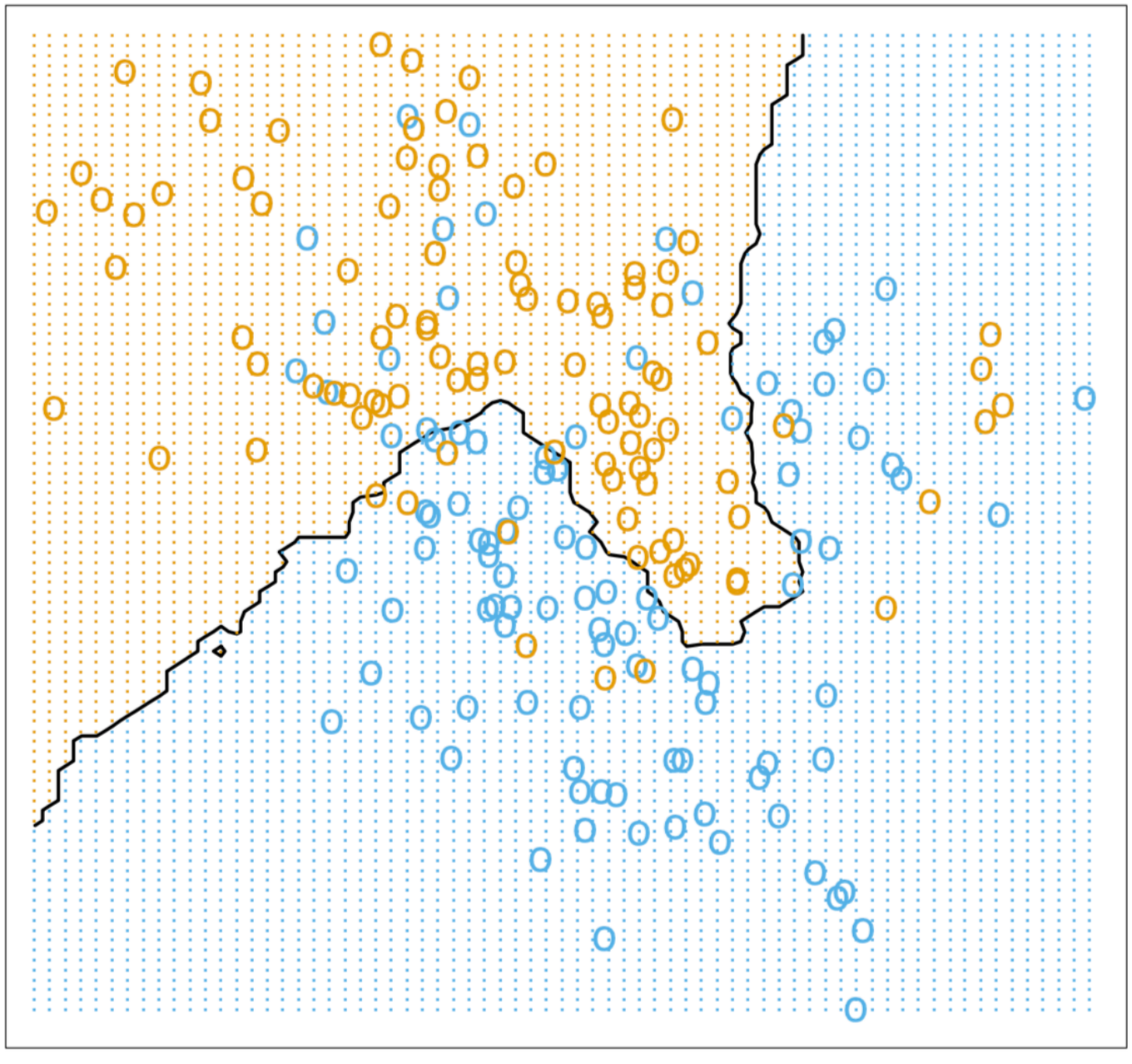
Nearest-Neighbour classifier: Example



- Memorize all labels
- Predict the label of the most similar training point

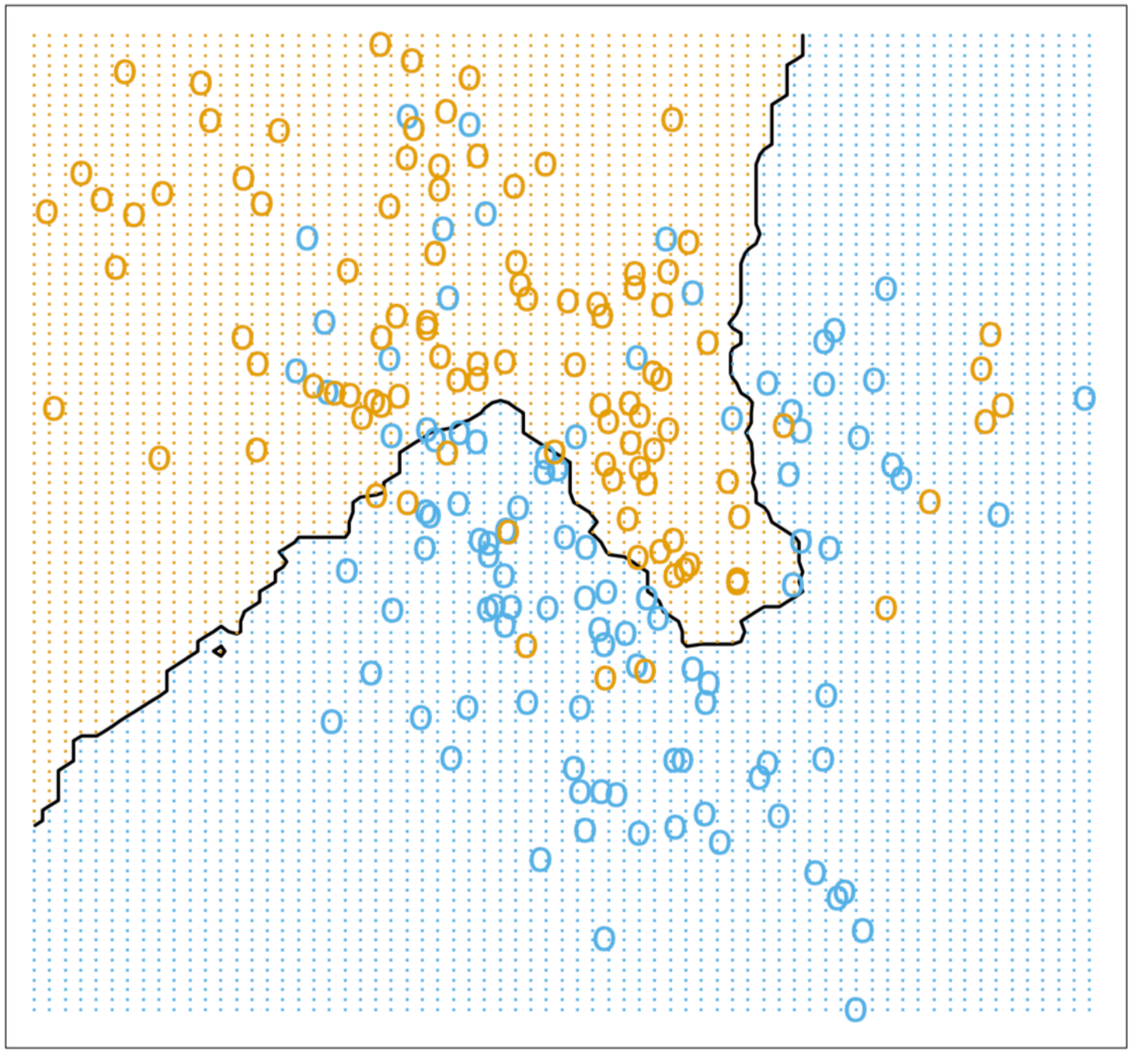
(from Hastie, Tibshirani, Friedman)

Nearest-Neighbour classifier: Example



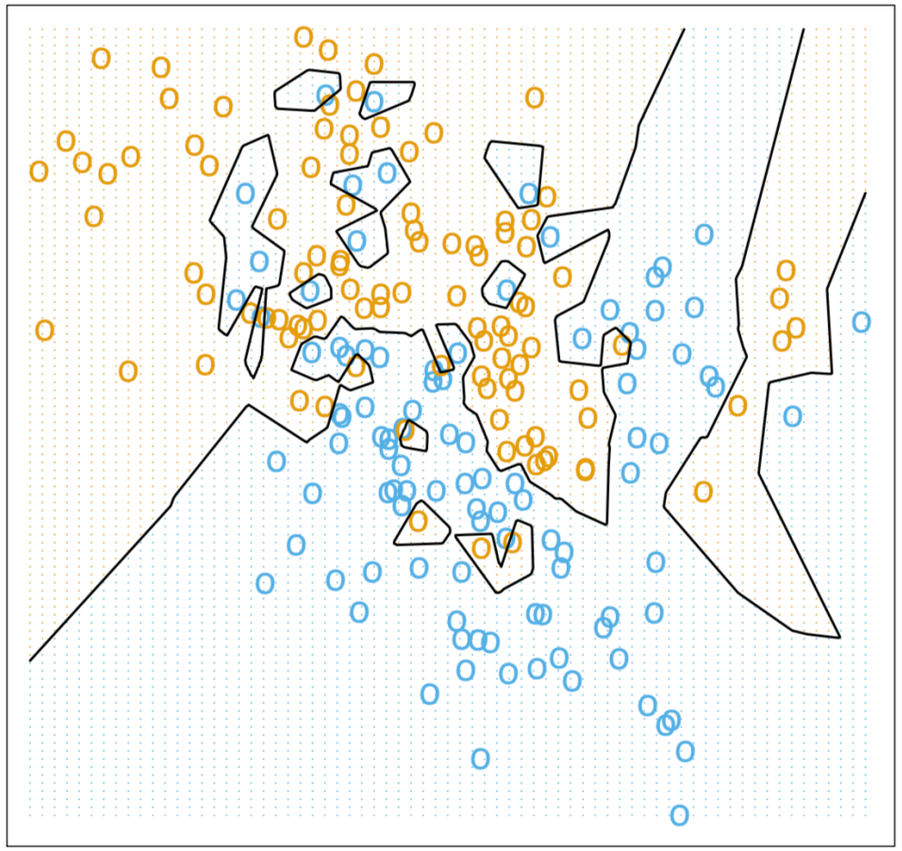
- Memorize all labels
- Predict the label averaging over many similar training points

How to choose k ?

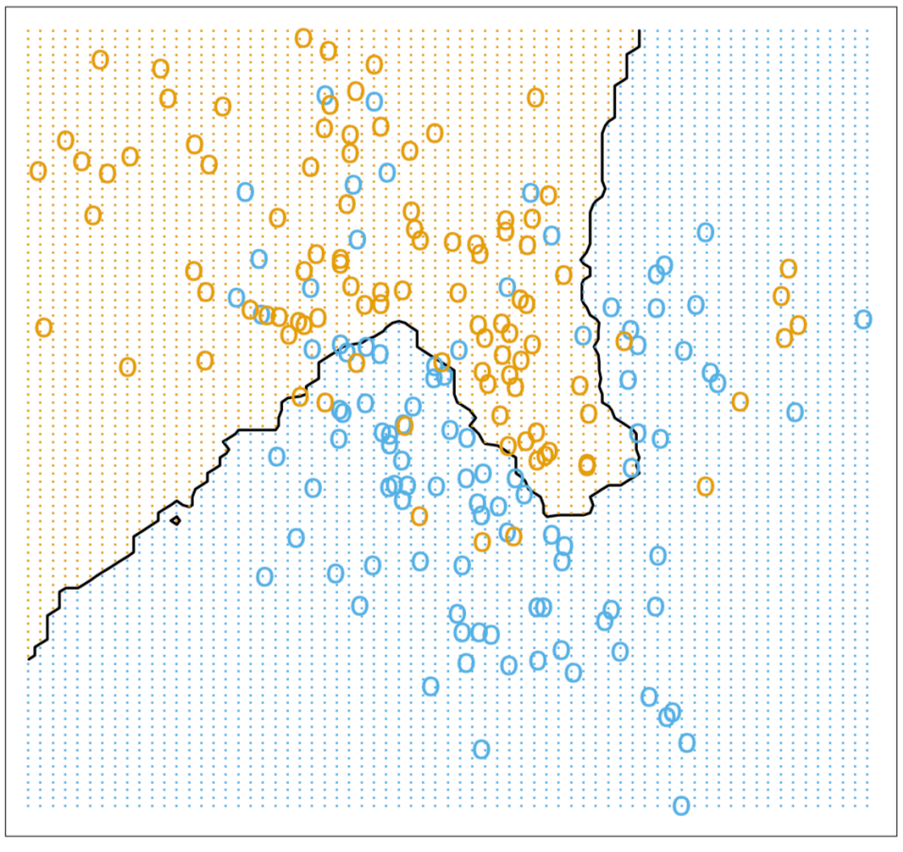


- Memorize all labels
- Predict the label averaging over many similar training points

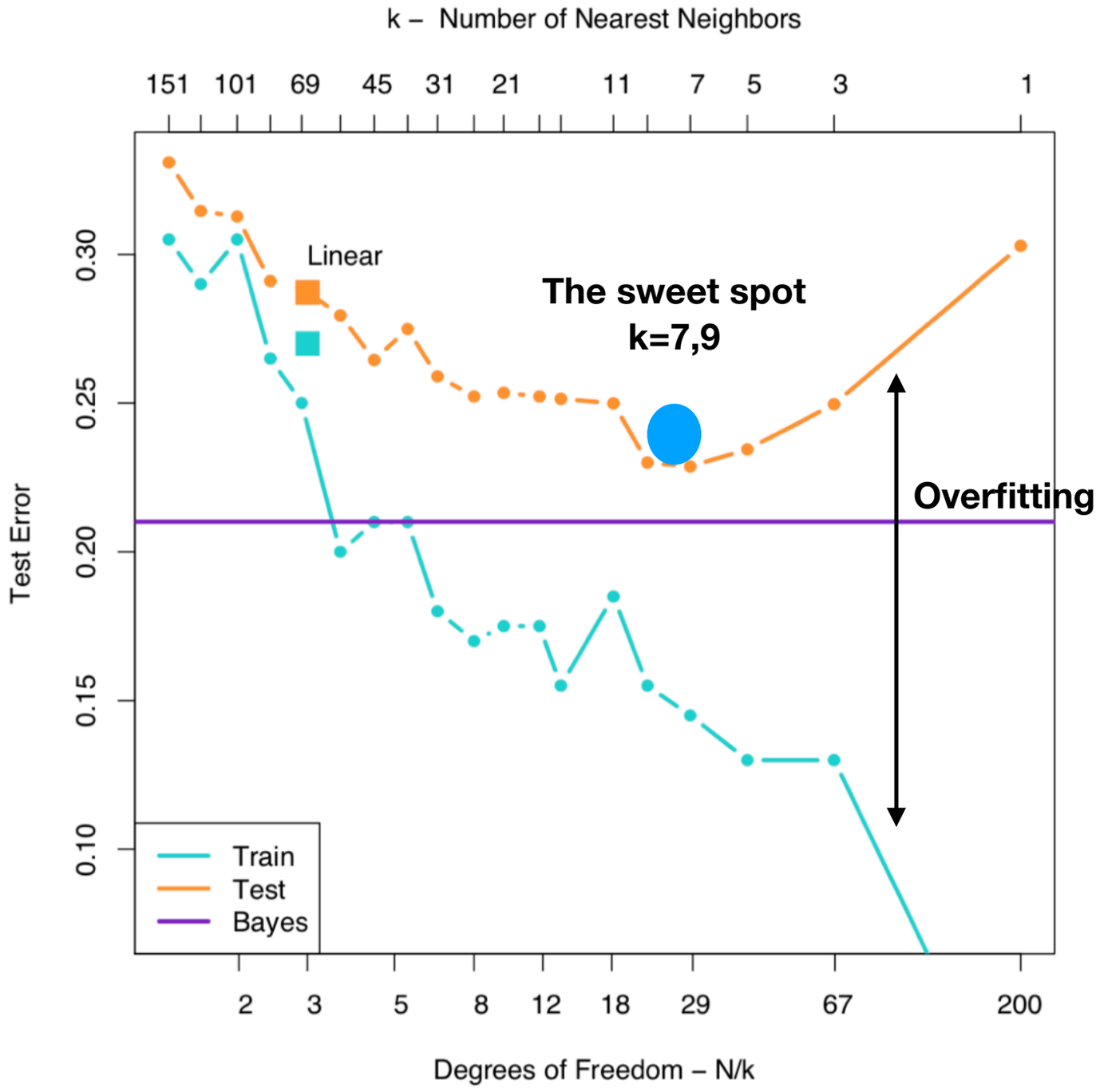
1–Nearest Neighbor Classifier



15-Nearest Neighbor Classifier

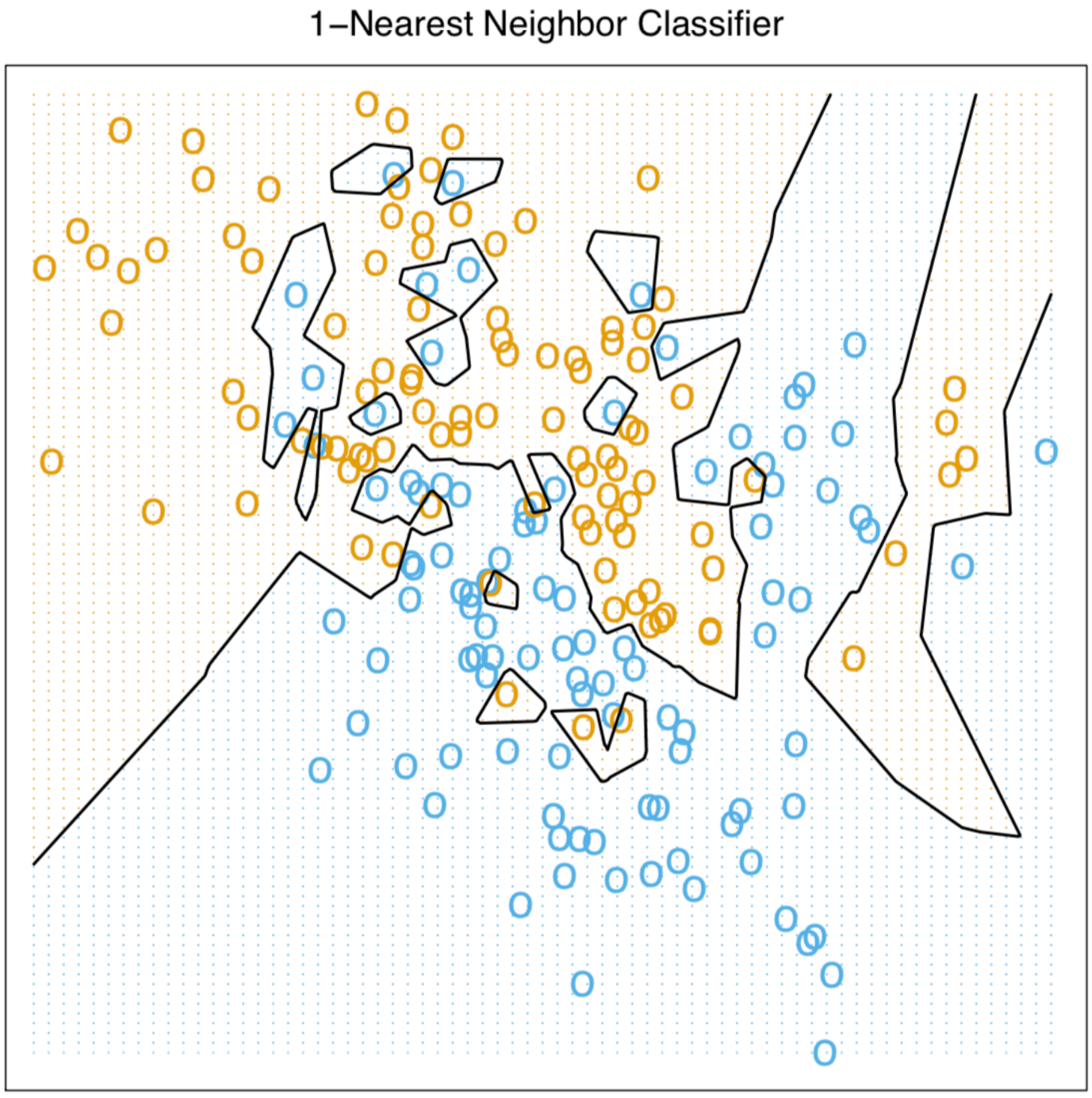


Testing on points not used in the training set « Cross-validation »



Finding « hyper-parameters » requires Cross-validation

Nearest-Neighbour classifiers



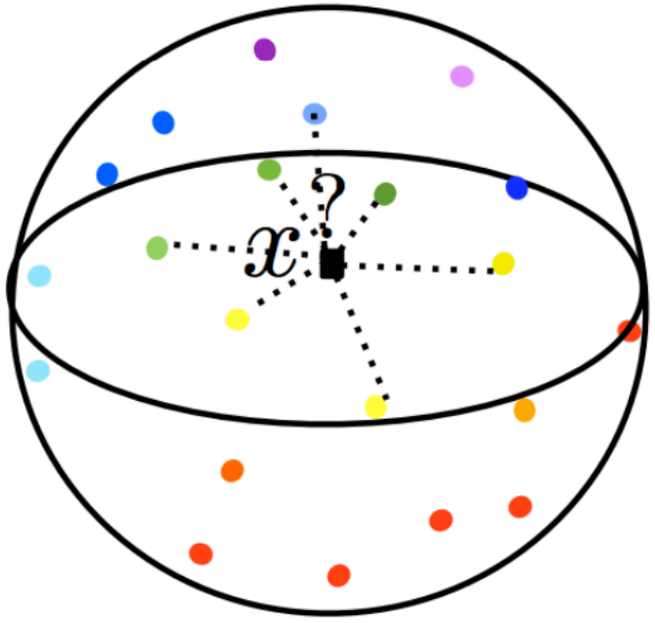
Pro:

– Simple to implement
– Flexible to feature/distance choices
– Naturally handles multi-class cases
– Can do well in practice
(with enough representative data)

Con:

– Large search problem
– Storage of data
– Must have a meaningful distance
– Curse of dimensionality

The curse of dimensionality



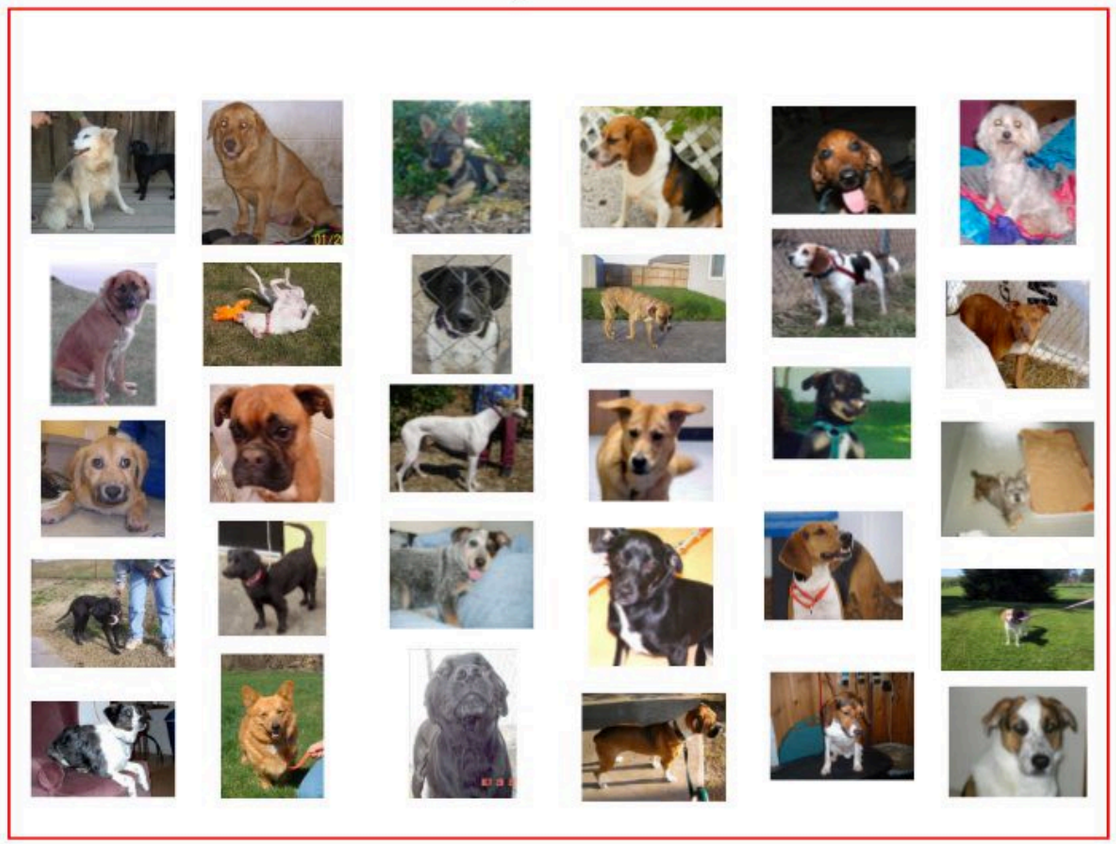
Need $O(\varepsilon^{-d})$ points to cover $[0, 1]^d$
at a Euclidean distance ε

Images: $d = 10^6$

Distance are always large

Cats

Dogs



Funes the memorious

Knn just memorise everything…
no « understanding/learning » whatsoever!

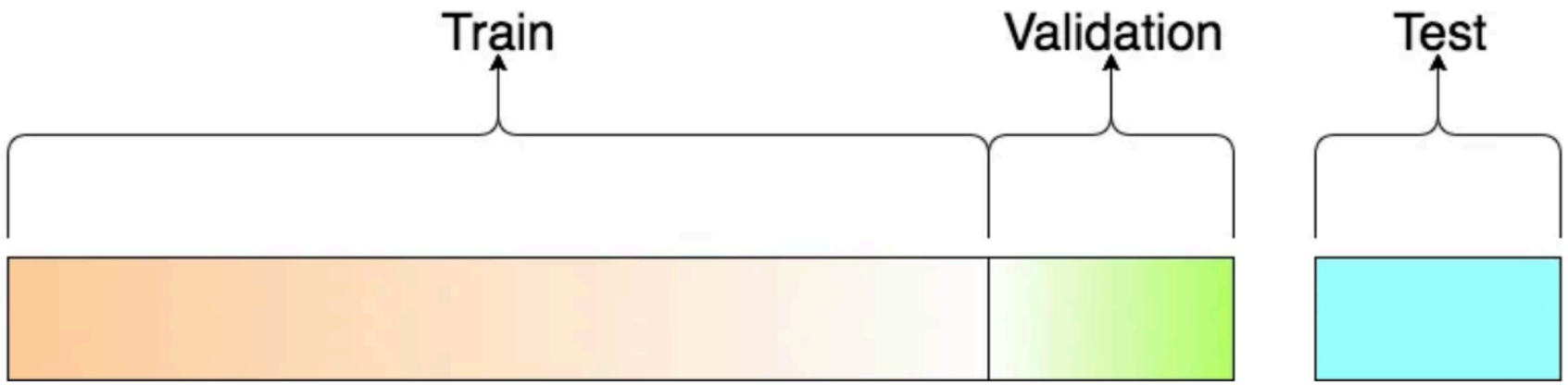
‘Not only was it difficult for him to understand that the generic term ‘dog’ could embrace so many disparate individuals of diverse size and shapes, it bothered him that the dog seen in profile at 3:14 would be called the same dog at 3:15 seen from the front.’

Without effort, he had learned English, French, Portuguese, Latin. I suspect, nevertheless, that he was not very capable of thought. To think is to forget a difference, to generalize, to abstract. In the overly replete world of Funes there were nothing but details, almost contiguous details.

Jorge Louis Borges, « *Funes the Memorious* » *1942*

Supervised learning good practices

Estimation
of generalisation error



A visualization of the splits

Training

Model selection
Tuning hyperparameters

Splitting Data into 3 Parts

To split data into three parts, we can set the `test_size` parameter to 0.33, which corresponds to a 33%-33%-34% split ratio. Here is an example code snippet:

⚠ This code is experimental content and was generated by AI. Please refer to this code as experimental only since we cannot currently guarantee its validity

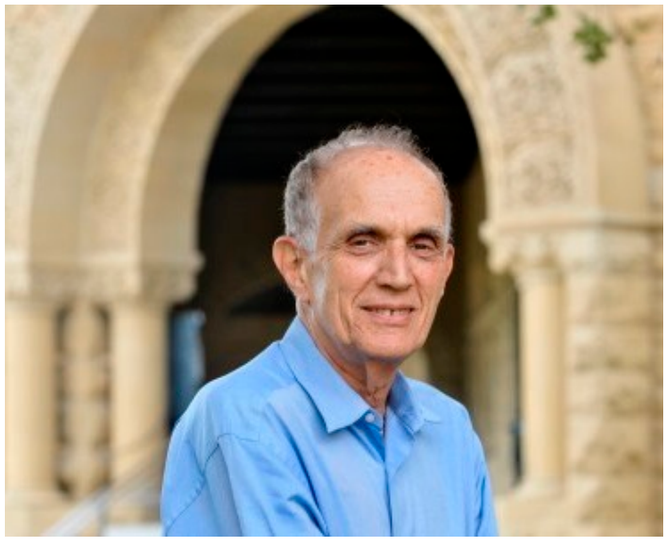
```python
from sklearn.model_selection import train_test_split

X = ... # input features
y = ... # target variable

X_train, X_val, X_test, y_train, y_val, y_test = train_test_split(X, y, test_size=0.33, ra

print(f"Number of training examples: {len(X_train)}")
print(f"Number of validation examples: {len(X_val)}")
print(f"Number of test examples: {len(X_test)}")
```

Bagging
(Bootstrap Aggregating)



Bradley Efron (1979)

Leo Breiman (1994)

Bootstrap

Why working with a single data set?

```
[[ 0.524]
 [ 0.4  ]
 [ 0.361]
 [ 0.078]
 [ 0.859]
 [ 0.654]
 [ 0.384]
 [ 0.738]
 [ 0.156]
 [ 0.572]]
```

EE-411 Fundamentals of inference and learning, EPFL

Exercice Session 1: looking at data with python and pandas

In this first set of exercices, we will start the course by solving some simple python problems that will help you warm up for the future. We advise you to run python notebooks on your browser, for instance using google colab (Watch Introduction to Colab to find out more) or our own serveur in epfl (on noto.epfl.ch). If you need to refresh your python skills, you may start by our introductory notebooks here and there.

What you will learn today: In this first session, we will discuss briefly how to use python, how to use pandas (a powerful, and easy to use, open source data analysis and manipulation tool), and discuss the idea of permutation test and bootstrap, that are amazingly useful concepts from statistics.

The Brexit data: who wanted out?

We will introduce the concept of permutation test in the hypothesis testing problem exploiting a cool analysis on the Brexit referendum following this great resource https://matthew-brett.github.io/les-pilot/brexit_ages.html. This will give us the opportunity to review the basic functionalities of pandas, a pivotal package in data-handling which we will find often during this course. In fact, you may want to follow an introduction to pandas (for instance this one, or the very useful Most Important Pandas Functions for Data Science).

Here is our problem: The Hansard society made a poll in the 2016 interviewing 1771 people on the Brexit referendum (data available here https://beta.ukdataservice.ac.uk/datacatalogue/studies/study?id=8183).

If we dig into the data, we see that the average age of the "Brexiteers" is higher from the one of the "Remainers". We are interested in addressing the following question: how **confident** can we be that this rule is general and is not an artifact due to low sample size?

This problem falls in the broader context of hypothesis testing problem:

- Hypothesis (H1) - effective difference in the age of Brexiteers and Remainers

vs

- "Null hypothesis" (H0) - no difference in the age of the two groups

This seems like a hard problem but we will see that with a very simple idea we will be able to characterize th
steps and let's understand how to handle a dataset with pandas.

set?

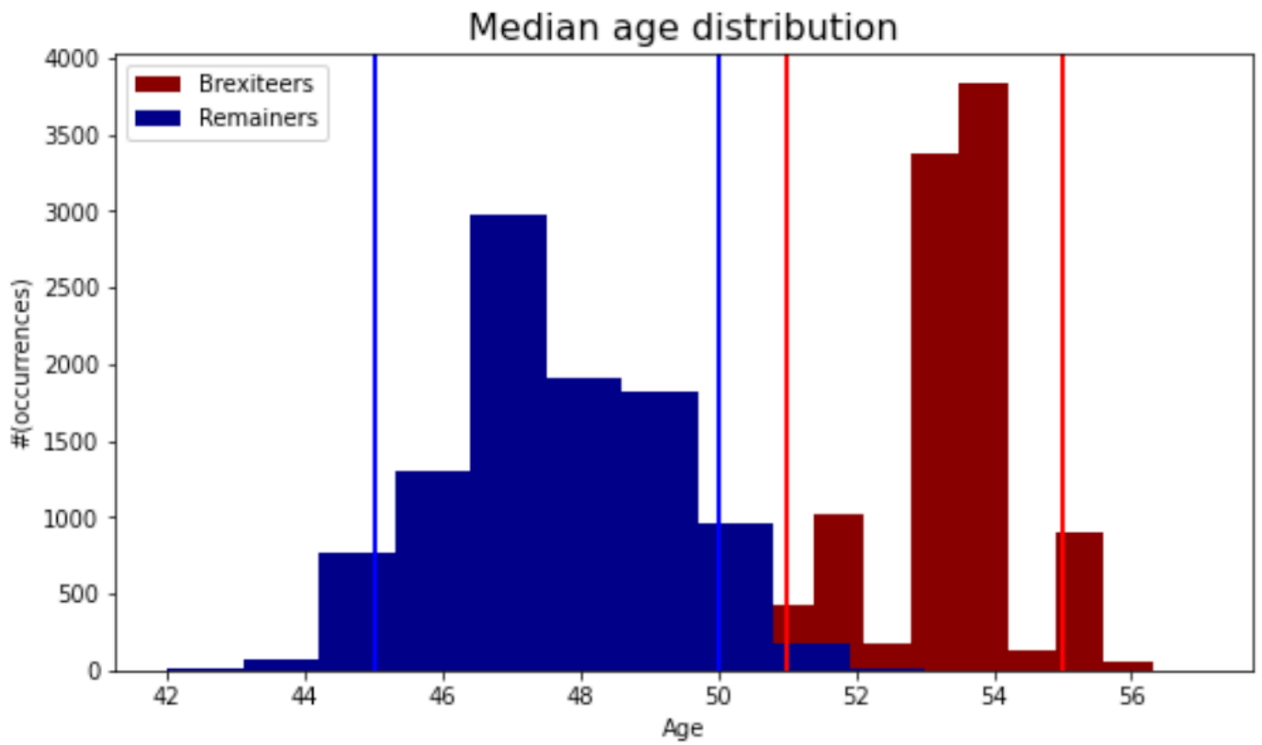
replacement:

```
56]
56]
24]
78]
24]
59]
61]
78]
54]
 ]]
```

```
[[ 0.078]
 [ 0.524]
 [ 0.859]
 [ 0.384]
 [ 0.859]
 [ 0.361]
 [ 0.361]
 [ 0.738]
 [ 0.572]
 [ 0.156]]
```

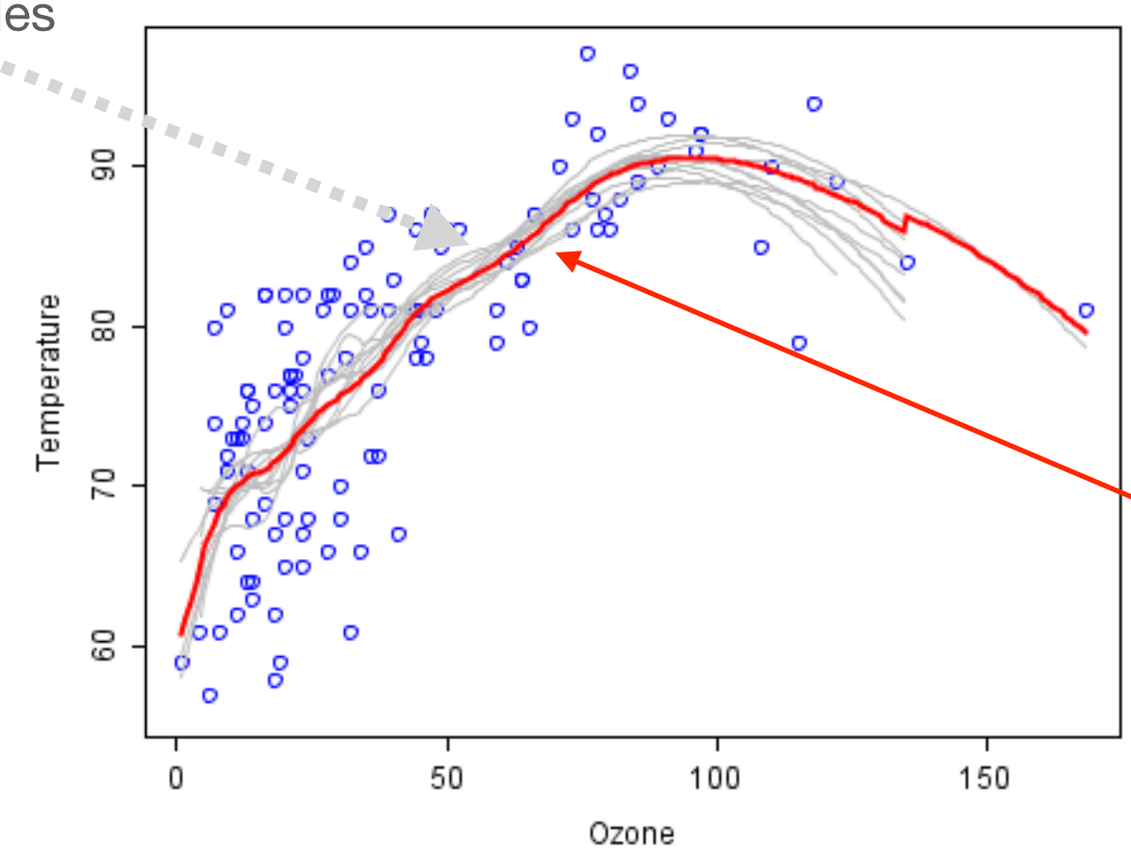
Consider these

Many uses: error anal



Bagging
(Bootstrap Aggregating)

Fits on bootstrap samples

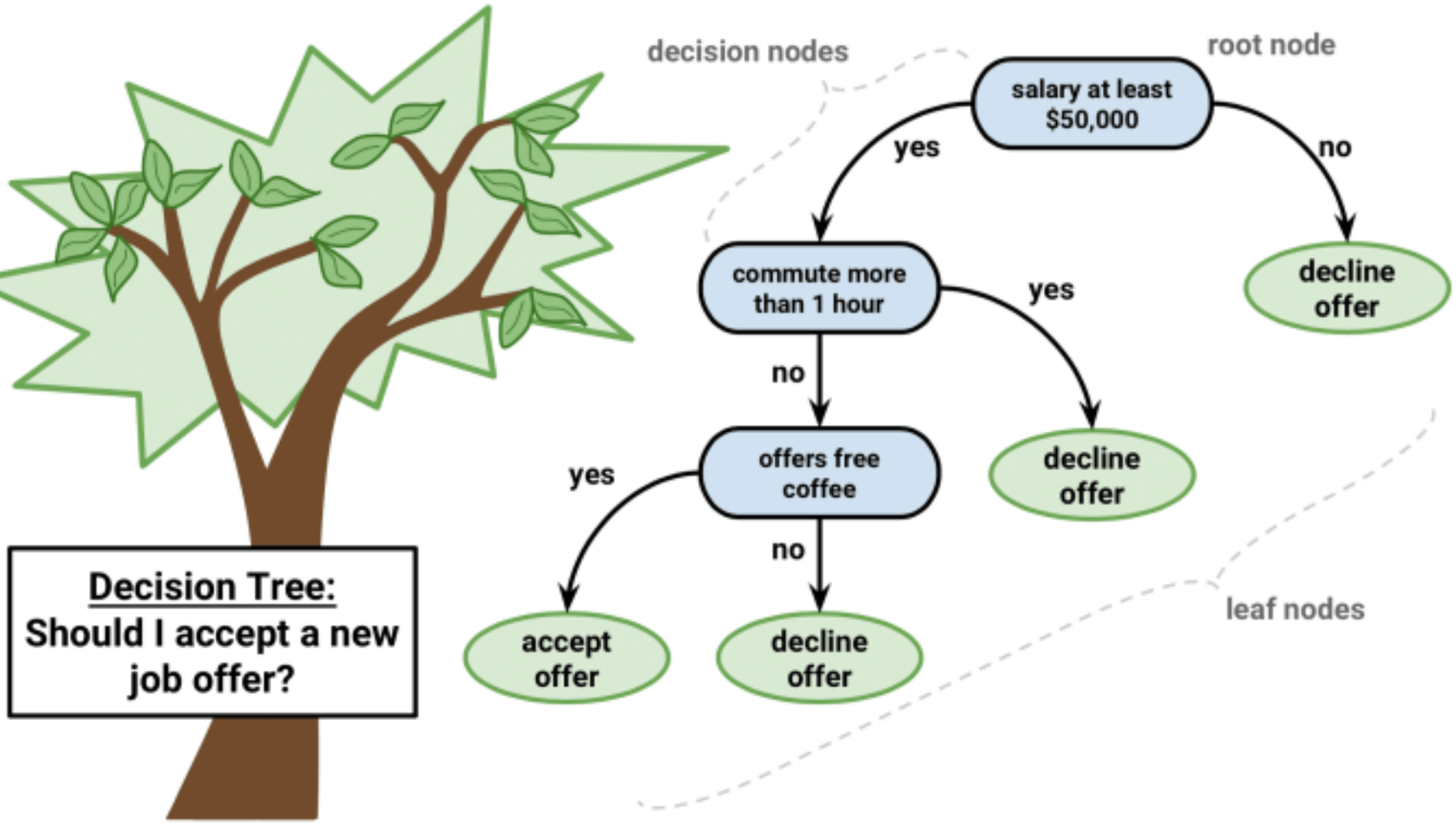


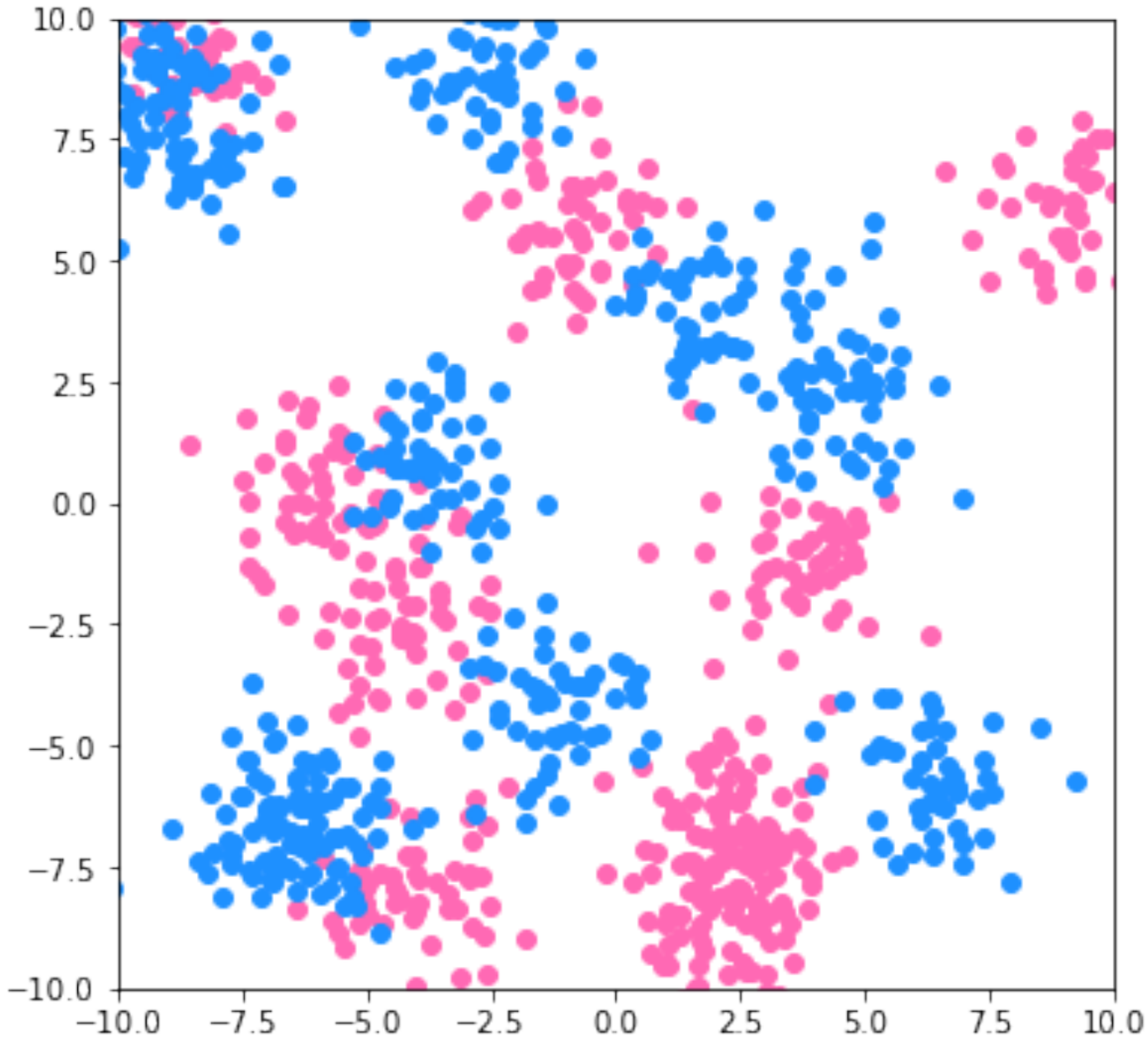
Ozone data

Averaging models reduces overfitting!

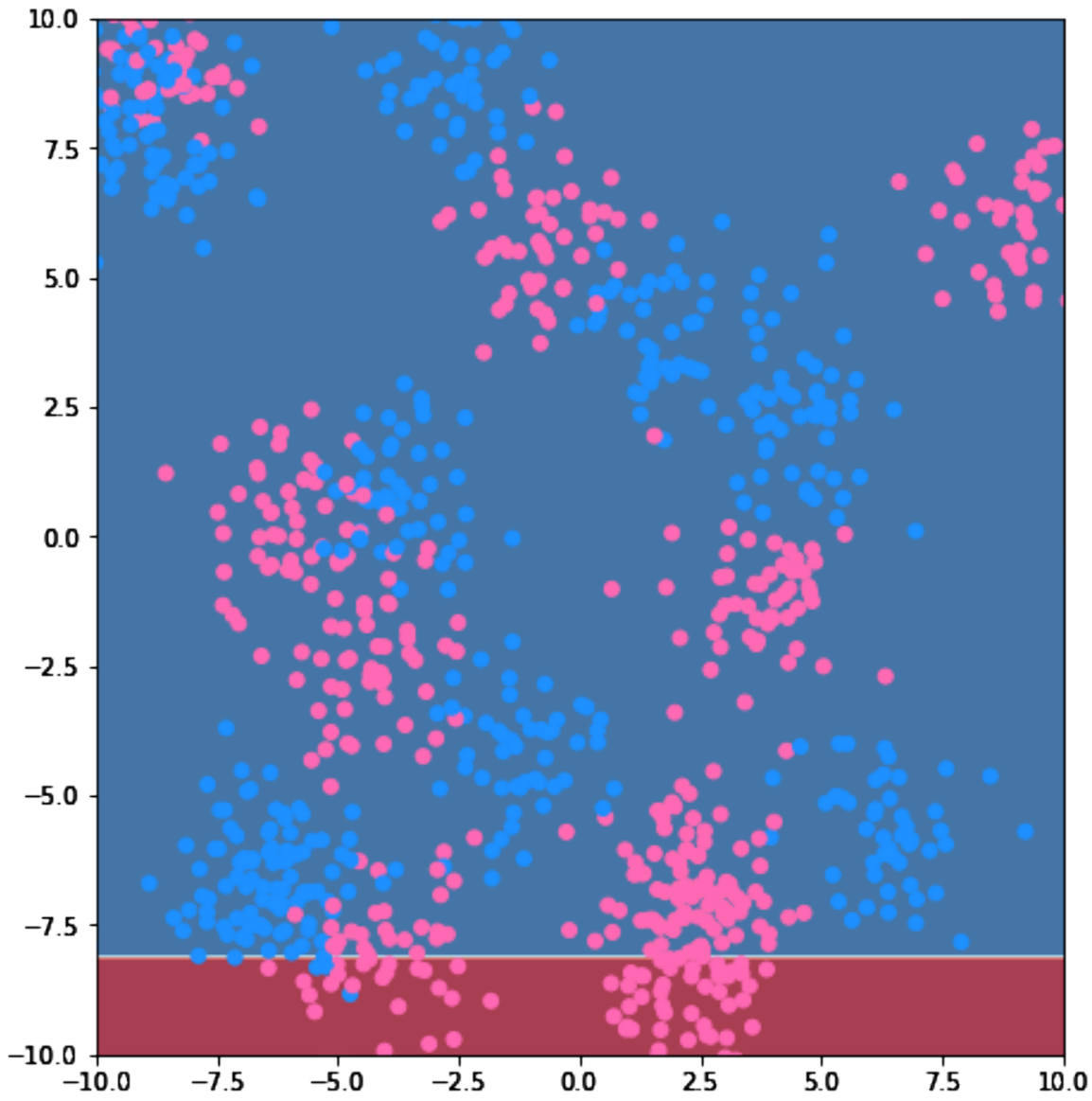
And for classification?

Decision trees!




10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

Decision tree, depth=1



How regions are decided

Decision boundaries are decided by finding

a) one of the variable (i=1…d)

b) a particular value of this variable to cut between $x_i \geq v$ and $x_i \leq v$

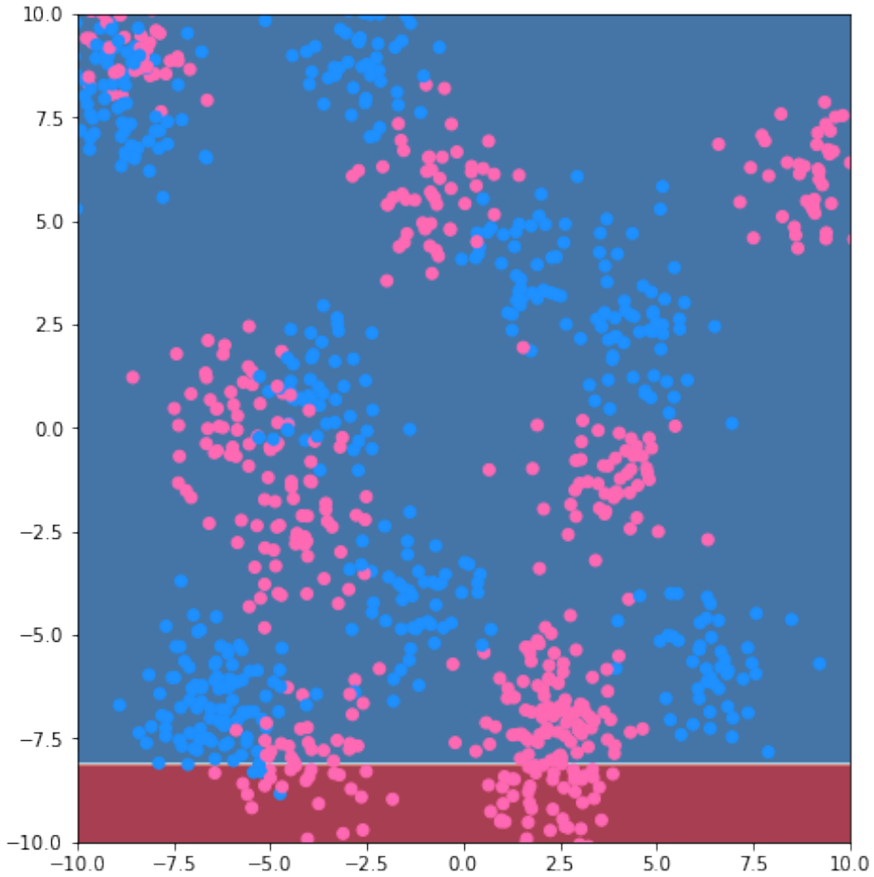
<u>(a) & (b) are chosen to minimise the GINI criterion of all regions</u>

GINI Coefficient: Call

* $p_1(R)$ the fraction of points with label 1 in region R
* $p_2(R)$ the fraction of points with label 2 in region R

The Gini impurity is given by $Imp(R) = \sum_{i=1}^{2} p_i(R)(1 - p_i(R))$

(Note: this is 0 if everyone is perfectly classified)

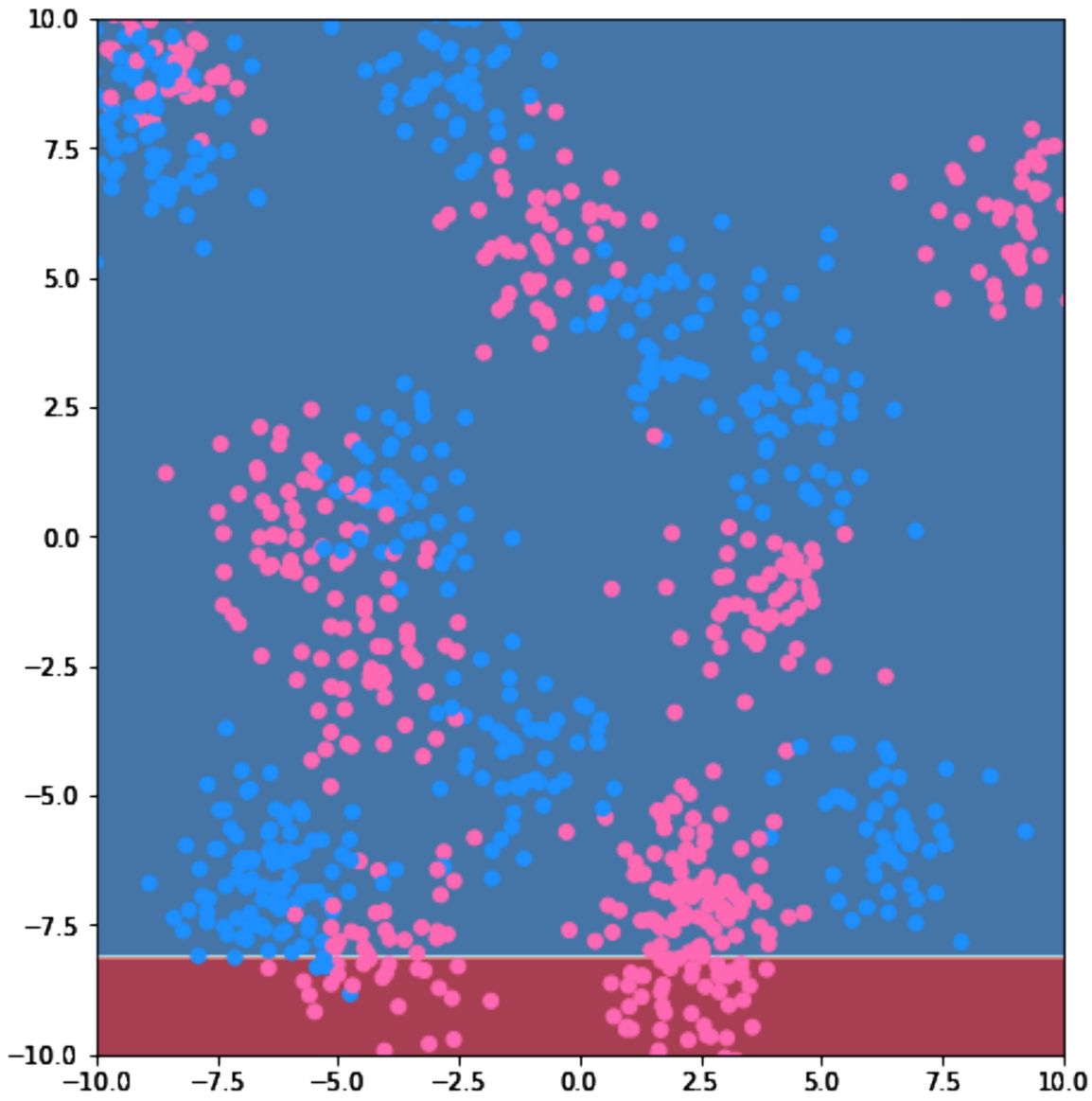


Alternative possibilities

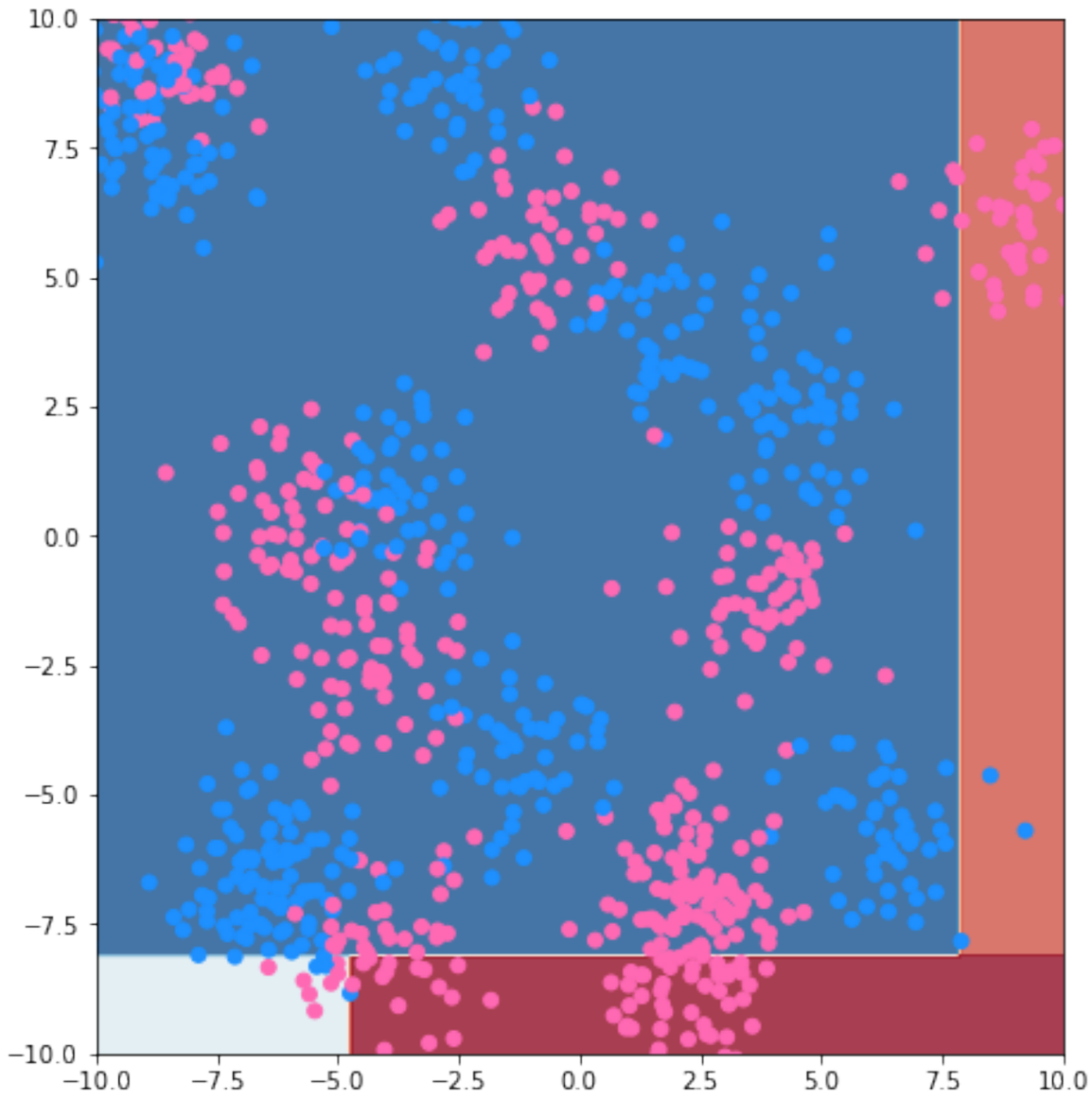
Shannon entropy : $Imp(R) = \sum_{i=1}^{2} - p_i(R)\log(p_i(R))$

Misclassification error: $Imp(R) = 1 - \max_i p_i(R)$

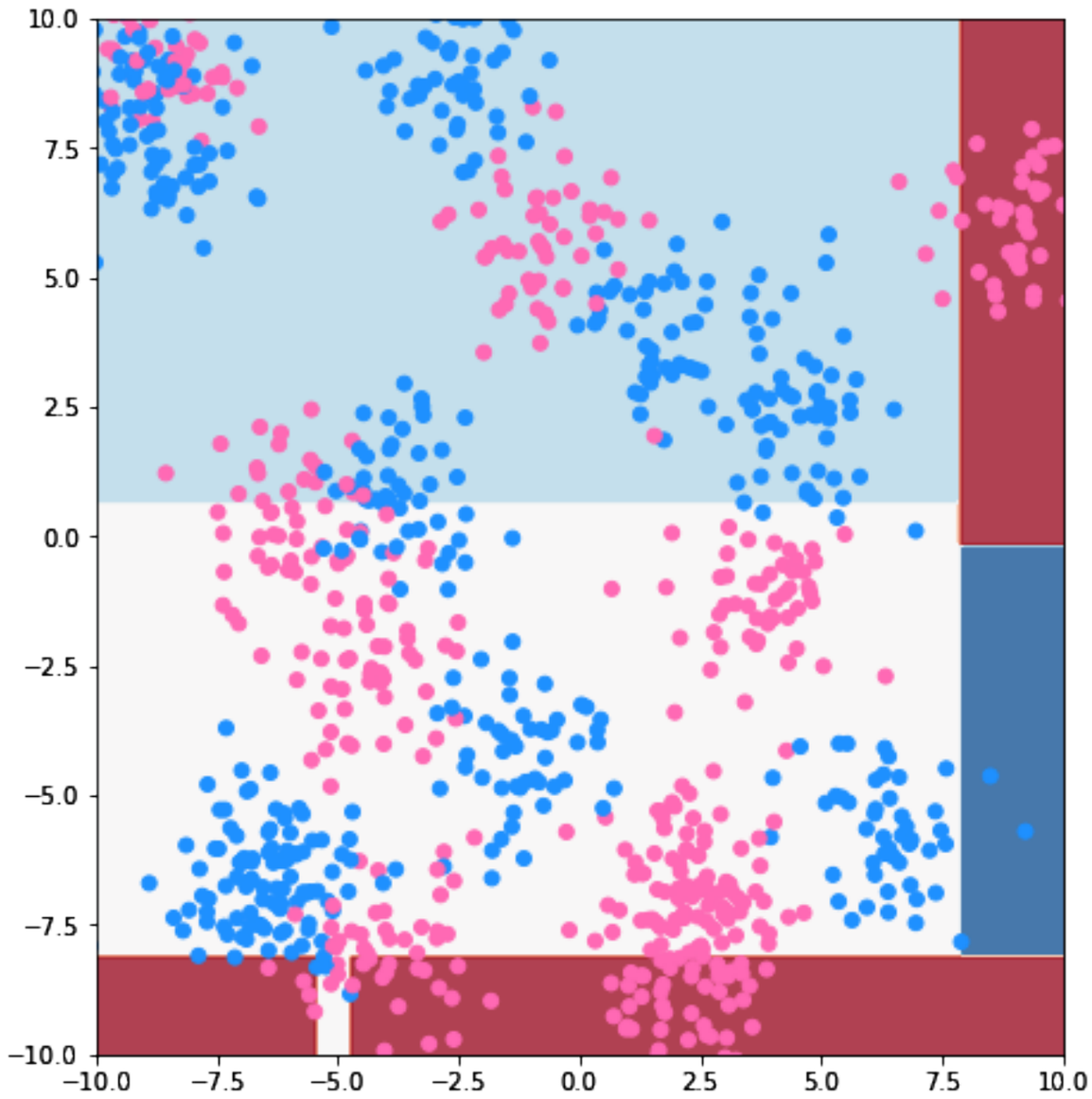
Decision tree, depth=1



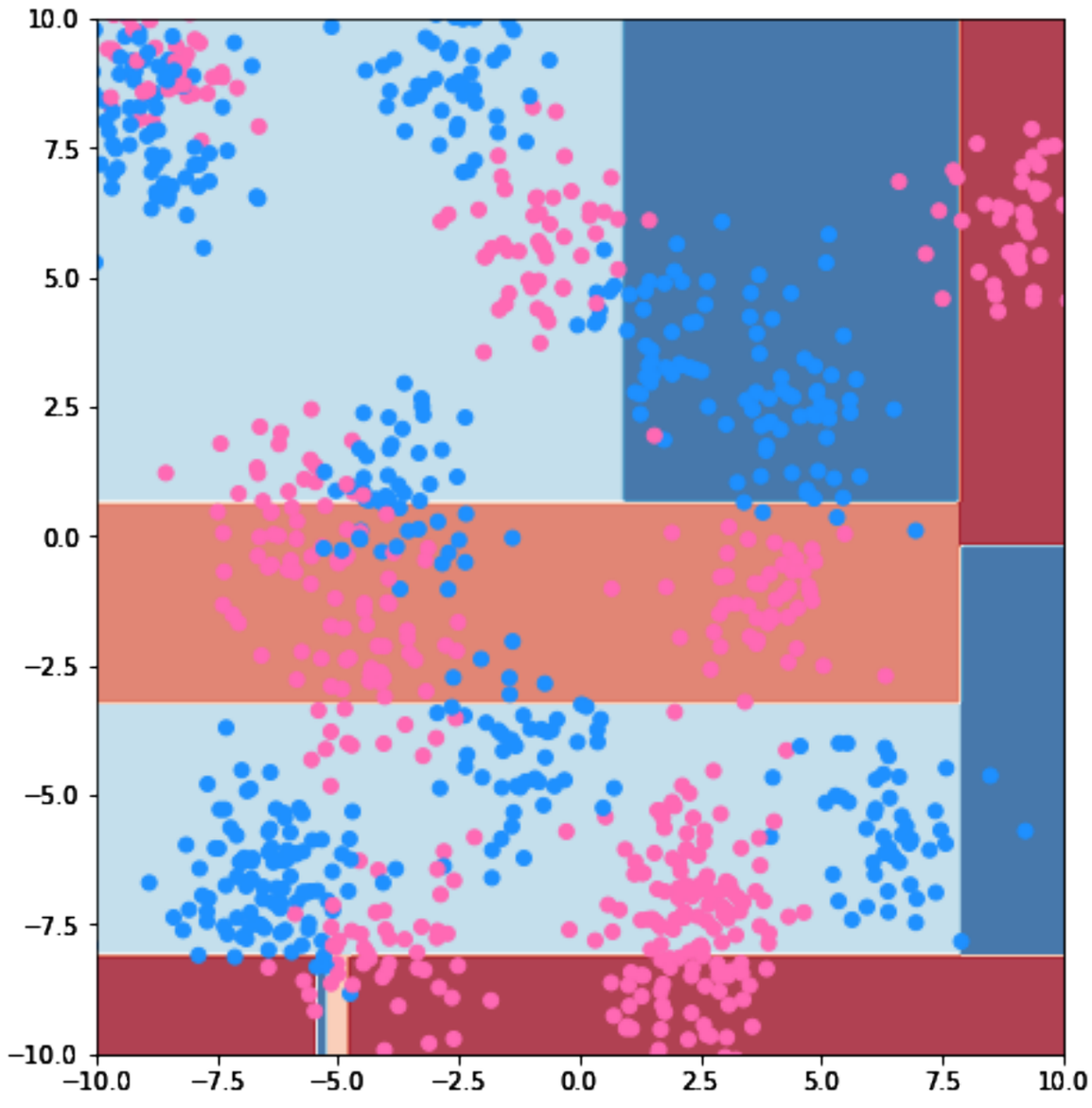
Decision tree, depth=2



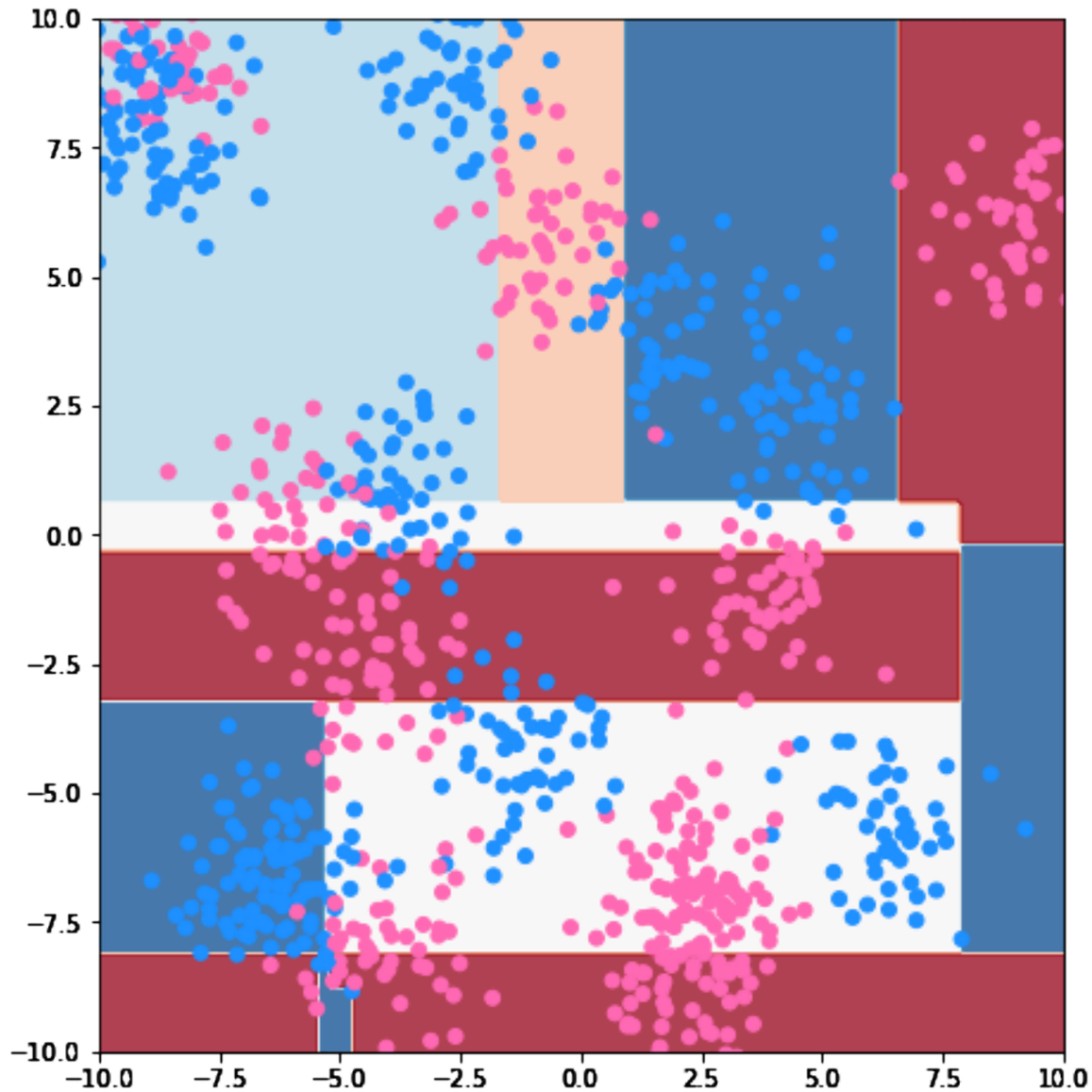
Decision tree, depth=3



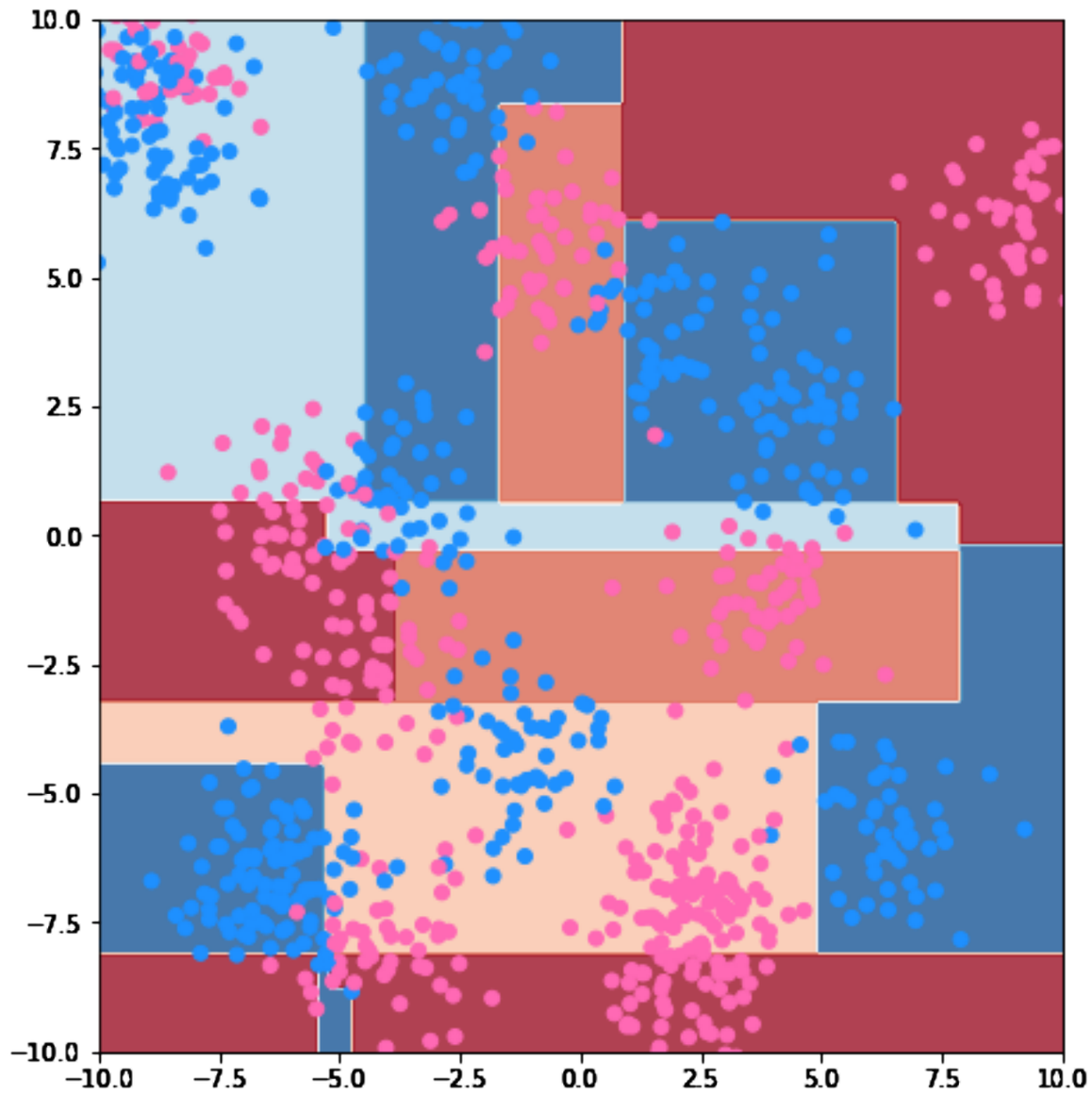
Decision tree, depth=4



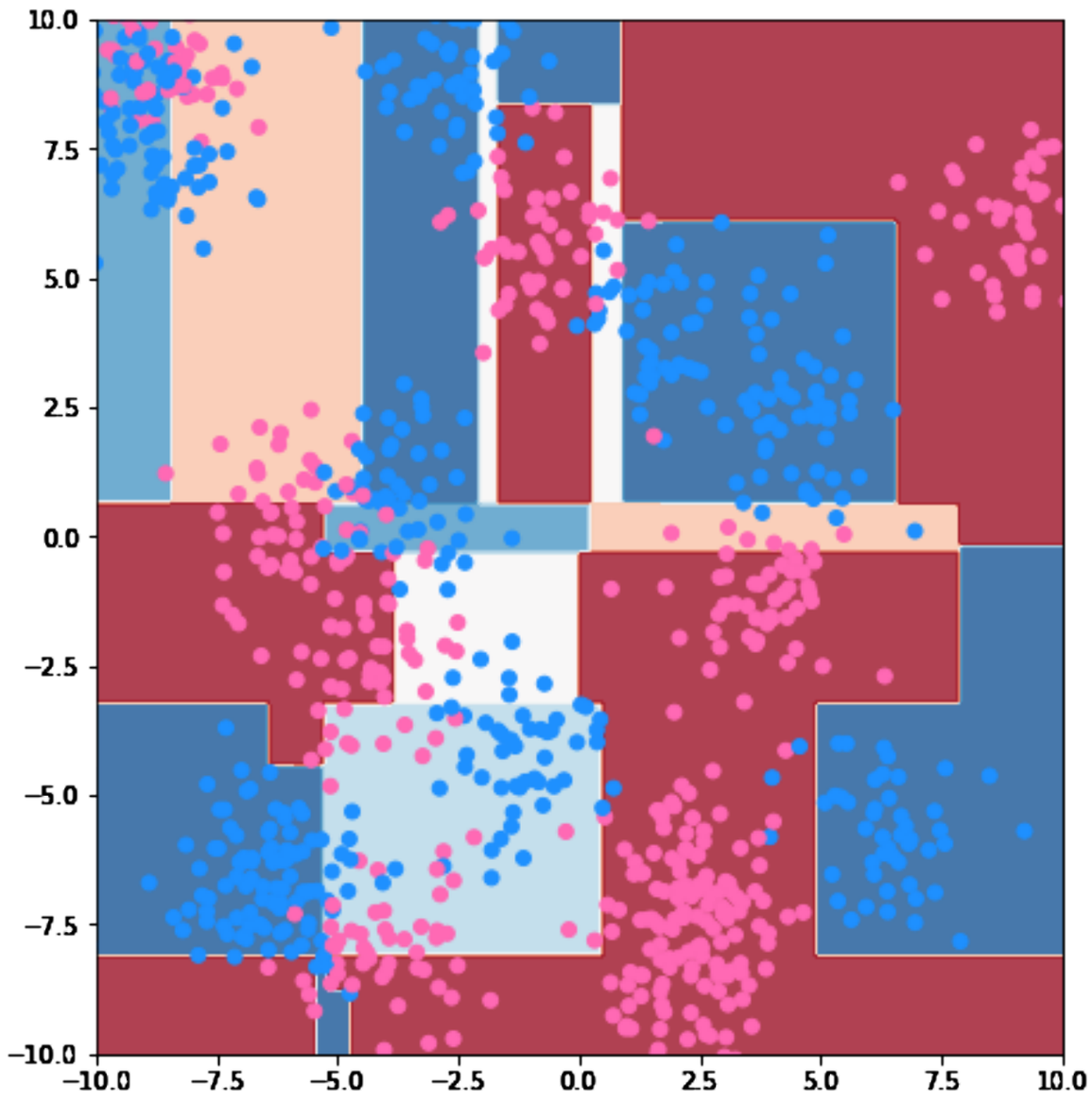
Decision tree, depth=5



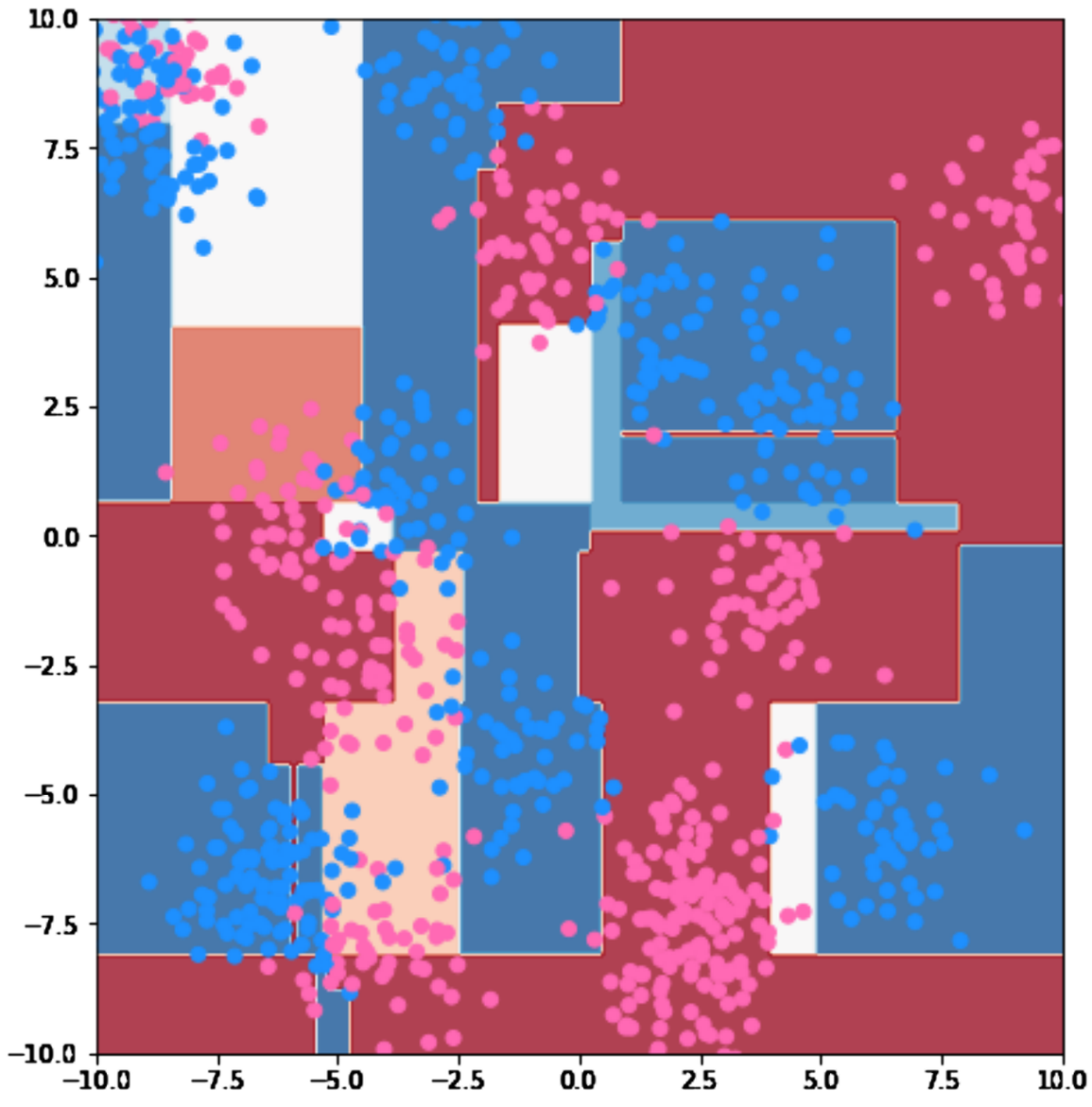
Decision tree, depth=6



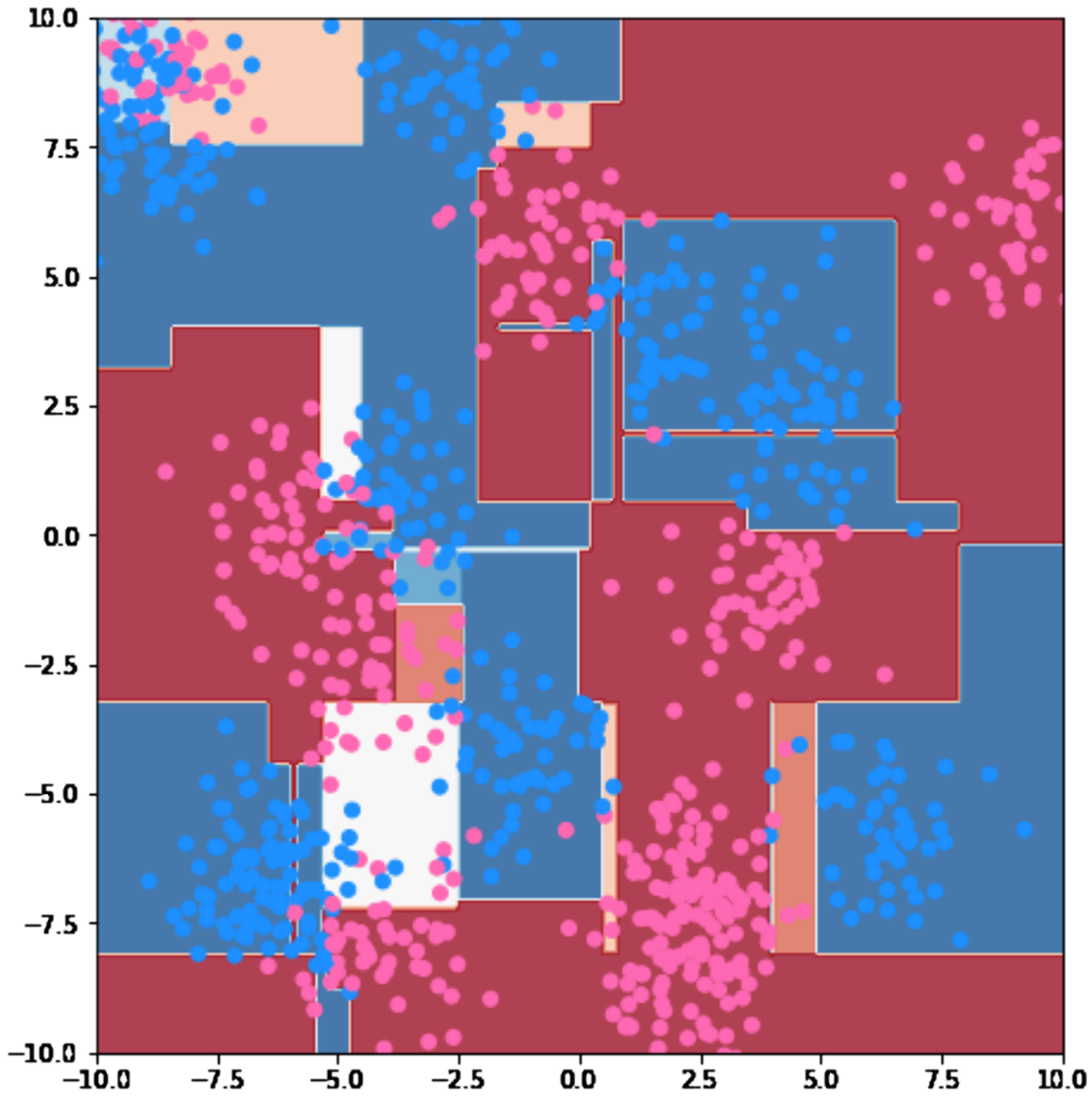
Decision tree, depth=7



Decision tree, depth=8



Decision tree, depth=9



Bagging

"Bootstrap Aggregation"

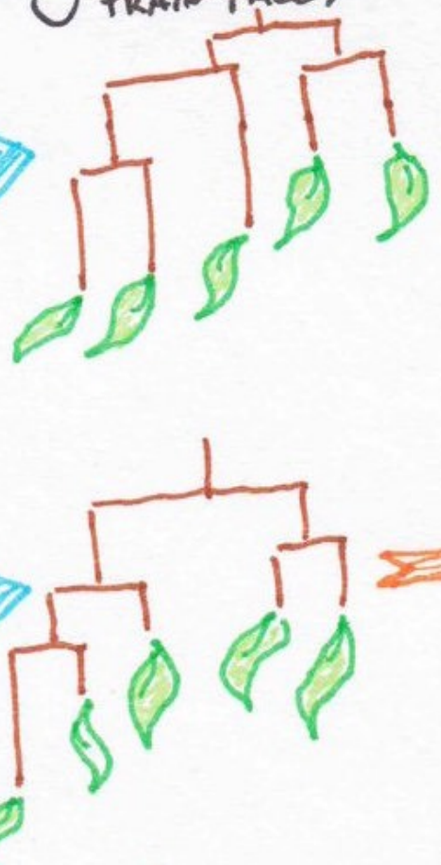
DATASET

1
2
3
4

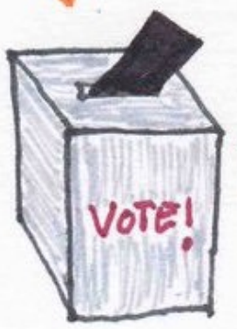
RESAMPLE W/ REPLACEMENT

3	2	4	4	2
1	3	1	1	1
2	1	1	3	3
3	1	3	4	4

TRAIN TREES



ALL TREES VOTE

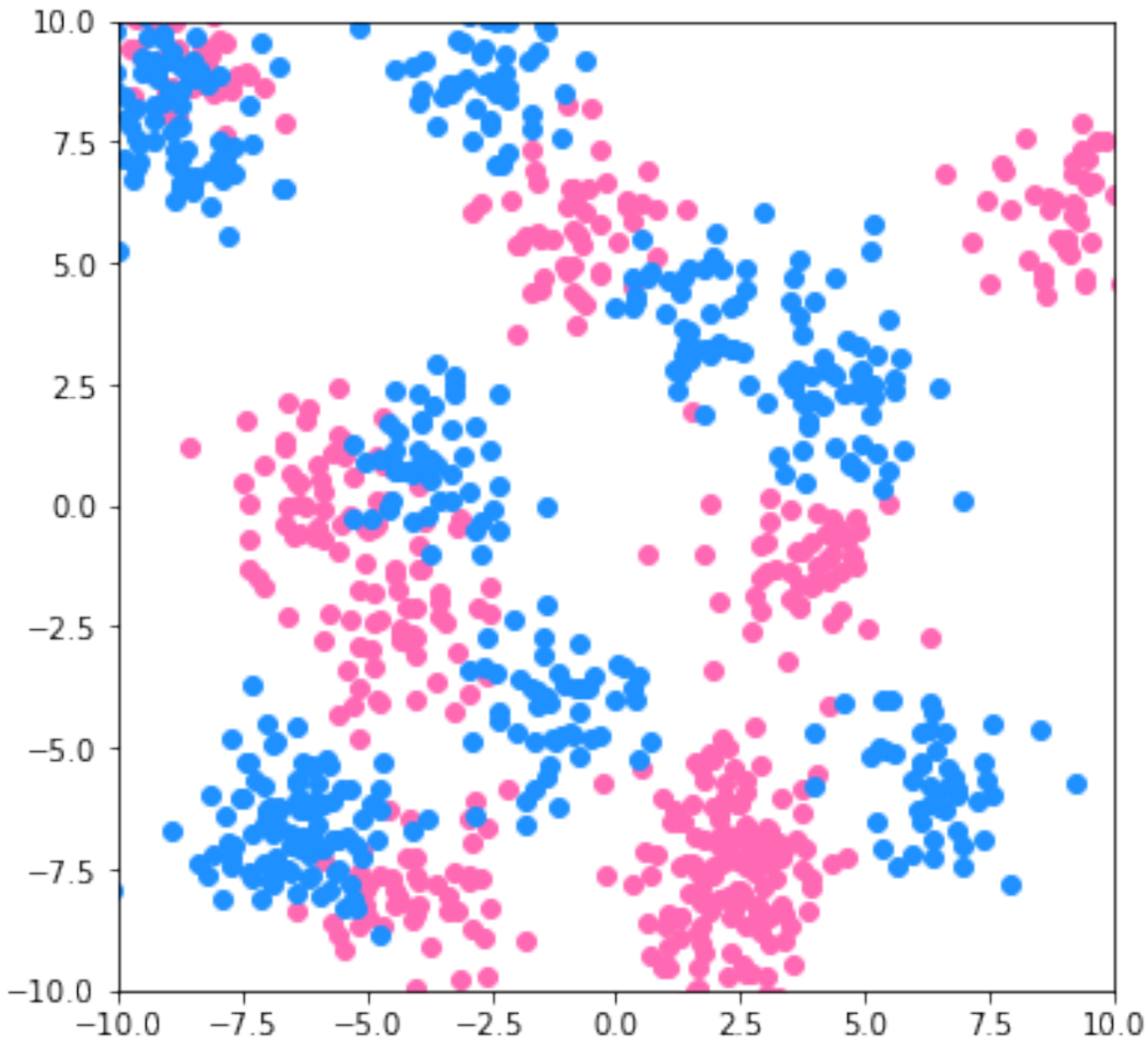


OUTCOME IS PREDICTION

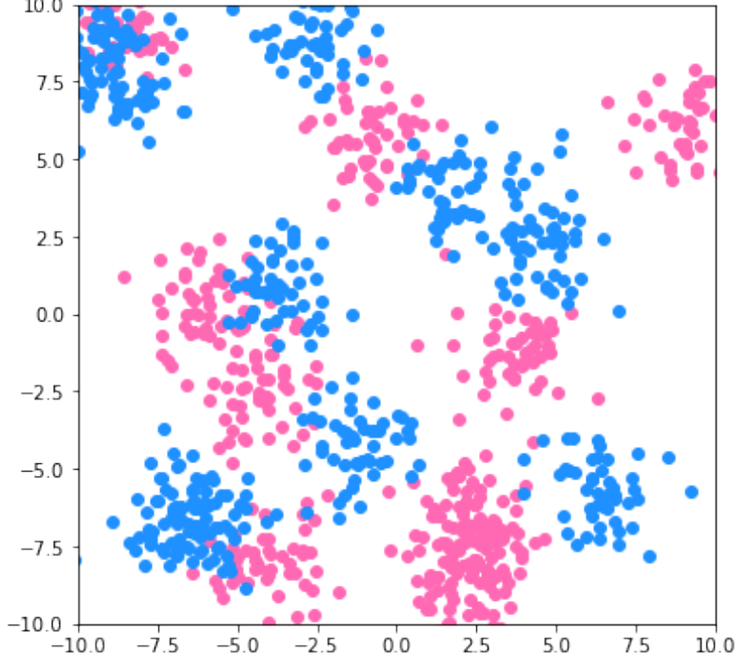
Prediction

Note: Random Forests is bagging but random sets of features in addition to bootstrapped samples

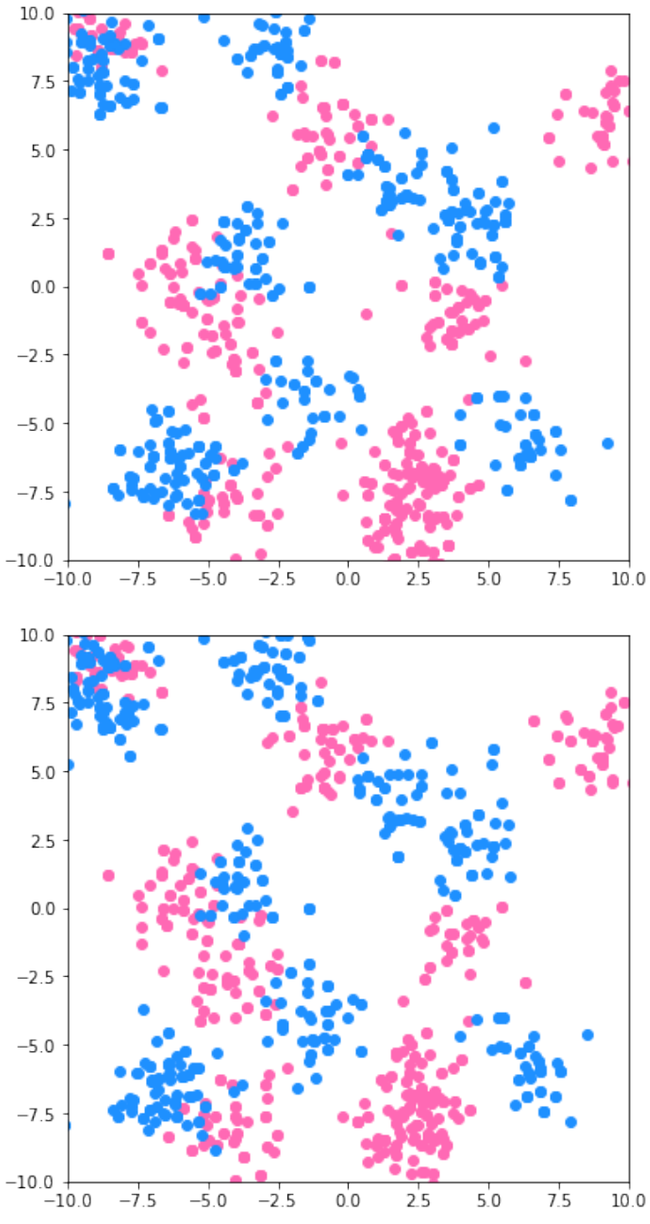
The original set

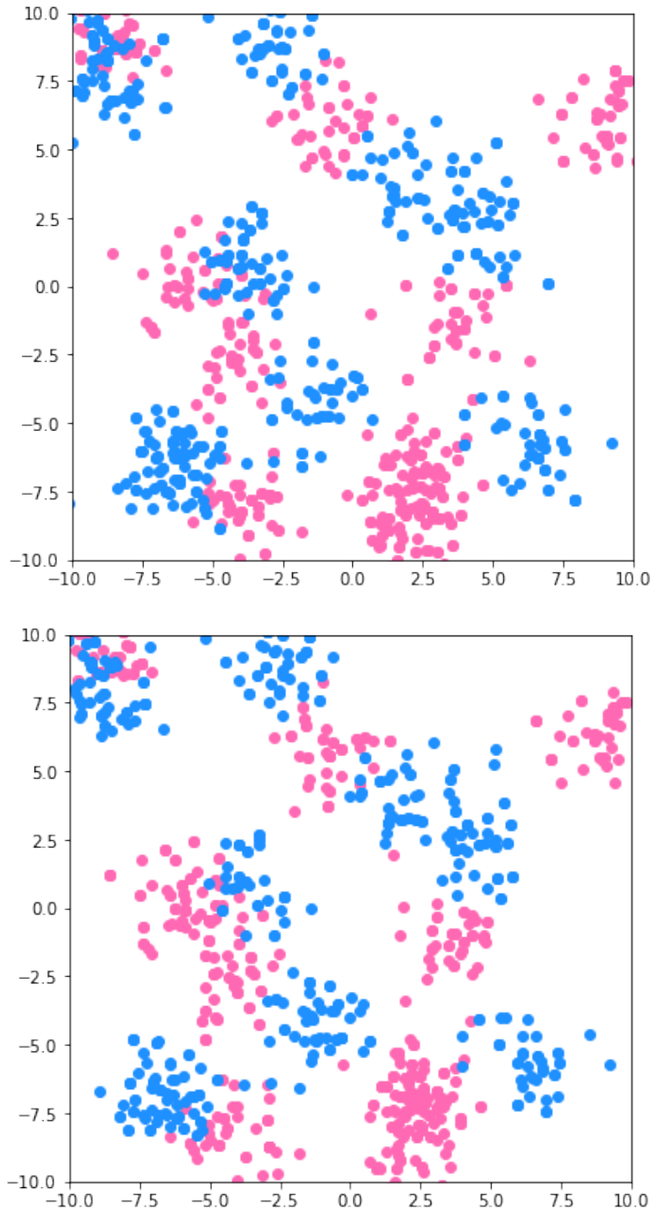


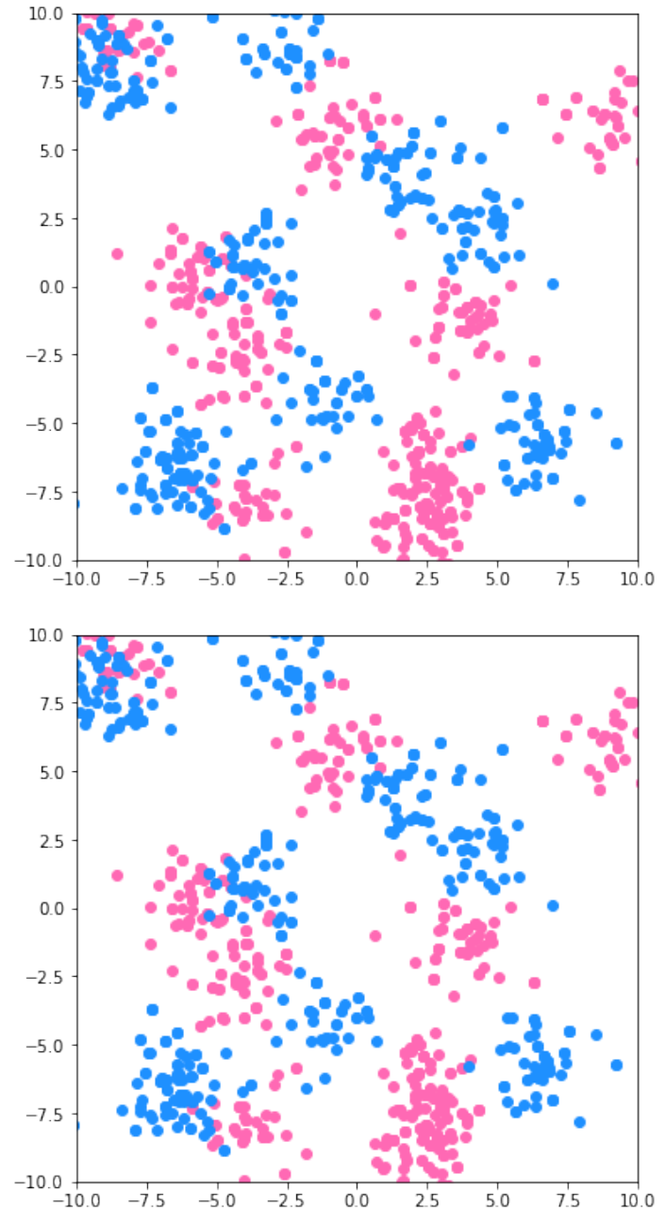
BOOSTRAP!!



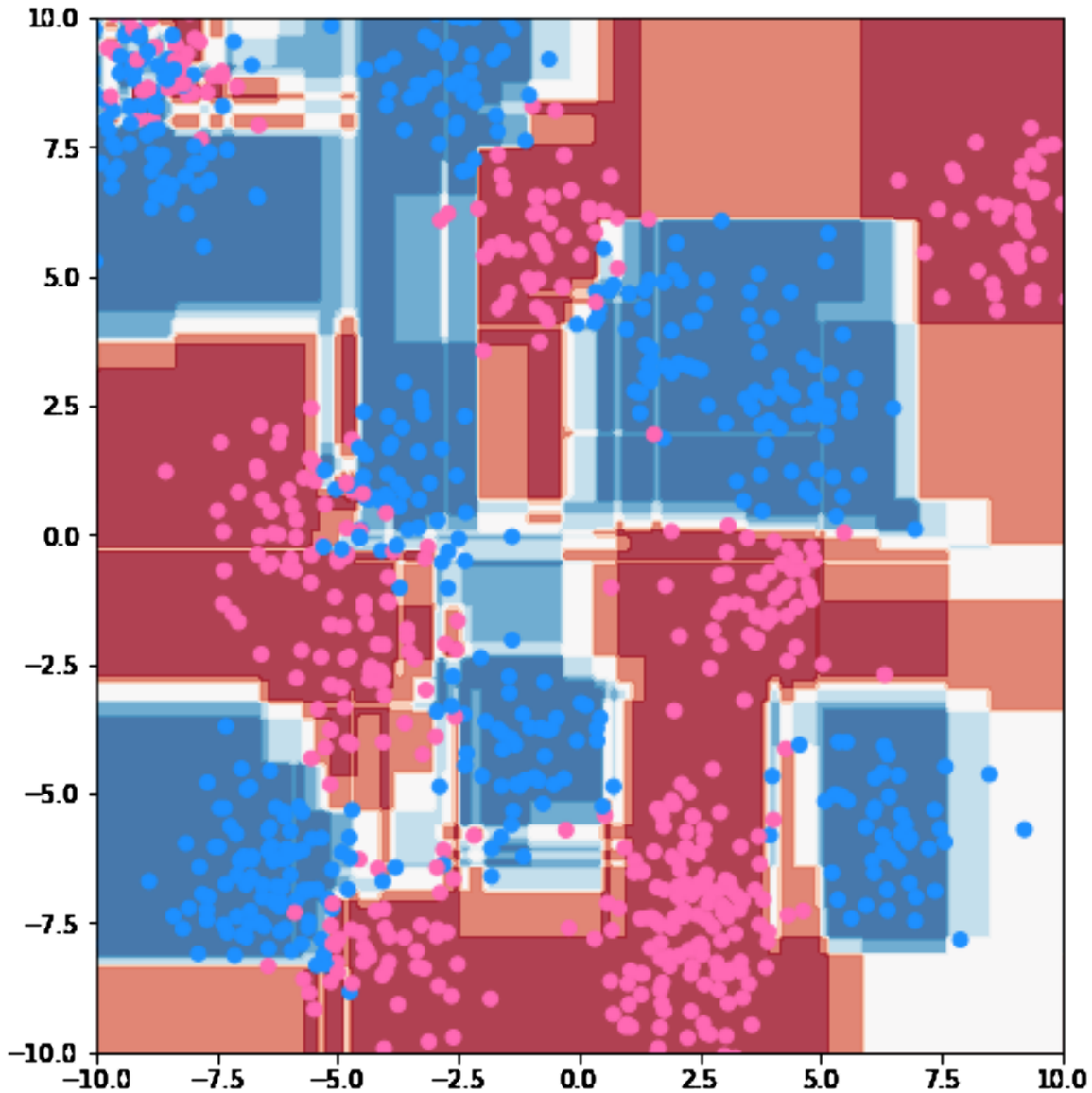
BOOSTRAP!!



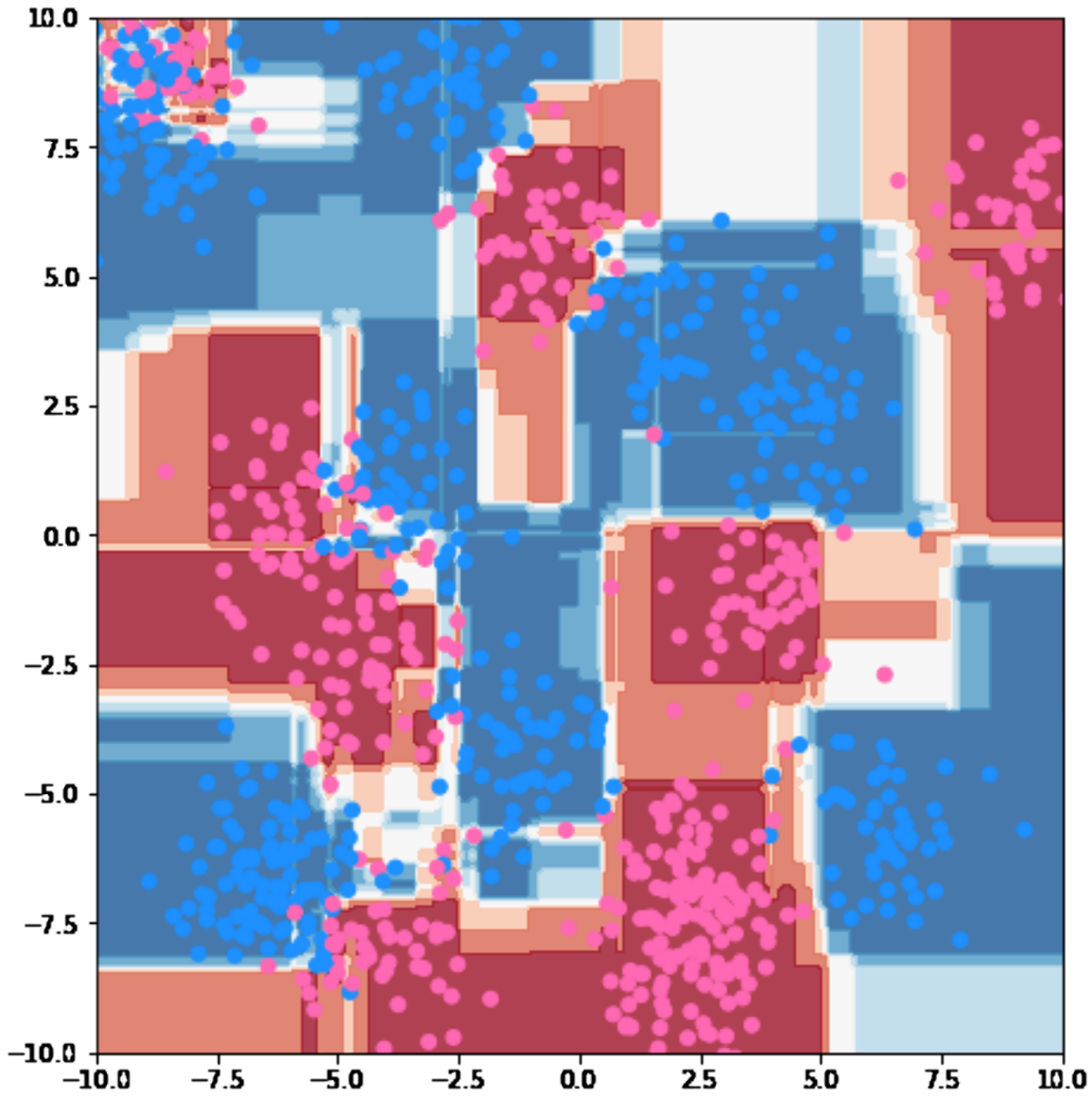




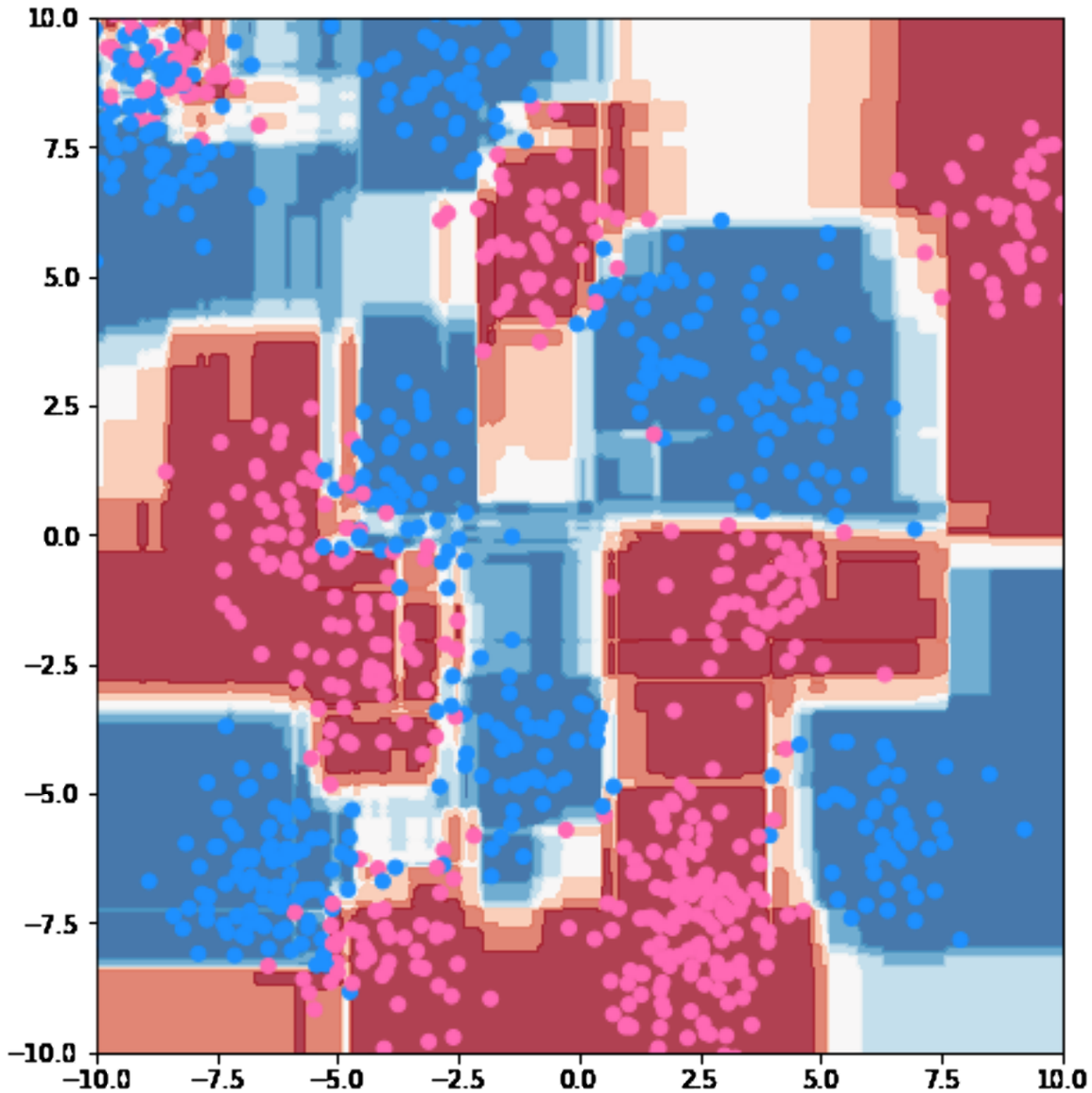
Bagging 5 Decision tree!



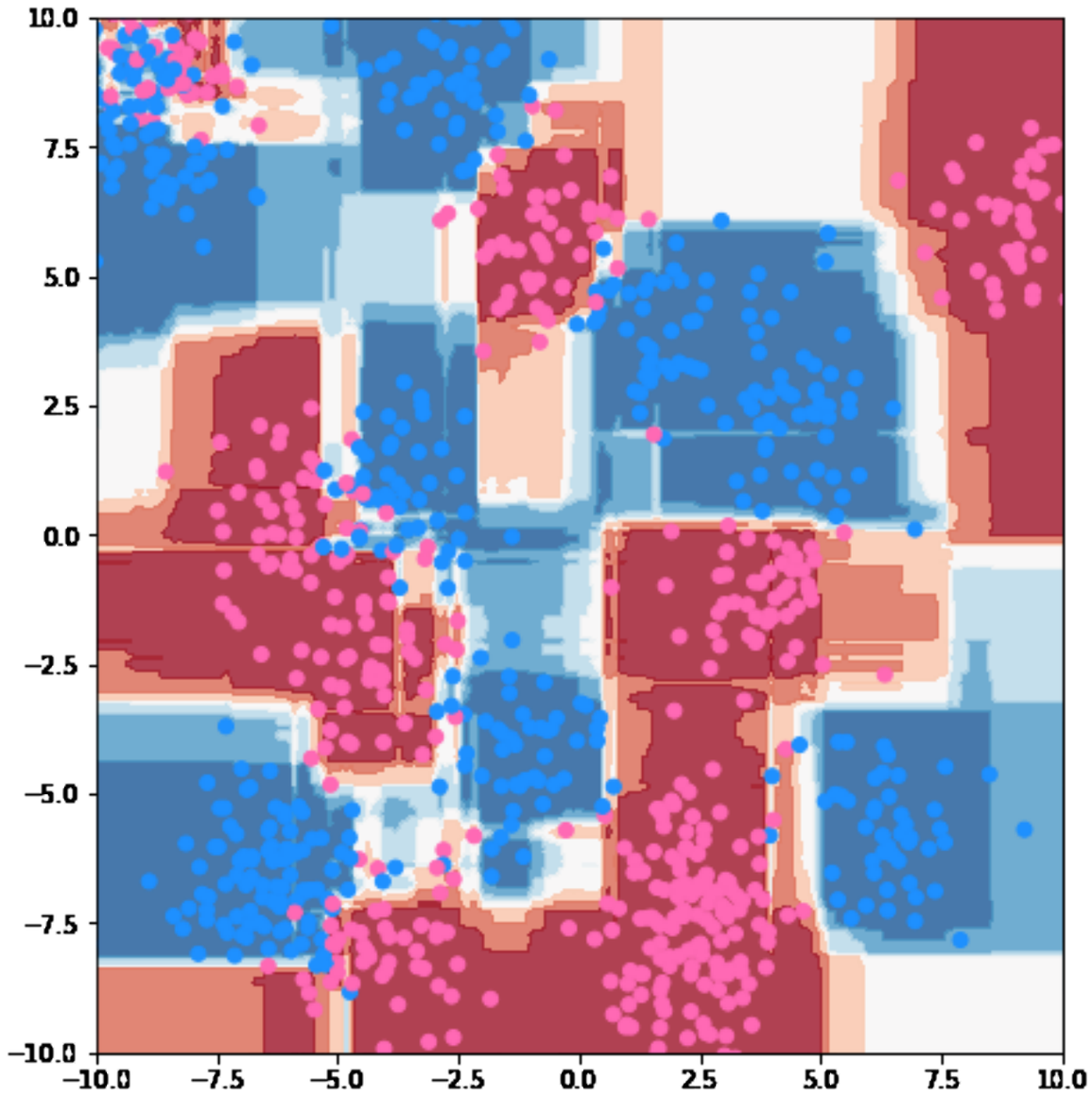
Bagging 10 Decision tree!



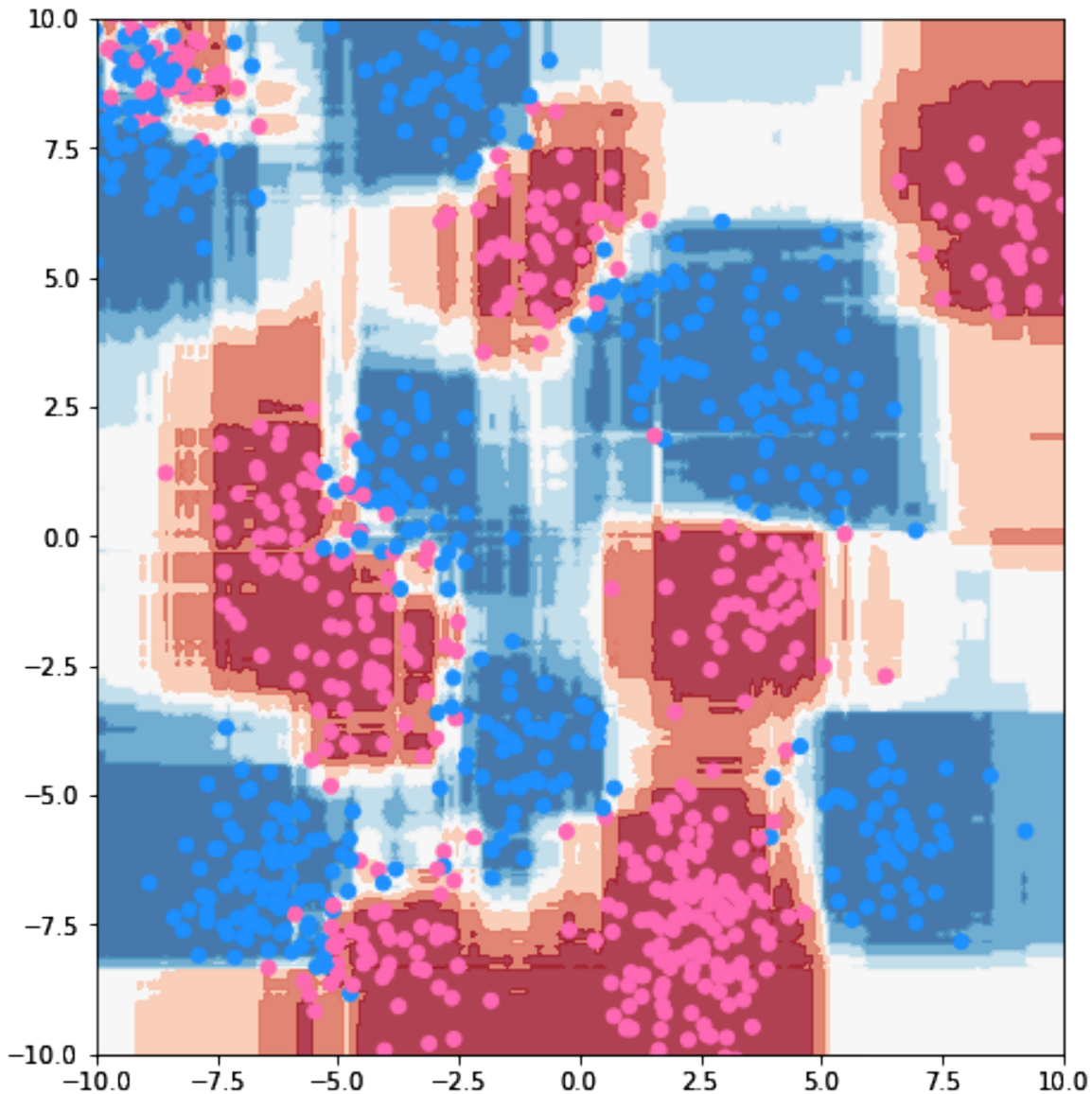
Bagging 20 Decision tree!



Bagging 20 Decision tree!



In practice: Random Forrest



Further randomization: at each candidate split in the learning process, a random subset of the features. For a classification problem with p features, $\sqrt{p}$ **(rounded down)** features are used in each split

Sk-learn!

sklearn.ensemble.RandomForestClassifier

class sklearn.ensemble.RandomForestClassifier(*n_estimators=100*, ***, *criterion='gini'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features='auto'*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *bootstrap=True*, *oob_score=False*, *n_jobs=None*, *random_state=None*, *verbose=0*, *warm_start=False*, *class_weight=None*, *ccp_alpha=0.0*, *max_samples=None*)

[source]

A random forest classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the `max_samples` parameter if `bootstrap=True` (default), otherwise the whole dataset is used to build each tree.

Read more in the User Guide.

Boosting

Can we make many dumb learners smart?

Yes (but you need a smart leader!)

Can we make many dumb learners smart?

Kearns and Valiant ’88:
- Does weak learnability imply strong learnability?
In other words, can we boost from weak to strong?

Schapire ’89
Freund and Schapire ’95
- Yes, with adaBoost

How to combine many weak classifiers?

Ingredients:

(i) T classifiers $h_t(.)$, each of them slightly better than random
(ii) A training set with N labeled examples

Adaptive linear combination of classifiers

$$H(\vec{x}) = \sum_{t=1}^{T} \alpha_t h_t(\vec{x})$$

ADABOOST

Exponential loss:

$$\mathcal{R} = \sum_i e^{-Y_i H(\vec{x}_i)}$$

$Y_i = \pm 1$ $\quad$ $h_t(x_i) = \pm 1$

Goal: start from t=0, add new classifiers and

(i) Adapt the weight alpha at each steps
(ii) Re-weight the instances in the training set : more weight to ill-classified instances
(each new classifier concentrate on badly classified examples)

How does one set the weight and α at each time steps?

Assume we have done the job until time τ-1 !

$$\mathcal{R} = \sum_i e^{-Y_i H(\vec{x}_i)} \qquad H(\vec{x}) = \sum_{t=1}^{T} \alpha_t h_t(\vec{x})$$

$$\mathcal{R} = \sum_i e^{-Y_i \overbrace{\sum_{t=1}^{\tau-1} \alpha_t h_t(\vec{x}_i)}^{\lambda_i^\tau} - Y_i \alpha_\tau h_\tau(\vec{x}_i)}$$

$$\mathcal{R} = \sum_i \underbrace{e^{-Y_i \lambda_i^\tau}}_{\omega_i^\tau} e^{-Y_i \alpha_\tau h_\tau(\vec{x}_i)}$$

$$\mathcal{R} = \sum_i \omega_i^\tau e^{-Y_i \alpha_\tau h_\tau(\vec{x}_i)}$$

ω_i^τ **= weights for each instances at time τ**

How does one set the weight and α at each time steps?

Assume we have done the job until time τ-1 !

$$\mathcal{R} = \sum_i \omega_i^\tau e^{-Y_i \alpha_\tau h_\tau(\vec{x}_i)}$$

$$\mathscr{R} = e^{-\alpha_t}\left(1 - \underbrace{\sum_i \omega_i \mathbf{1}(Y_i \neq h_i)}_{\epsilon_t}\right) + e^{\alpha_t}\left(\underbrace{\sum_i \omega_i \mathbf{1}(Y_i \neq h_i)}_{\epsilon_t}\right)$$

ϵ_t is what the classifier $h_t(.)$ is trying to minimise

$$\mathcal{R} = e^{-\alpha_\tau} - e^{-\alpha_\tau}\epsilon_t + e^{\alpha_\tau}\epsilon_t = e^{-\alpha_\tau}(1-\epsilon_t) + e^{\alpha_\tau}\epsilon_t$$

Let us choose α_t in order to minimize the global risk

$$\partial_{\alpha_t}\mathcal{R} = 0 \rightarrow \alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$$

ADABOOST TRAINING

$$\forall\ i : \omega_i^{t=1} = \frac{1}{n}$$

$$\text{for t} = 1, \ldots, \mathrm{T_{max}} \qquad \textbf{Do}$$

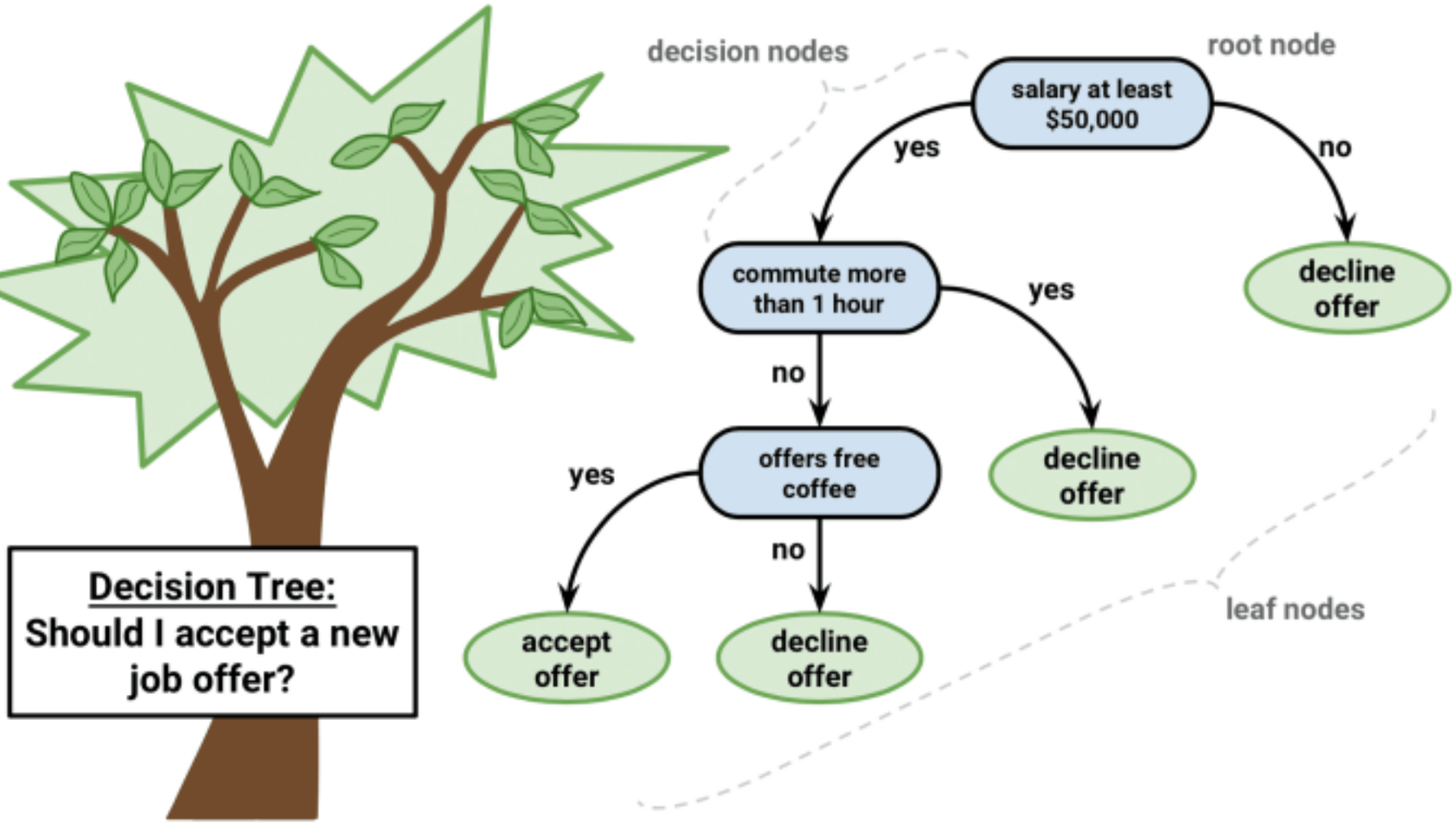
Run a classifier for : $\epsilon_t = \left[\sum_i \omega_i^{\tau} \mathbf{1}(Y_i \neq h_i))\right]$

Set: $\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$

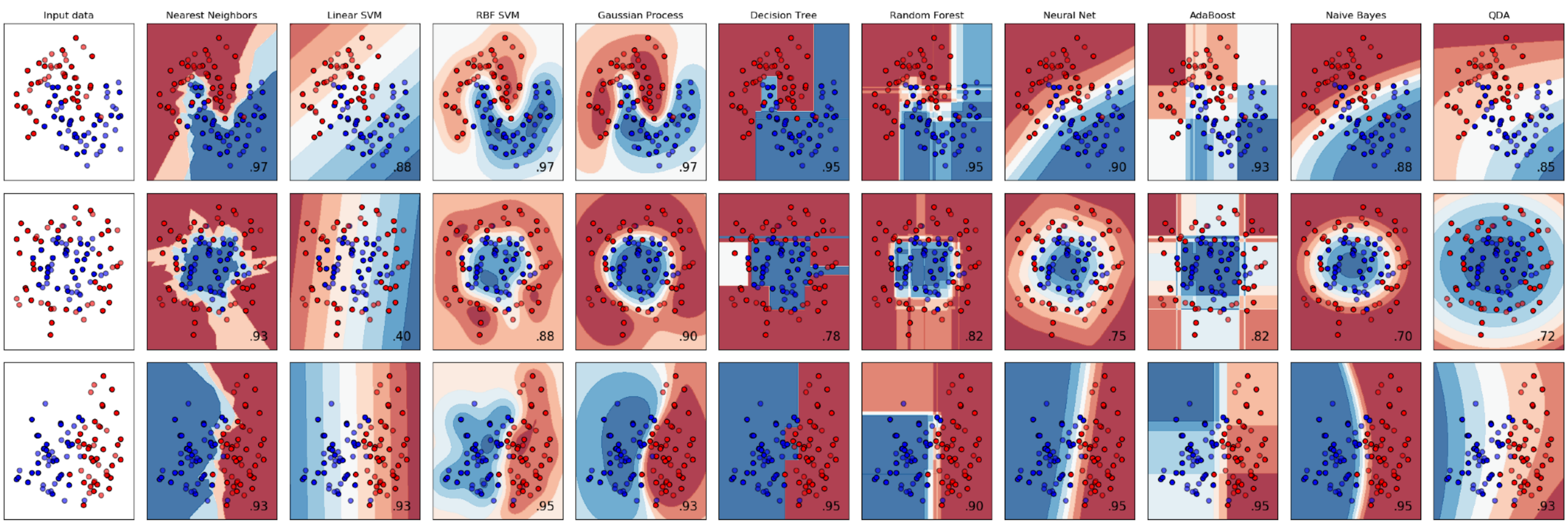
Agregate classifier: $H(.) = H^{t-1}(.) + \alpha_t h_t(.)$

Update weights: $\omega_i^{t+1} = \frac{\omega_i^t e^{-Y_i \alpha_t h_t(\vec{x}_i)}}{\sum_i \omega_i^t e^{-Y_i \alpha_t h_t(\vec{x}_i)}}$

Decision trees!



A tour on standard machine learning classifiers



From sk-learn