# Lecture #10: **Introduction to Support Vector Machines**

**Mat Kallada**

**STAT2450 - Introduction to Data Mining with R**

## Outline for Today

Support Vector Machines - Another way to draw lines

Multi-class Support Vector Machines

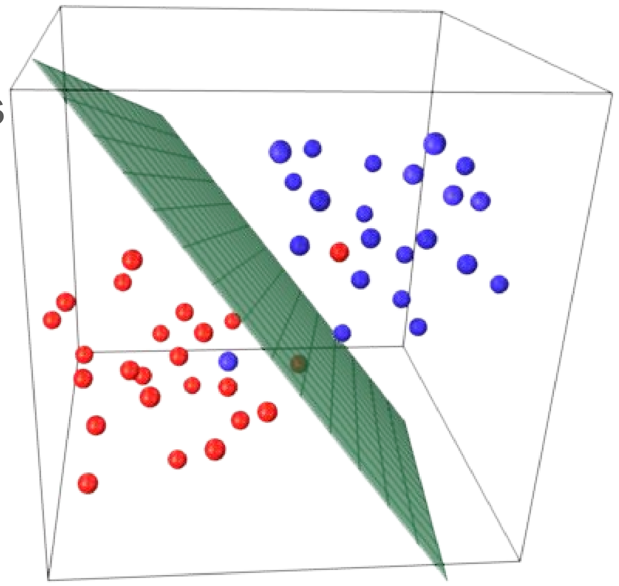
Kernels and Support Vector Machines

Support Vector Machines for Regression

Let’s change gears for a bit...

Remember, We learned **two ways to draw lines**

To solve regression or classification tasks



## Ways to Create Predictive Models

(I.e. Methods to solve the Supervised Data Mining Setup)

**Decision Trees**

Construct a decision tree which chops on the vector space

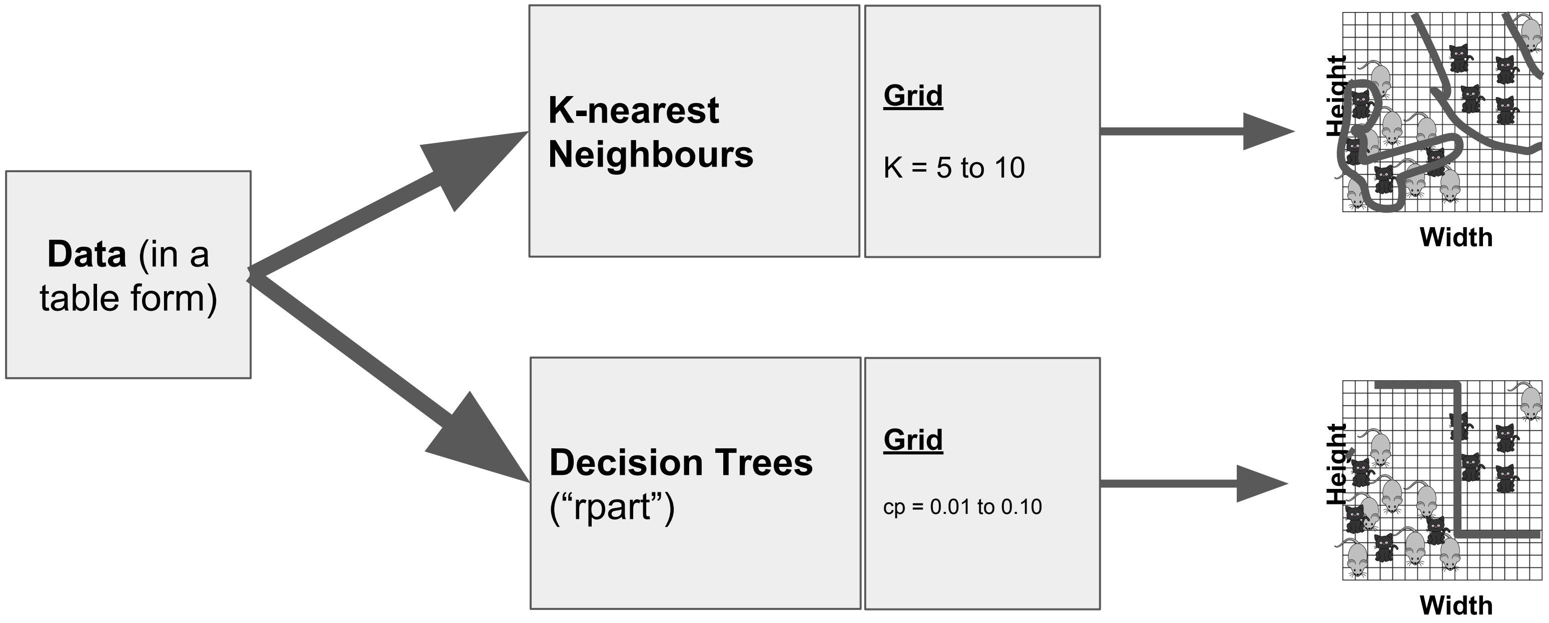
“Chops” are feature splits which minimize error

**K-Nearest Neighbours**

Look at the K-closest Points in Training Data

# **Supervised Data Mining**: The Line Drawing Contest

Who can draw the most “realistic” line?



## **Supervised Data Mining**: The Line Drawing Contest

**Why are lines a big deal again?**

The actual underlying hypothesis is unknown

Infinitely many ways we can create lines

Lots of features in our dataset makes it difficult to draw them by hand

## **Support Vector Machines:** How do they draw lines?

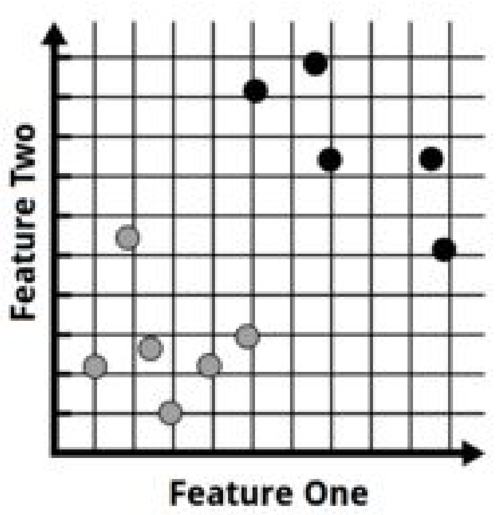
They are another supervised data mining technique used for either **regression** or **classification**

Invented by **Vladimir Vapnik** (now at Facebook)

## **Support Vector Machines:** How do they draw lines?

**Let's look at two-class classification first.**

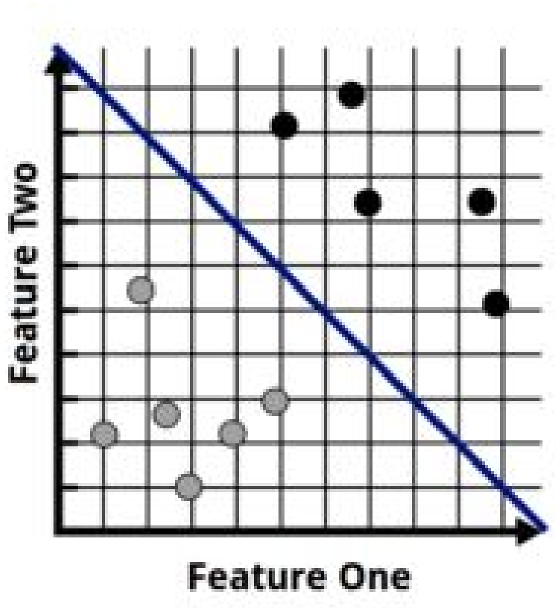
Consider the classification scenario below:



How could we create a classifier which best divides the two classes?

## **Support Vector Machines:** How do they draw lines?

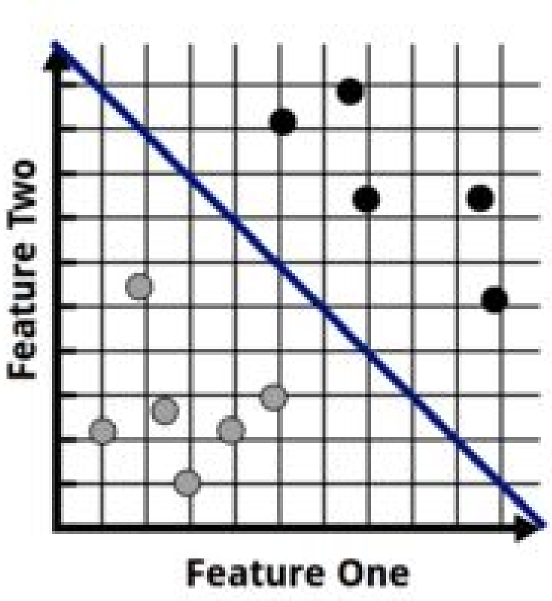
Let’s look at classification first. Consider the classification scenario below:



Hmm - probably right there.

## **Support Vector Machines:** How do they draw lines?

It’s **“right in between”** both classes and divides them both pretty well.



It is a line **equidistant** from the “outside” points of each class

## **Support Vector Machines:** How do they draw lines?

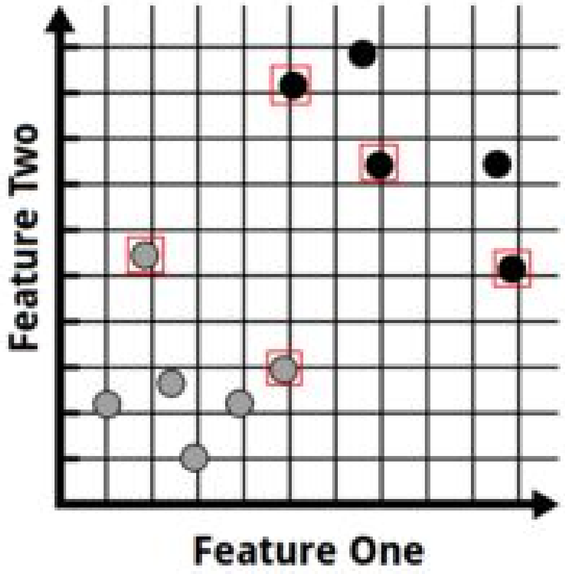
There are **two steps** for this:

1. Identify these “outside” points

2. Draw a line equidistant between both sets of outside points

## **Support Vector Machines:** How do they draw lines?

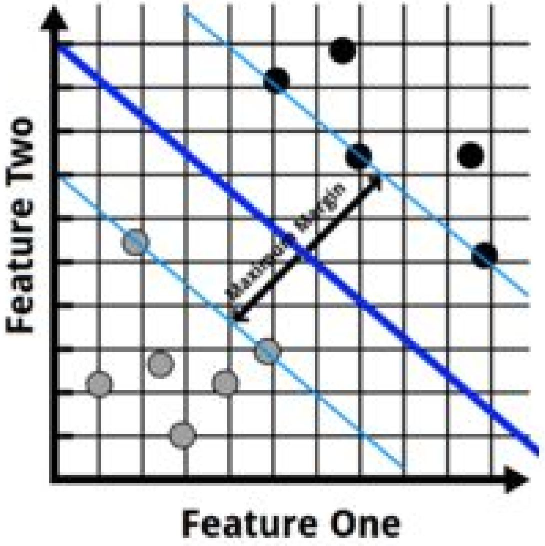
**Step 1:** Identify these “outside” points.



They are called the “support vectors” in the SVM model.

## **Support Vector Machines:** How do they draw lines?

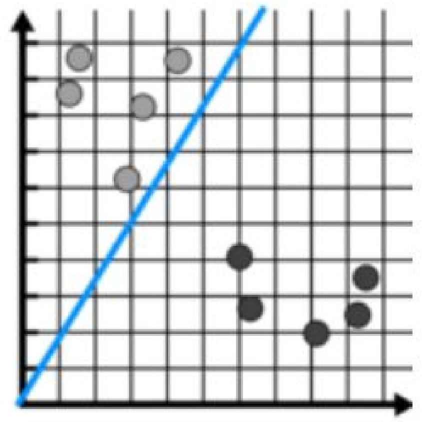
**Step 2:** Draw a line equidistant between support vectors

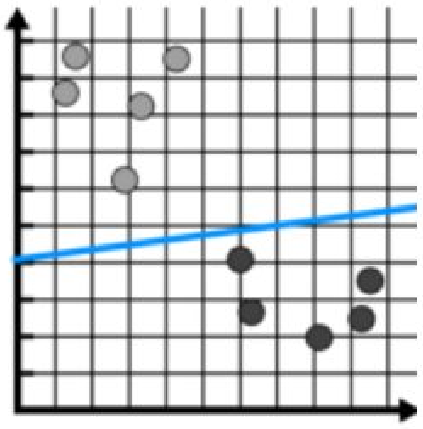


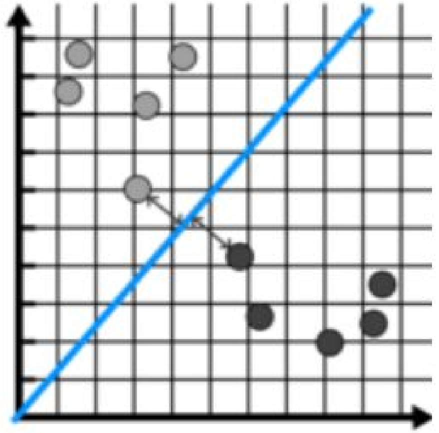
Draw the dividing line which is perpendicular to the margin with the furthest distance between the boundaries of the support vectors

## **Support Vector Machines:** How do they draw lines?

**Step 2:** Draw a line equidistant between support vectors

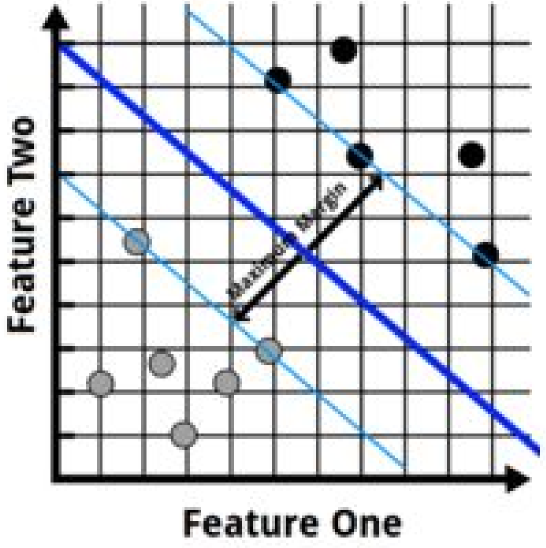

(e) This doesn't seem right. The decision boundary is too close to one class.


(f) Still no - with so many possible solutions - which one should we pick?


(g) This one. Why not use the line with a margin having the maximal distance away from each classes outer-point(s)? Seems like the most probable to me.

## **Support Vector Machines:** How do they draw lines?

**Step 2:** Draw a line equidistant between both classes



**Final Exam:** I'll ask a question related to how/why does the SVM draw this line.

## <u>**Step 2:**</u> Draw a line equidistant between both classes

To find this **'middle-ground' line**

Consider that we need to find the appropriate slope and intercept of the line with respect to an optimization task of <u>**maximizing the margin**</u> between support vectors.

**<u>Step 2:</u>** Draw a line equidistant between both classes

The idea is easy to understand, but there is beautiful math behind the scenes to find this line.

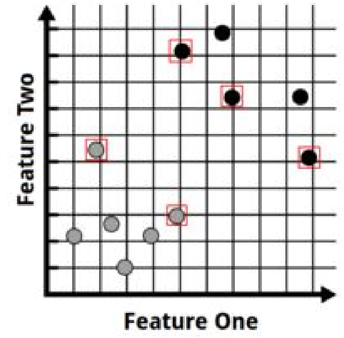
If you are interested in this, please have a look at Lecture Notes.

## **Support Vector Machines:** How do they draw lines?

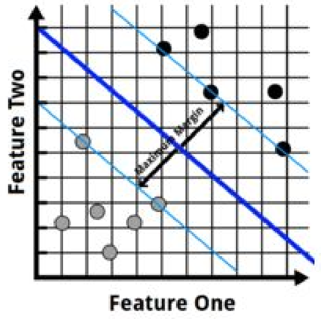
In a nutshell, a predictive model built with SVM has two steps:

**Step 1:** Find the “outside” data points (called the support vectors)

**Step 2:** Draw line equidistant between them.



**Step 1:** **Find Support Vectors**



**Step 2:** **Draw Equidistant Line**

## **Support Vector Machines:** Some issues

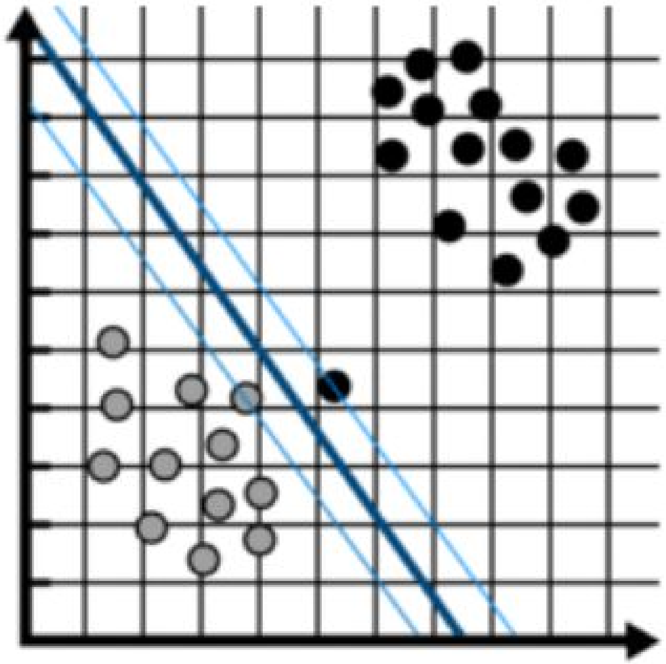
What about **noisy observations**?

Deformed cats may ruin the equidistant line

What if noise were chosen as a support vector?

## **Noisy Observations** can ruin the line

If the support vectors were noise, we may get something like this...



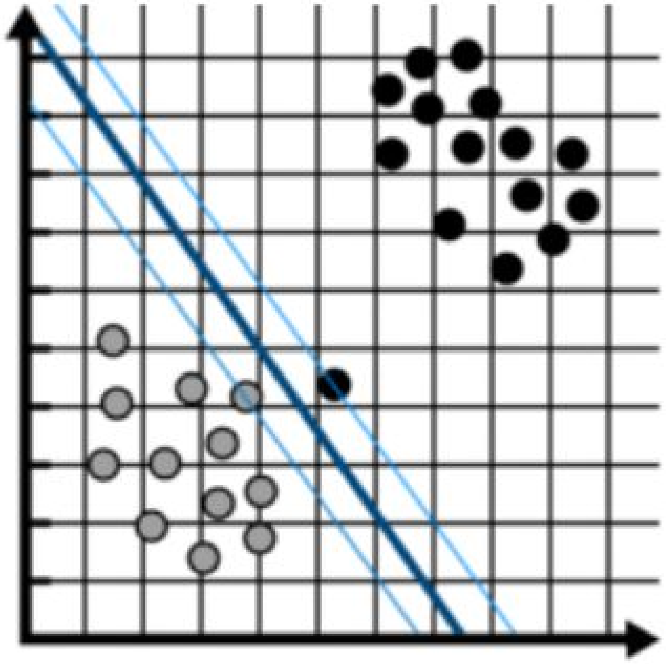
## **Noisy Observations** can ruin the line

A <u>**single noisy example**</u> messes up everything

The support vector is incorrectly chosen.

Our model is <u>invalid</u> and has **overfit**

It wouldn’t generalize very well to real-world cases



## **Support Vector Machines:** The Cost Parameter

Like K in K-nearest Neighbours

Like “cp” and “max depth” in Decision Trees

We have a hyperparameter to control complexity of the model

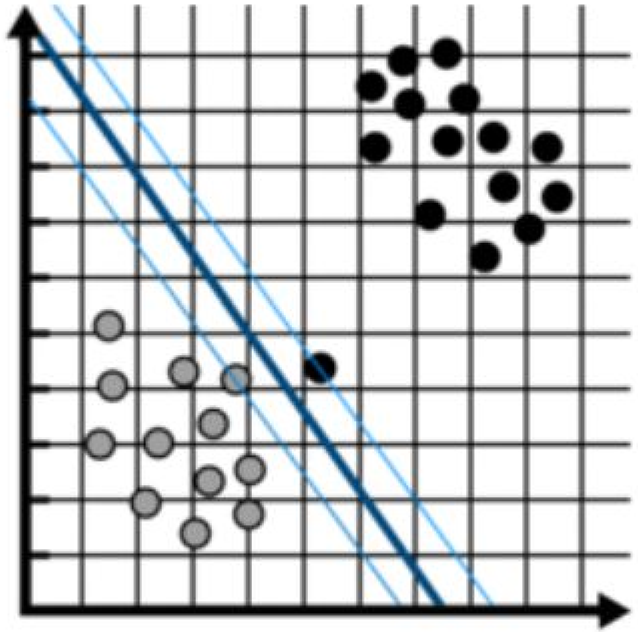
That is, how tolerable it is to noisy observations in our data

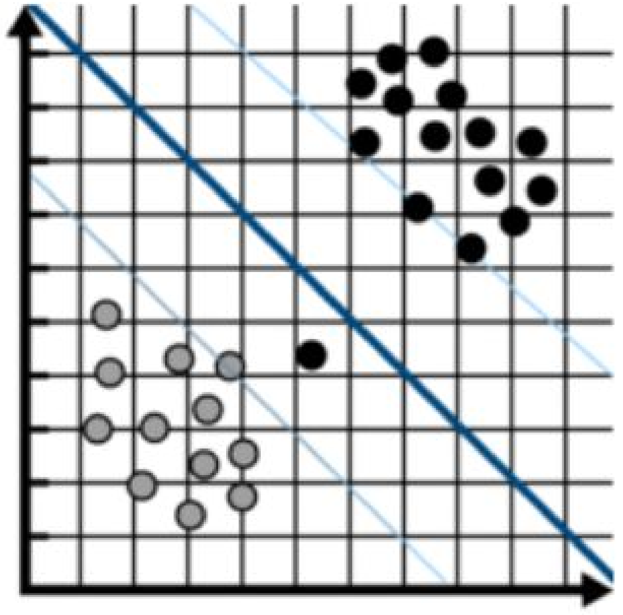
## **Support Vector Machines:** The Cost Parameter

We can specify the “**Cost**” hyperparameter or “C” to avoid noise.

A way of determining the **resistance** of the chosen support vectors to noise.

## **Support Vector Machines:** The Cost Parameter


C = 1,000


C = 0.001

## **Support Vector Machines:** The Cost Parameter

**Cost (C)**: When picking the support vectors, the “cost” of incorrectly classifying a data point.

## **Support Vector Machines:** The Cost Parameter

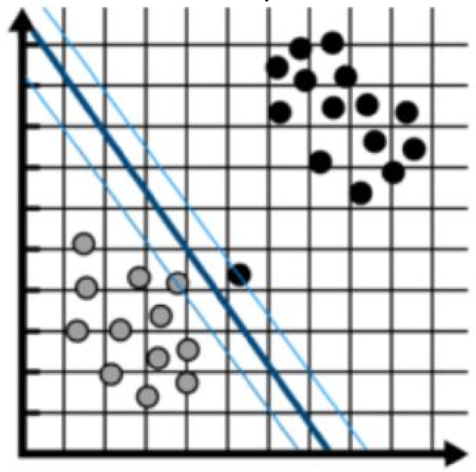
**Cost (C)**: When picking the support vectors, the “cost” of incorrectly classifying a data point.

Higher C values means that there is a **higher** cost to incorrectly classifying a training point. Too high means we’ll underfit.

Lower C values means that there is a **lower** cost to incorrectly classifying a training point. Too low means we’ll overfit.

## **Support Vector Machines:** The Cost Parameter

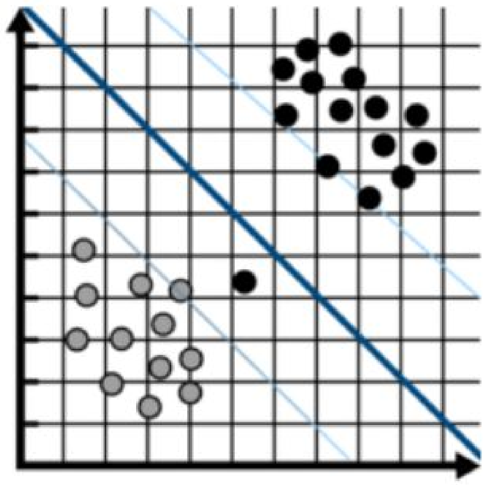
C = 1,000



The cost is **high** to make mistakes on the training data.
Since the cost is high, we can’t make mistakes
Let’s draw a line here

## **Support Vector Machines:** The Cost Parameter

C = 0.001



The cost is **low** to make mistakes on the training data.
Since the cost is low, we can make some mistakes
Let’s draw a line here

## **Support Vector Machines:** The Cost Parameter

“Cost” for SVMs is like K for KNN (or cp for Decision Trees)

We just try a bunch of different cost values until we find a good one

The one that will develop a model that works well.

Find one that avoids both overfitting and underfitting.

## Outline for Today

Support Vector Machines - Another way to draw lines

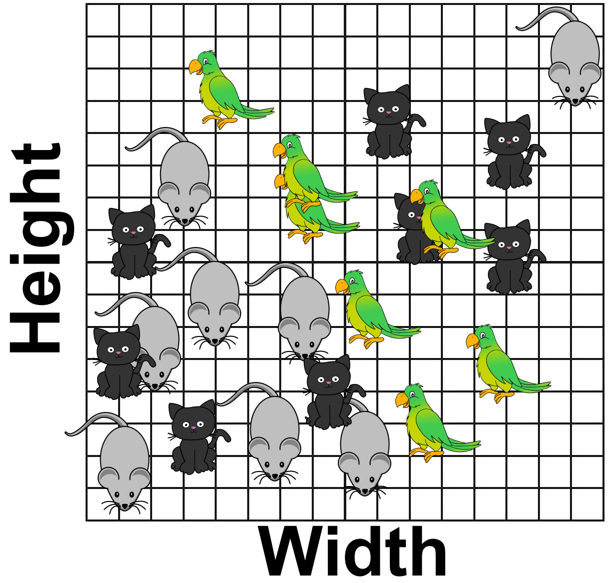
Multi-class Support Vector Machines ←

Kernels and Support Vector Machines

Support Vector Machines for Regression

## **Support Vector Machines:** Multi-class Problems

What if we had more than two classes?



How would we draw the line now?

## **Support Vector Machines:** Multi-class Problems

K-Nearest Neighbours and Decision Trees are naturally made to handle multi-class tasks

SVMs are not made for classification tasks with multiple classes.

## The “**One-vs-One**” trick for Multi-Class SVMs

We **can’t use SVMs** by themselves for multi-class problems

We can use a trick for SVMs to solve multiclass problems

## The “**One-vs-One**” trick for Multi-Class SVMs

We train three classifiers for all combination of classes:

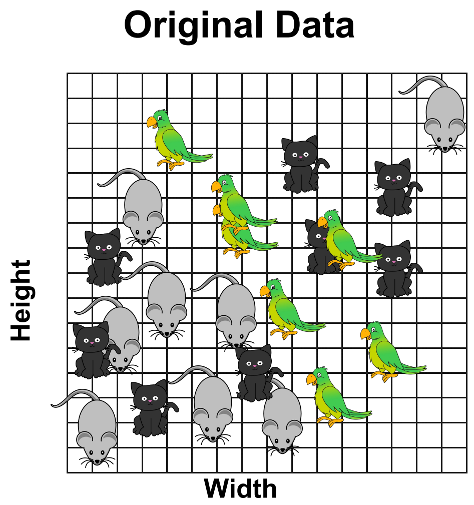
- Cat vs. Parrot
- Cat vs. Mouse
- Mouse vs. Parrot

Run SVM three different times

When an unknown observation comes in, we evaluate the point with each classifier.

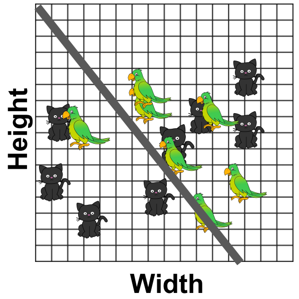
Do a majority vote as final prediction

# The “One-vs-One” trick for Multi-Class SVMs

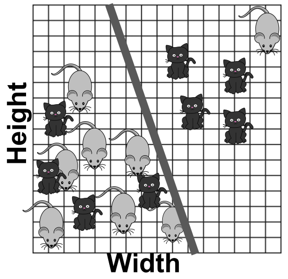


We must create three separate SVM models

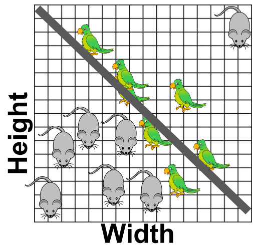
Cat vs Parrot



Cat vs Mouse



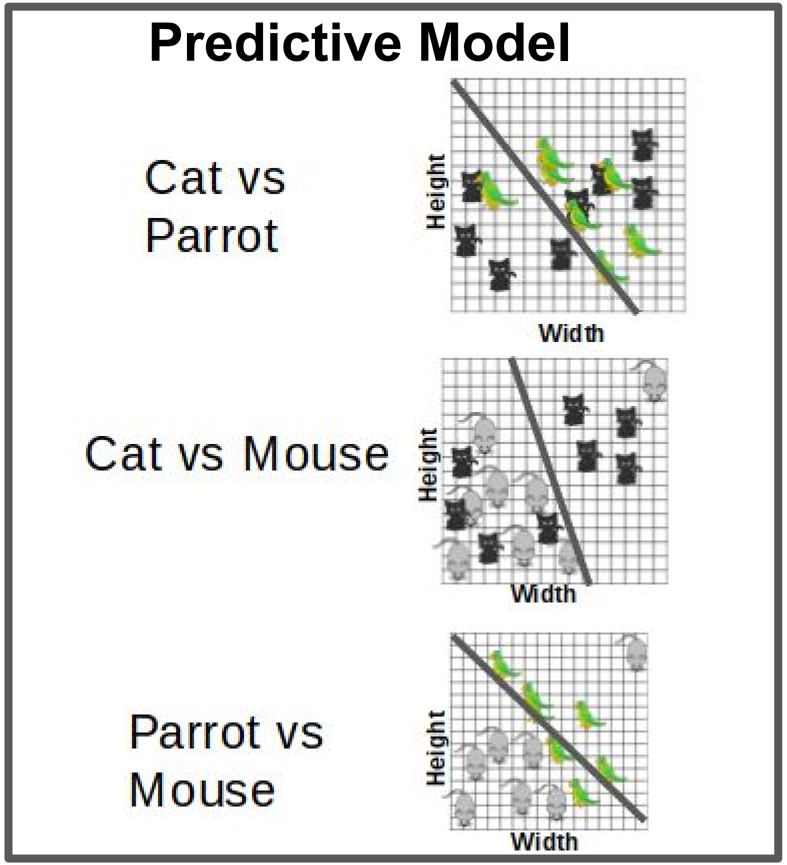
Parrot vs Mouse



# The “One-vs-One” trick for Multi-Class SVMs

Our predictive model is composed of sub-models.

Three different SVM sub-models



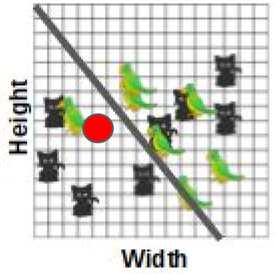
# The “One-vs-One” trick for Multi-Class SVMs

What Species is this?

<4.2, 5.4>

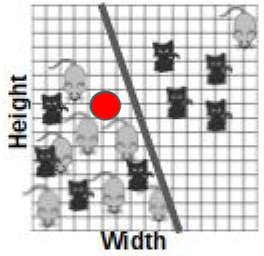
**Predictive Model**

Cat vs Parrot



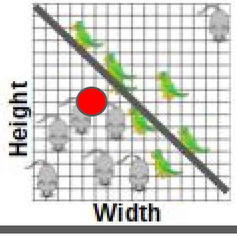
→ *Cat*

Cat vs Mouse



→ *Mouse*

Parrot vs Mouse



→ *Mouse*

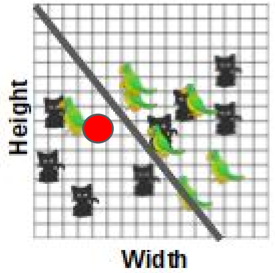
## The “One-vs-One” trick for Multi-Class SVMs

What Species is this?

<4.2, 5.4>

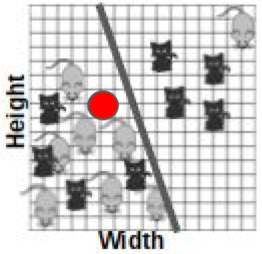
**Predictive Model**

Cat vs Parrot



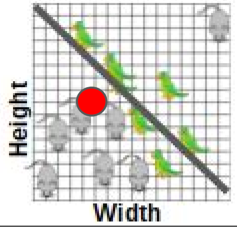
→ *Cat*

Cat vs Mouse



→ *Mouse*

Parrot vs Mouse



→ *Mouse*

→ *Mouse*

## **The “One-vs-One” trick**: Summary

This is sort-of like cheating

But SVMs cannot handle multi-classes by themselves

## **The “One-vs-One” trick**: Summary

SVM uses the One-vs-One trick for multi-class problems

Sub-models are built each possible class combination

Majority Vote afterwards for final prediction

R does this trick for us in the background