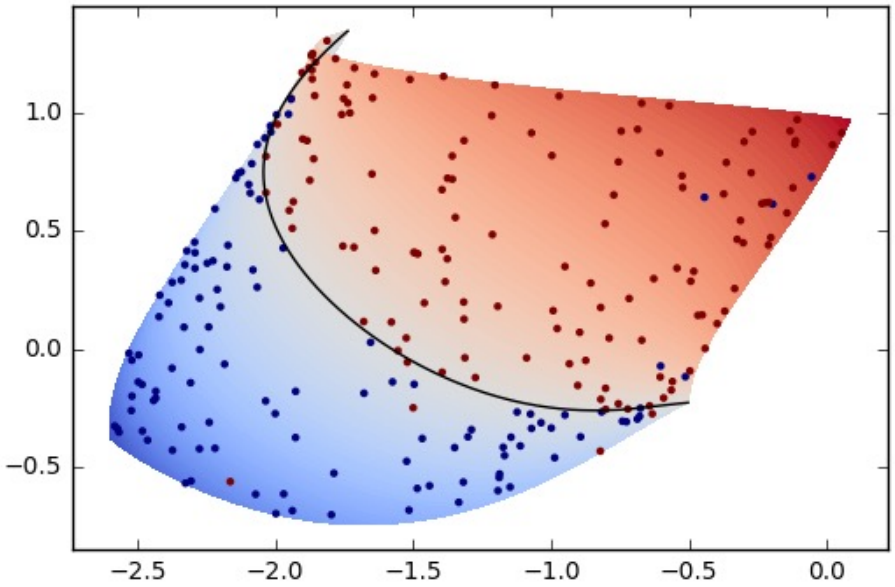
Lecture 6: Nearest Centroids Classifiers



Gavin Kerrigan

Spring 2023

Some materials courtesy Padhraic Smyth, Alex Ihler.

Announcements

- HW1 due tonight
  - Due 11:59PM – no late submissions
  - Submit PDF via Gradescope
  - Make sure all code/answers are visible

- HW2 released early next week
  - Problem 1: Nearest centroids on MNIST (today's lecture)

Today's Lecture

Classification

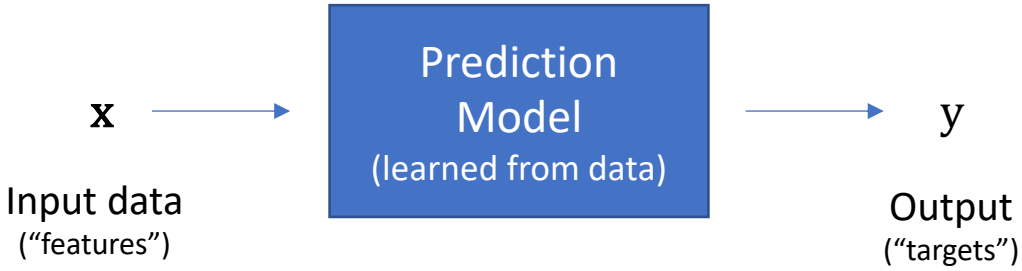
Feature Means and Distances

Nearest Centroids Classifier

Evaluating Classifiers

Supervised Learning

Supervised Learning: data comes in (**x**, y) pairs



Regression:
y is real-valued, e.g,
y is any value on the real-line, or
$y > 0$
$y \in [a,b]$

Classification:
y can take a finite set of K values, e.g.,
$K = 2$, $y \in \{0, 1\}$
$K = 10$, $y \in \{1, 2, \ldots, 10\}$

Class Labels

Classification: given a feature vector **x**, predict one of K labels for **x**

Example: classifying image of an animal, **labelset** = {“cat”, ”dog”, “other”}

In practice we index the labels by y = 1, 2, …. K

e.g., if K = 3, our labelset is {1, 2, 3} (rather than {“cat”, ”dog”, “other”})

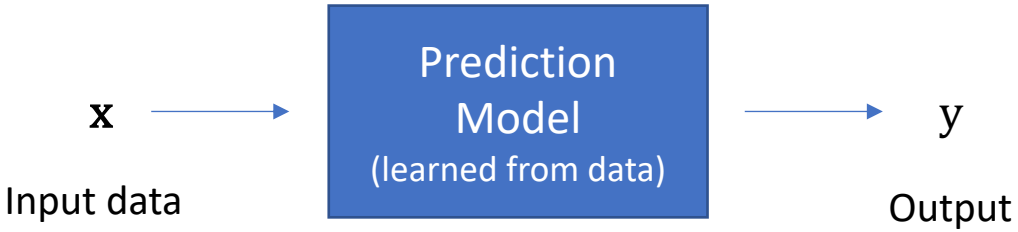
Exception: if K = 2, binary classification, we traditionally use a labelset = {0,1}

Notation:

$y \in \{1, 2, \ldots K\}$ means that y takes values in the labelset {1, 2, … K}

y = c, means that the label is c

Examples of Classification



Text of email	→	Spam or not (K = 2)
Pixels in image	→	Type of animal (K = 500 or more)
Speech signal	→	English word (K = 100,000 or more)
Loan application	→	Grant loan or not (K = 2)
Medical data	→	Diagnosis (K = 2 or more)

Data Matrix

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \cdots & \cdots & \cdots & \cdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix}$$

Caps and bold font for a matrix

n rows: each one is a feature vector for a different individual or object

d columns = d features

x_{ij} is the jth feature value (column) for the ith row

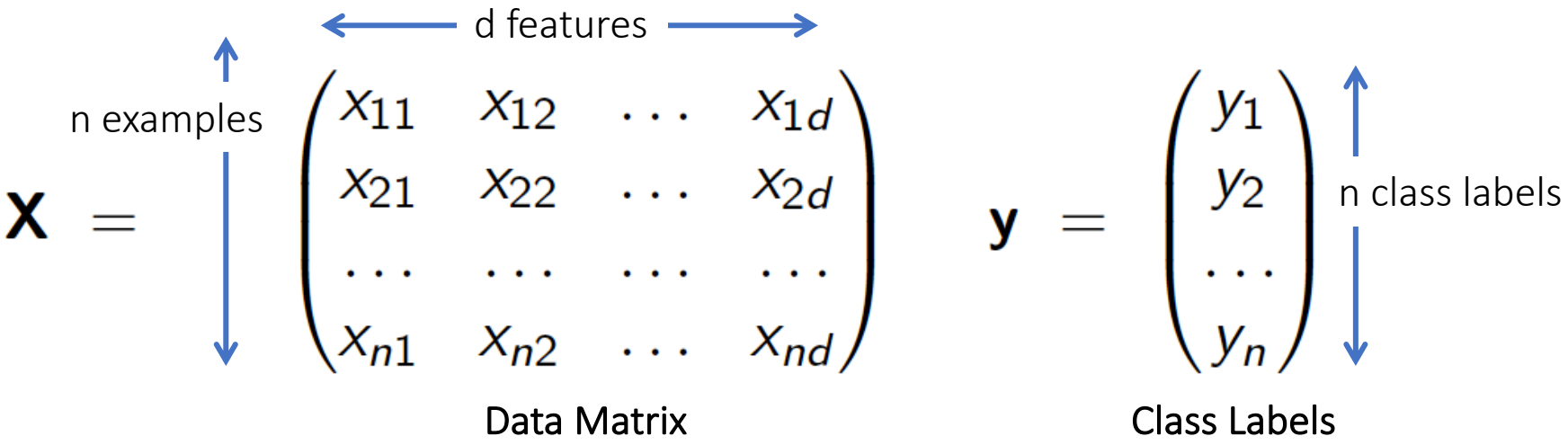
$\boldsymbol{x}_i$ is the feature vector for the ith row

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

Bold font for a vector

n rows: one target for each datapoint

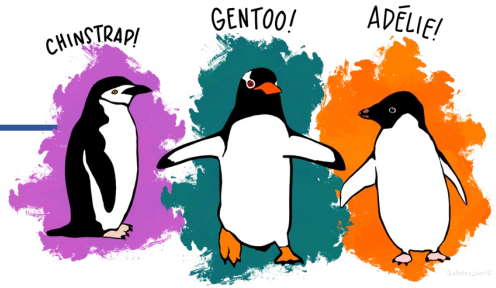
Features + Class Labels: Labeled Dataset



Example

$$\mathbf{X} = \begin{pmatrix} 21.4 & 6.1 & 200 \\ 28.1 & 5.5 & 145 \\ 24.7 & 2.2 & 94 \\ 32.0 & 4.2 & 155 \\ \cdots & \cdots & \cdots \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} 3 \\ 1 \\ 2 \\ 3 \\ \cdots \end{pmatrix}$$

Return of the Penguins



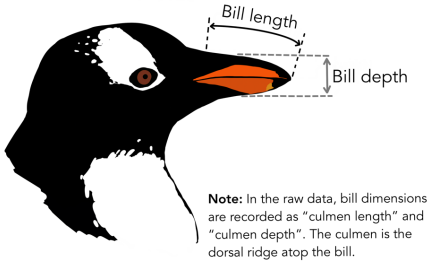
- A small dataset useful for exploration

- Data Summary
  - 344 penguins
  - Each described by 4 real-valued features (”attributes”)
    - Bill-length, bill-depth, flipper-length, body-mass
    - Other attributes per penguin: gender, year, island
  - Each penguin is in 1 of 3 classes: adelie, chinstrap, gentoo

  - Natural to put data in a 344 x 4 table
    - 344 rows (penguins) by 4 columns (features)
  - ..with additional 344 x 1 column of class labels

For more information see: github.com/allisonhorst/palmerpenguins/blob/main/README.md

Data Matrix for Penguin Data



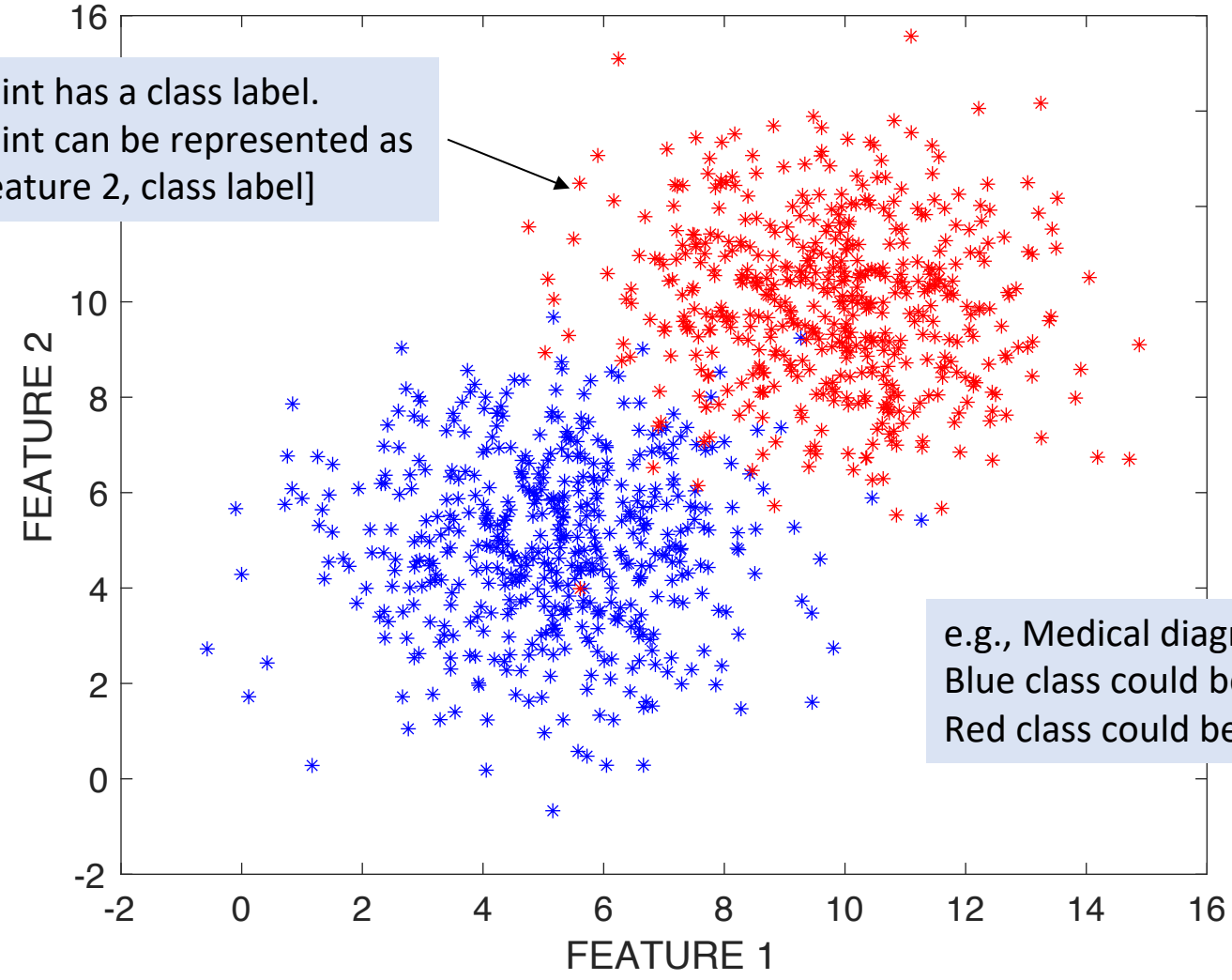
bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
39.1	18.7	181.0	3750.0
39.5	17.4	186.0	3800.0
40.3	18.0	195.0	3250.0
NaN	NaN	NaN	NaN
36.7	19.3	193.0	3450.0

Here are the first 5 rows for the real-valued variables in the 344 x 4 data array for the penguins

Each row is a different penguin

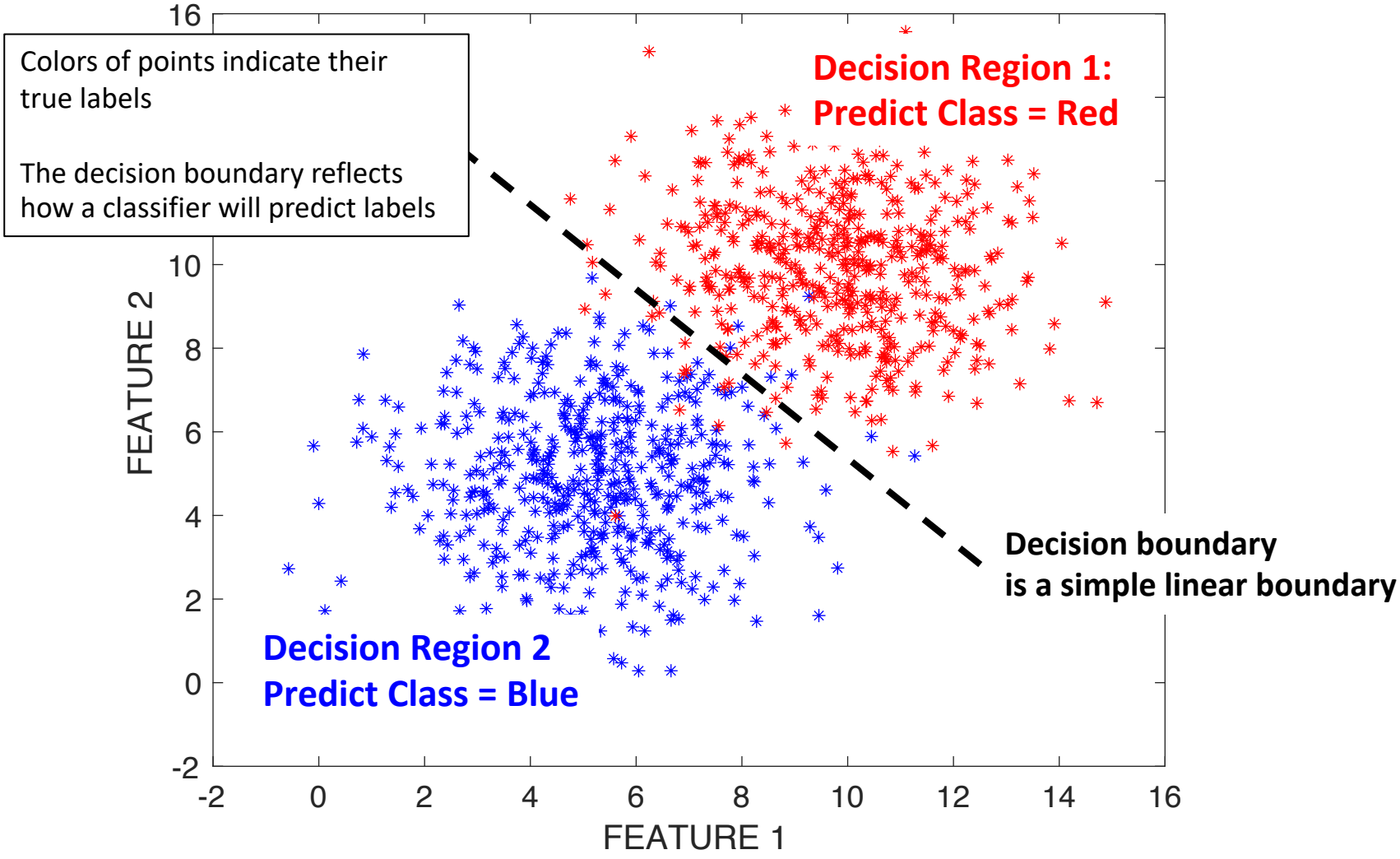
Labeled Data: Feature Vectors + Class Labels

Each data point has a class label.
Each data point can be represented as
[feature 1, feature 2, class label]

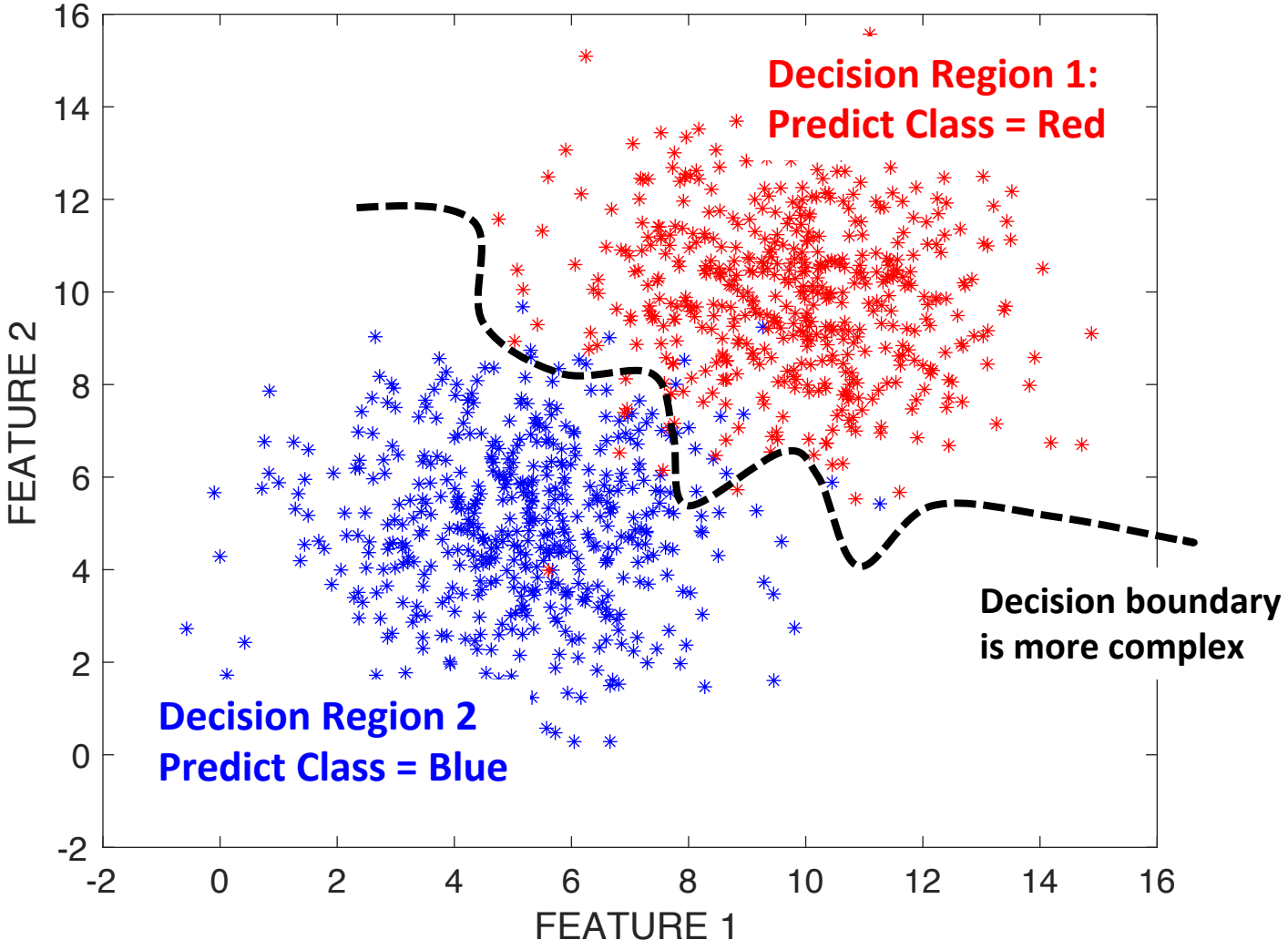


e.g., Medical diagnosis:
Blue class could be healthy
Red class could be not-healthy

Example of A Simple Classifier



A More Complex Classifier



Classification

- The number of features is known as the **dimensionality** of the classification problem
- In our examples we just have two dimensions (two features)
  - So we can easily just draw a decision boundary on the plot
- In practice we will be in a much higher-dimensional space (many features)
  - So we can't just plot the data and look at it
- A typical machine learning problem might have 100 or 1000 or even 1 million features (e.g., for text)
- This is where machine learning excels: searching these high-dimensional feature spaces for good decision boundaries

A Classifier as a Mapping

Classifier is a mapping from features to predicted labels

Notation: the "hat" over y indicates that $\hat{y}$ is not the true y, but instead a prediction or estimate

We can think of a classifier as a **mapping** from a d–dimensional space to one of K labels

If the mapping is smooth (it usually is), local regions in **x** get mapped to the same label

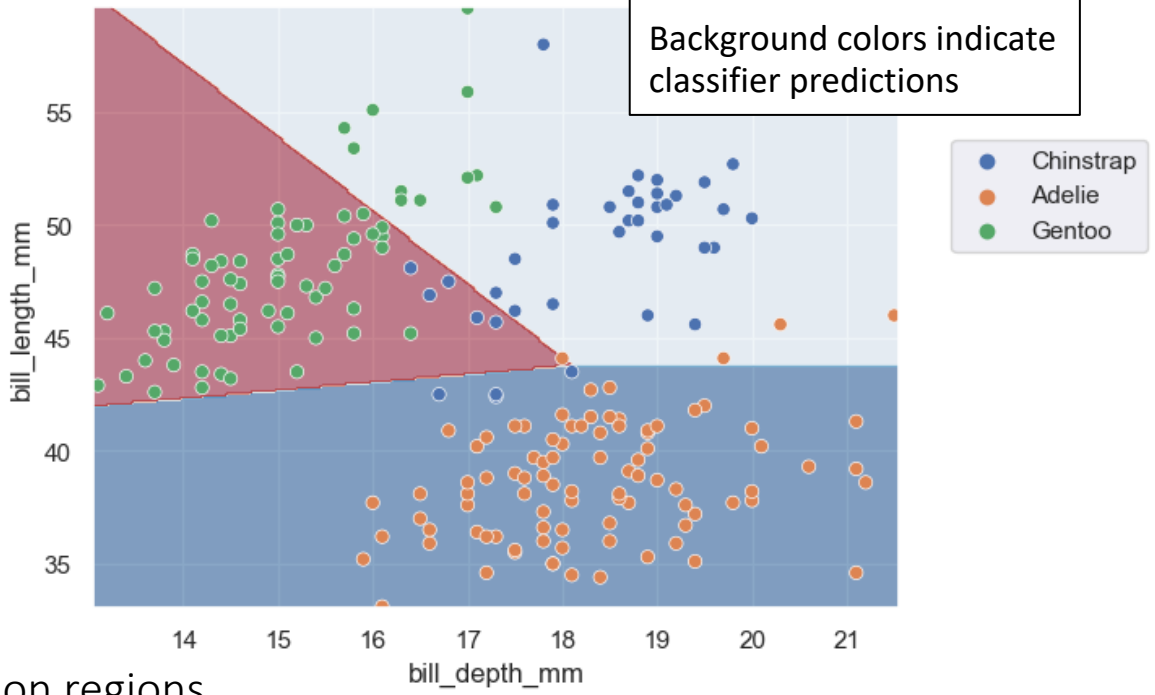
So we can think of **decision regions**: regions in the input space x all mapped to some label

Decision Regions for a Penguin Classifier

Each feature vector is mapped to a class label

So, each class "claims" the regions of input space for points assigned to it

These are "decision regions"



Above we see an example of decision regions for a classifier for the penguin data, for two of the features

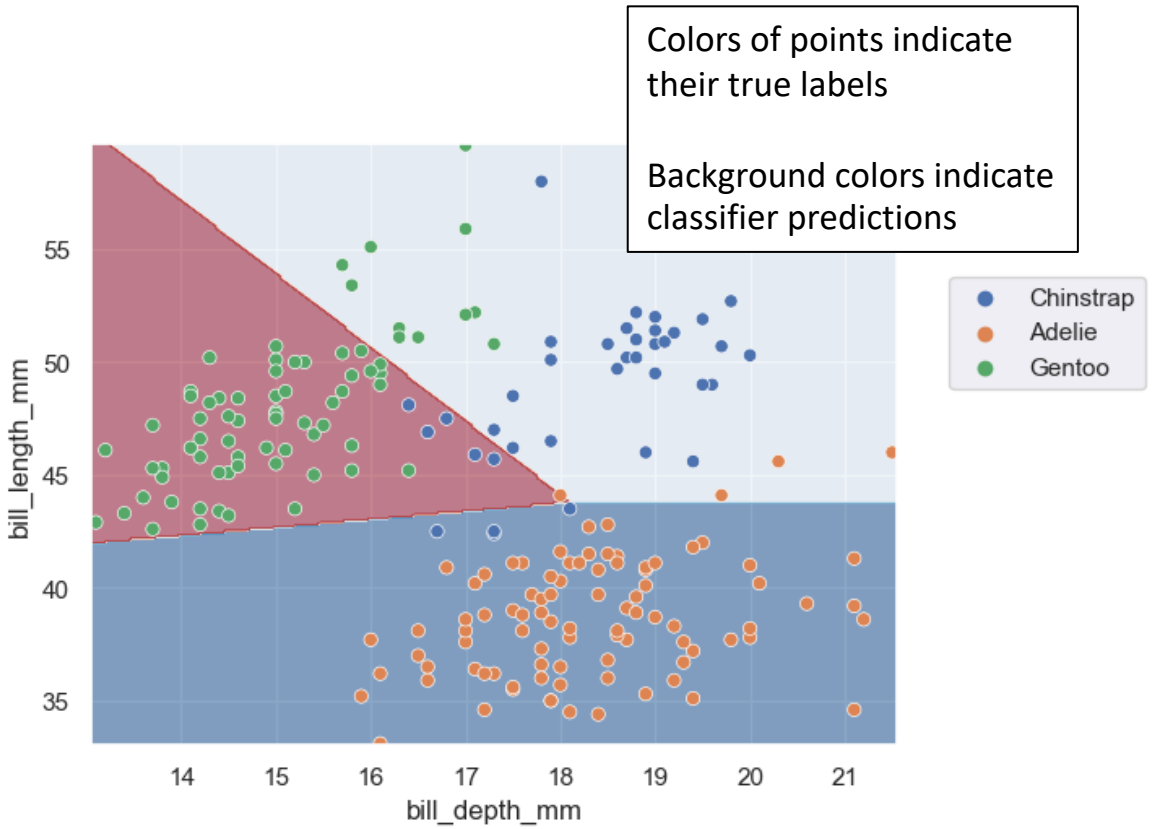
Decision Boundaries for a Penguin Classifier

Decision boundaries are (not surprisingly) defined as the boundaries of a classifier's decision regions

Points on the boundary are equally likely to belong to each class on either side

In the example on the right the boundaries appear to be straight lines (i.e., linear)

In the general case we can also have non-linear boundaries



Questions?

Today's Lecture

Classification

Feature Means and Distances

Nearest Centroids Classifier

Evaluating Classifiers

Mean Values of Individual Features

Visually, to compute the mean value of feature 2 …. we sum all values in 2nd col and divide by n

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \cdots & \cdots & \cdots & \cdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix}$$

Mathematically,
mean value for feature j (e.g., j=2)

$$\mu_j = \frac{1}{n} \sum_{i=1}^{n} x_{ij}$$

Sum over each row of matrix **X** (index i is for rows)

Only sum the elements of column j in matrix **X**

Means and Mean Vectors

Mean value for feature j

$$\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij}$$

average over rows for column j (feature j) in **X**

Mean feature vector
(will also be referred to as the "centroid")

$$\boldsymbol{\mu} = (\mu_1, \ldots, \mu_d)$$

$$= \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$$

average of the n vectors in **X**

Simple numerical example

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} 21.4 & 6.1 & 200 \\ 28.1 & 5.5 & 145 \end{pmatrix}$$

Mean value for feature j = 1

$$\mu_1 = (21.4 + 28.1)/2 = 24.75$$

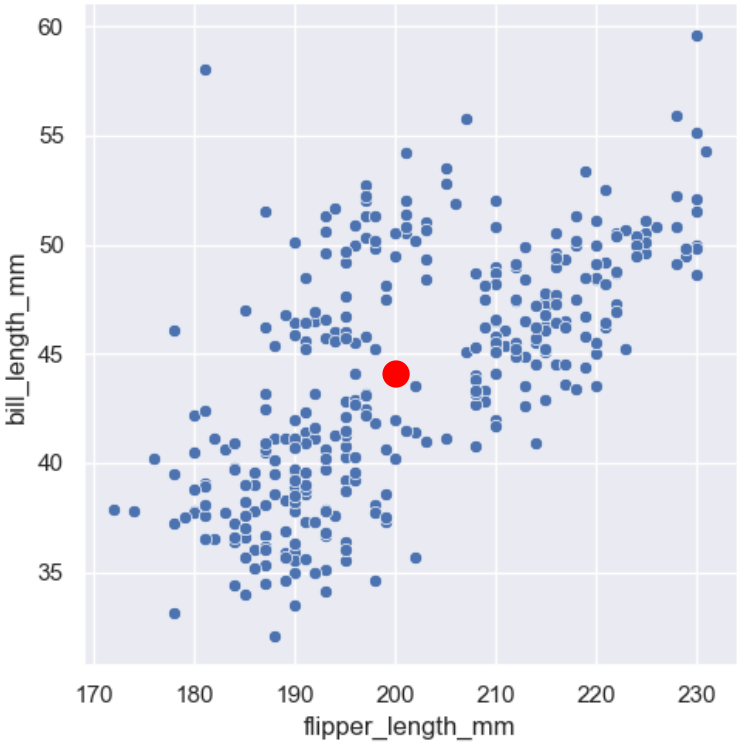
Mean feature vector (or centroid)

$$\boldsymbol{\mu} = (24.75\ ,\ 5.8\ ,\ 172.5)$$

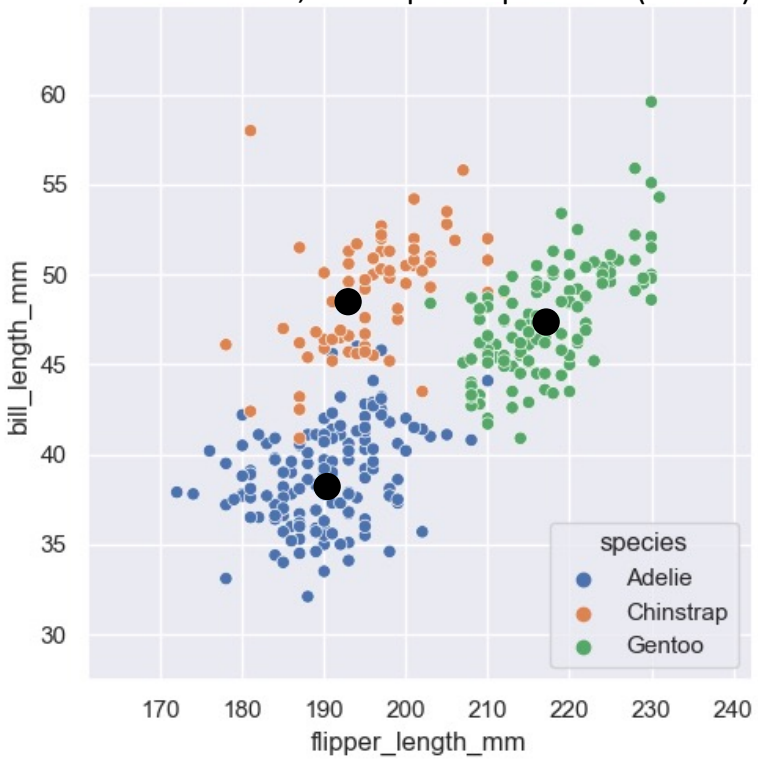
(in Python can use np.mean(X,axis=0),
where X is a 2 x 3 array)

Visual Example of Centroids (Means) in 2d

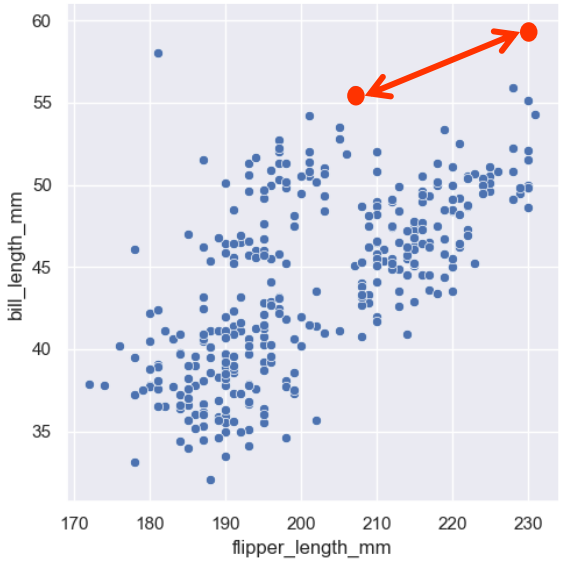
Single centroid for all 344 penguins



3 centroids, one per species (label)



Euclidean Distance between Vectors



Euclidean distance:

$$\text{dist}_E(\mathbf{x}, \mathbf{z}) = \sqrt{\left(\sum_{j=1}^{d}(x_j - z_j)^2\right)} = \left(\sum_{j=1}^{d}(x_j - z_j)^2\right)^{1/2}$$

Squared difference of the values of the two vectors **x** and **z** in each dimension

Euclidean (straight line) distance is the simplest notion of distance between 2 points (vectors).

The definition above is for any d-dimensional space

Euclidean Distance: Examples in Python

```
>>> import numpy as np
>>>  x = np.array( [1, 2, 5] )
>>> z1 = np.array( [3 ,3, 7] )
>>> z2 = np.array( [5 ,3, 10] )

>>> np.sqrt( np.sum( (x - z1)**2, axis=0) )
3.0


>>> np.sqrt( np.sum( (x - z2)**2, axis=0) )
6.48
```

This is one simple way to compute Euclidean distance in Python

There are other alternatives to do the same thing, e.g., `np.linalg.norm(x - z)`

General Distances and Norms

A general way to define distances is via *Lp* norms

The *Lp* norm of the difference between vectors x and z is defined as

$$\|\boldsymbol{x} - \boldsymbol{z}\|_p = \left(\sum_{j=1}^{d} |x_j - z_j|^p \right)^{1/p}$$

p=1 gives us Absolute Distance

$$\|\boldsymbol{x} - \boldsymbol{z}\|_1 = \sum_{j=1}^{d} |x_j - z_j| = \text{dist}_A(\boldsymbol{x}, \boldsymbol{z})$$

p=2 gives us Euclidean Distance

$$\|\boldsymbol{x} - \boldsymbol{z}\|_2 = \left(\sum_{j=1}^{d} (x_j - z_j)^2 \right)^{1/2} = \text{dist}_E(\boldsymbol{x}, \boldsymbol{z})$$

Euclidean Distances for Penguins

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
$\mathbf{x}_1$ →	39.1	18.7	181.0	3750.0
$\mathbf{x}_2$ →	39.5	17.4	186.0	3800.0
$\mathbf{x}_3$ →	40.3	18.0	195.0	3250.0

$\text{dist}_E(\mathbf{x}_1, \mathbf{x}_2) = 50.3$

$\text{dist}_E(\mathbf{x}_1, \mathbf{x}_3) = 500.2$

Distance of $\mathbf{x}_1$ to $\mathbf{x}_3$ is 10 times distance to $\mathbf{x}_2$

Say we measured body_mass in kilograms instead of grams.
i.e., we divide the body_mass values by 1000.
This has the effect of downweighting the body_mass feature in distance calculations

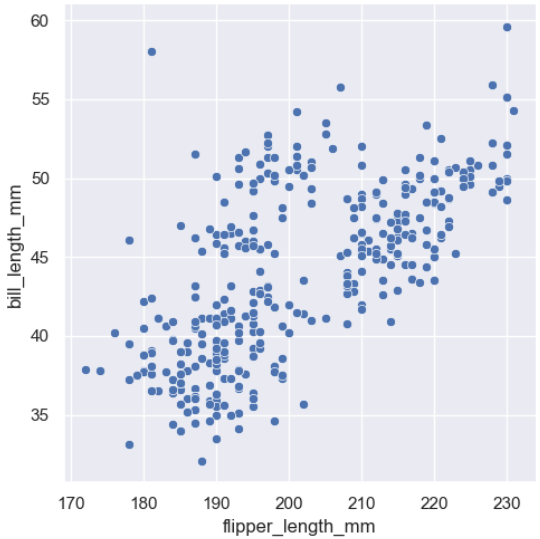
$\text{dist}_E(\mathbf{x}_1, \mathbf{x}_2) = 5.2$

$\text{dist}_E(\mathbf{x}_1, \mathbf{x}_3) = 14.1$

$\mathbf{x}_2$ is still closer to $\mathbf{x}_1$ than $\mathbf{x}_3$, but now by a smaller factor
…..so **units and scale matter when computing distance**

Geometric Concepts in Machine Learning

Data points (feature vectors) **x** exist in a d-dimensional space



Geometric thinking is natural in this space, e.g,

Distances between points (vectors)

Clouds or clusters of points

Boundaries (e.g., lines, planes) that separate points in the space

… and so on

Questions?

Today's Lecture

Classification

Feature Means and Distances

Nearest Centroids Classifier

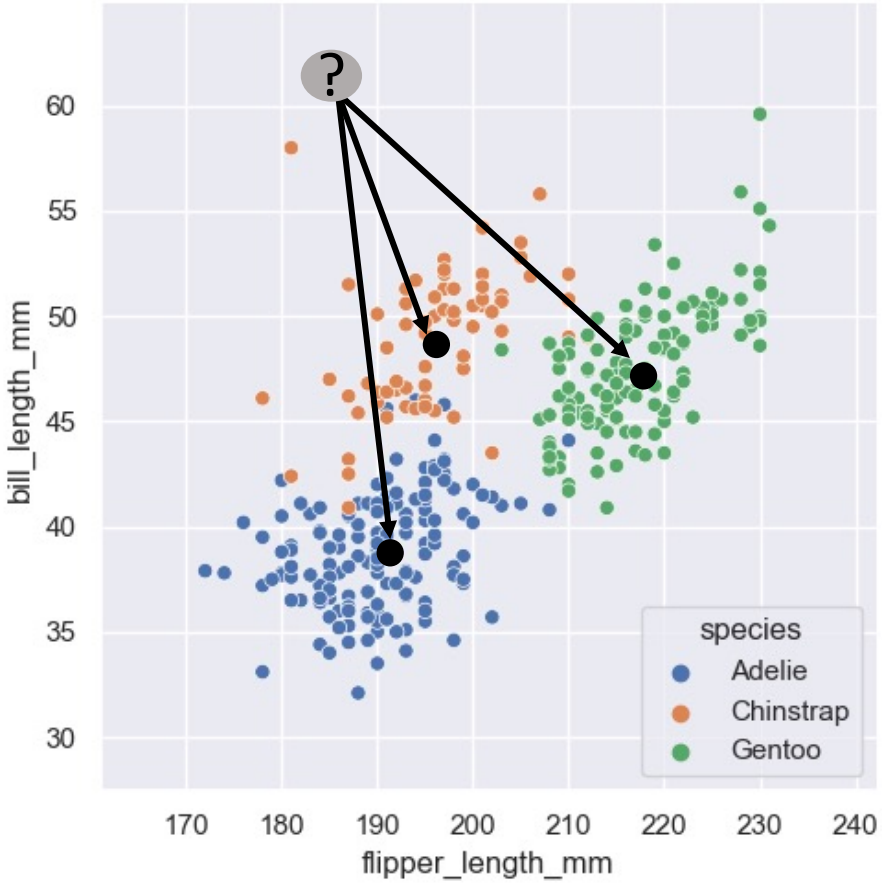
Evaluating Classifiers

The Nearest-Centroid Classifier

This is a very simple idea

Compute the centroid for each class
(i.e. centroid of datapoints in that class)

For a new unlabeled datapoint, assign
it to the class it is closest to
(in terms of Euclidean distance)

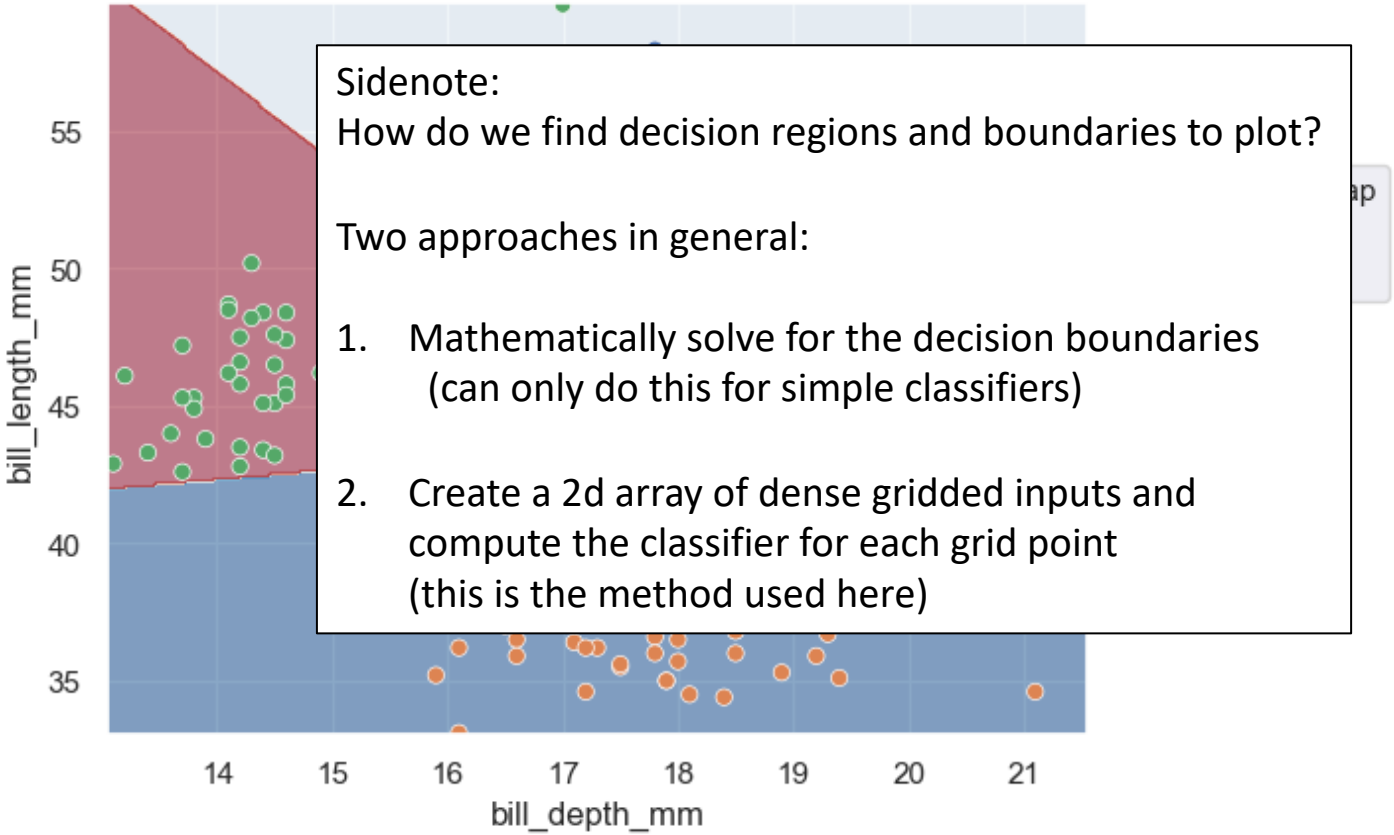


Even though it is very simple, it is plausible

- Every class is represented by a "template", i.e. its centriod

Example: Nearest Centroid Classifier on Penguin Data

Black points = class centroids
Colored background regions: decision regions learned by the classifier
Colored points: true labels of points (not all points end up in correct region)



Sidenote:
How do we find decision regions and boundaries to plot?

Two approaches in general:

1. Mathematically solve for the decision boundaries
   (can only do this for simple classifiers)

2. Create a 2d array of dense gridded inputs and compute the classifier for each grid point
   (this is the method used here)

Training the Nearest-Centroid Classifier

(simple example of pseudocode)

```
Nearest_Centroid_Training(X, y)    # X, y = labeled training dataset

C = length( unique(y) )            # find the number of class labels C

For c = 1,…, C                     # loop through each of the class labels
                                   # (this assumes the labels run from 1 to C)

    index_c = find(y == c)         # define index for which labels have value c

    X_c  = X(index_c, : )          # find rows in X that belong to class c

    Centroid_c =  calculate_centroid(X_c)   # compute centroid for class c

end

Return( Centroid_1,…,Centroid_C )   # return the C Centroids
```

Note: we previously used K instead of C for the number of classes

Prediction with the Nearest-Centroid Classifier

```
Nearest_Centroid_Prediction(x, C centroids)      # x is unlabeled

min_distance = infinity          # initialize the minimum distance

For c = 1,…,C                    # loop through each of the class labels

    d_c = distE( x, Centroid_c)  # find Euclidean distance of x to Centroidc

    If d_c <  min_distance,
        yhat  =  c
        min_distance = d_c
    Endif

end

Return: yhat                 # yhat = predicted label for x
                             #      = class whose centroid is closest to x
```

Questions?

Today's Lecture

Classification

Feature Means and Distances

Nearest Centroids Classifier

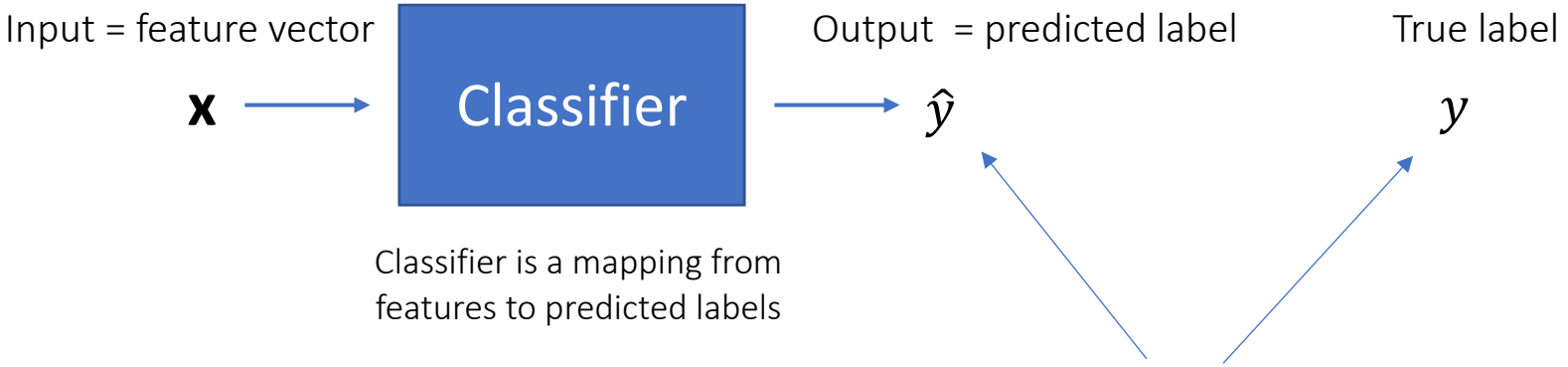
Evaluating Classifiers

A Classifier as a BlackBox

Classifier is a mapping from features to predicted labels

Note on notation: the "hat" over y here indicates that $\hat{y}$ is not the true y, but instead a prediction or estimate

Accuracy of a Classifier



Compare
If $\hat{y} = y$, then Acc = 1
If $\hat{y} \neq y$, then Acc = 0

In math notation:
Acc = $I(\hat{y}, y)$

Indicator function
Evaluates to 1 if arguments are equal
Evaluates to 0 otherwise

(here, Acc = accuracy of the classifier on one example = 0 or 1)

Average Accuracy of a Classifier

In general, we want to know classifier accuracy on average, over many **x** examples

We can estimate this on a **validation/test set** to get accuracy
Say the test dataset has N_{test} labeled examples, i.e., feature vectors with labels

Accuracy on the ith datapoint from the test set is $Acc_i = I(\hat{y}_i, y_i)$

Average accuracy is defined as:

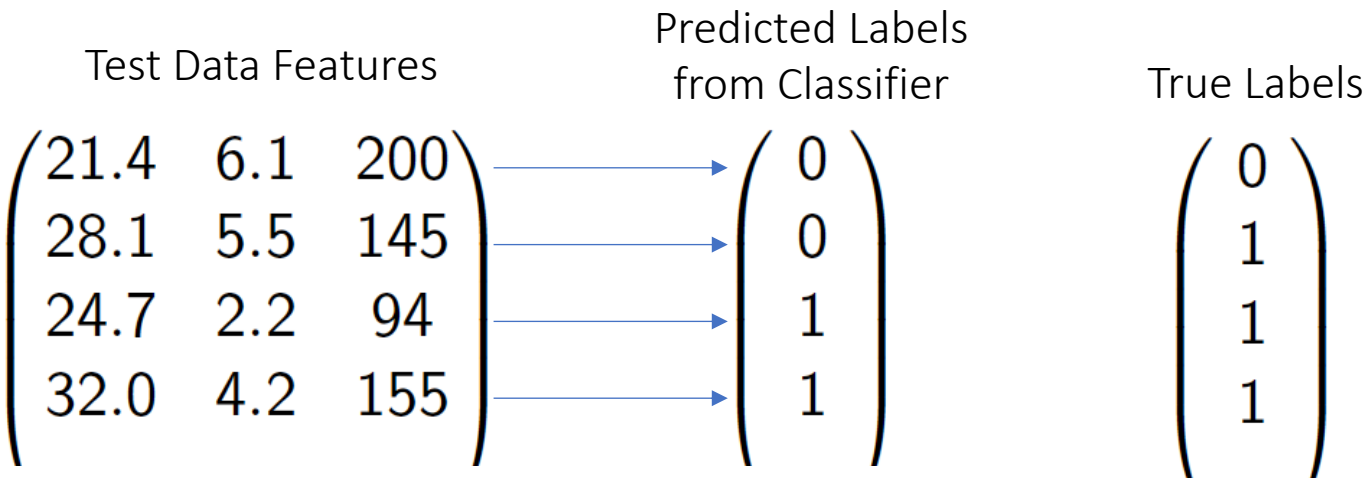
$$Acc = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} Acc_i$$

$$= \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} I(\hat{y}_i, y_i)$$

Sum over all test examples

Compute Accuracy for each example

Simple Example of Accuracy Calculation

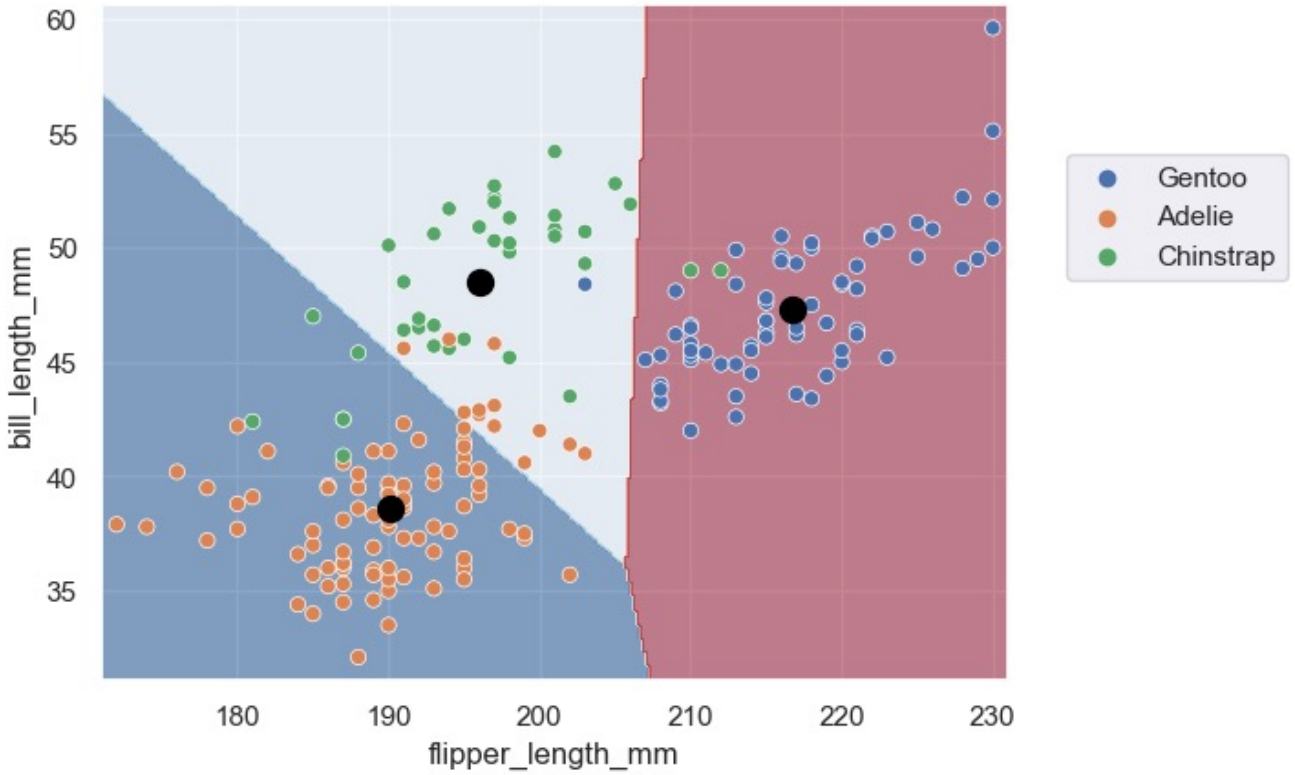


$$\text{Acc} = ¼(\ 1 + 0 + 1 + 1) = 3/4 = 0.75$$

Note:

1. We also often look at error rate = Err = 1 – Acc
2. We could express Accuracy as a fraction (0 to 1) or as a percentage

Nearest-Centroid Classifier



Train on Ntrain = 200 examples
Test on Ntest = 133 examples

Error on test set = 13.5%

Scaling (Standarizing) Features

- Notice on the previous slide that the scale, i.e., max – min, of the flipper feature (x-axis) is about 50 and that of bill-length feature (y-axis) is about 25

- So the flipper feature is getting twice as much weight in nearest-centroid distance calculations as bill-length

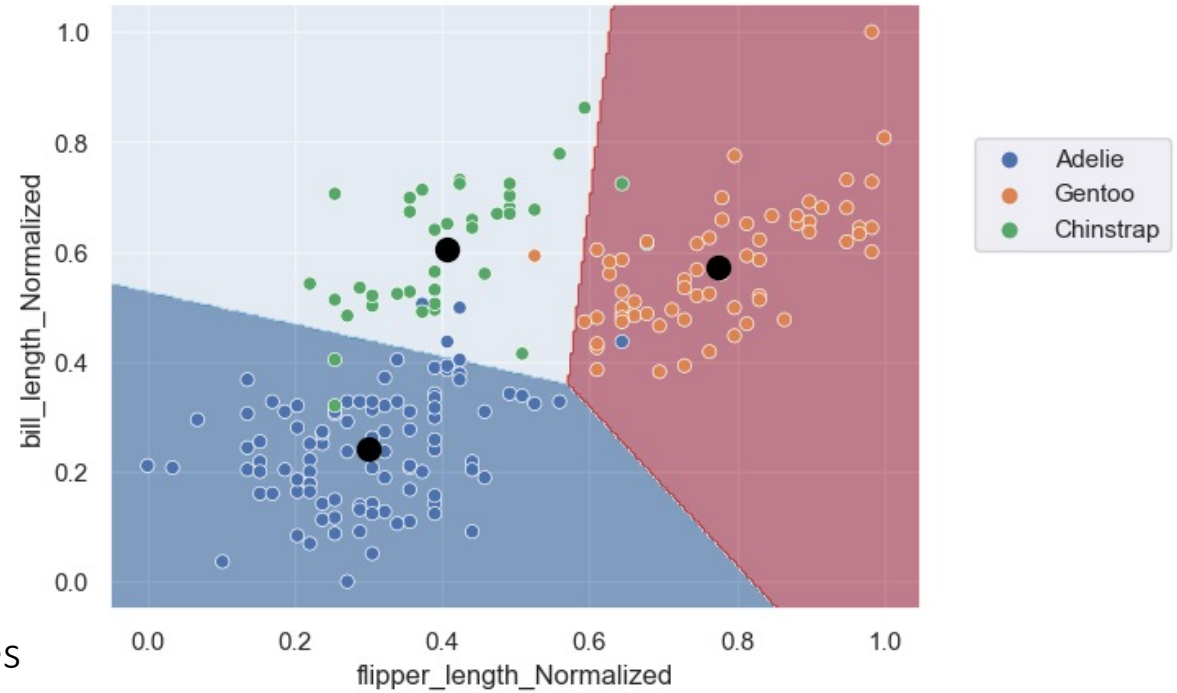
- We can scale **each feature** so that they are all on the same scale, e.g.,

$$x' = (x - \min(x)) / (\max(x) - \min(x))$$

This will scale a feature x to create a new scaled version x' where min(x') = 0 and max(x') = 1

- This is an alternative to "standard scaling" that we saw previously

Nearest-Centroid with Scaled Features



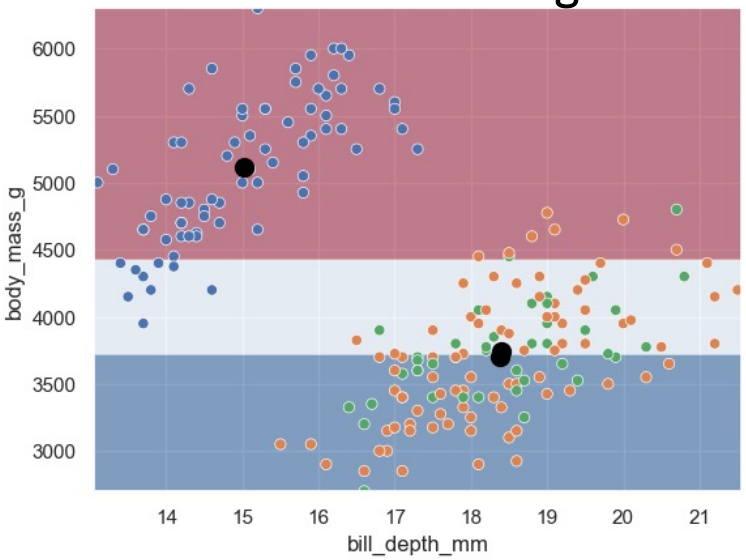
Train on Ntrain = 200 examples
Test on Ntest = 133 examples

Error on test set = 5.3% (roughly half what it was before)

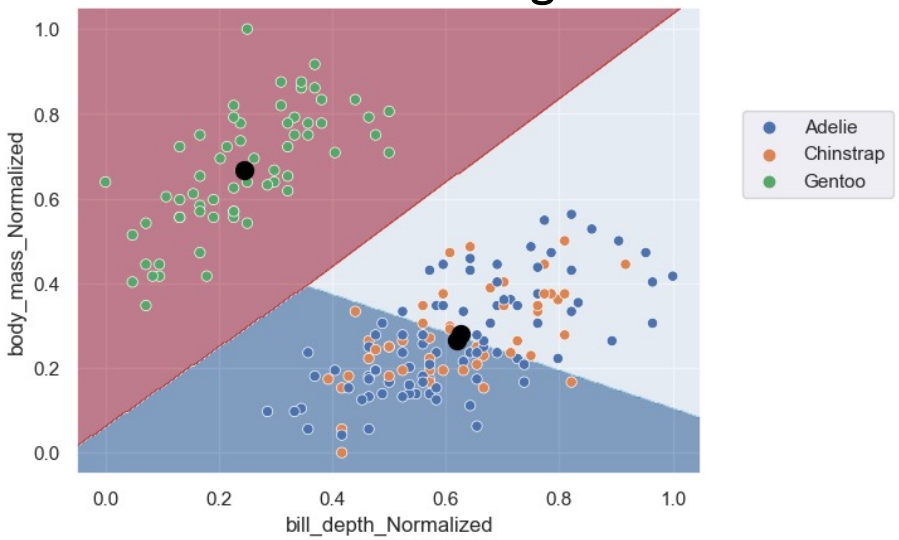
And using all 4 scaled features together? 1.5% error rate

Another Pair of Features

Before Scaling



After Scaling



Train on Ntrain = 200 examples
Test on Ntest = 133 examples

Error on test set = 36.8%

Train on Ntrain = 200 examples
Test on Ntest = 133 examples

Error on test set = 31.6%

MNIST Data

Classic dataset in ML

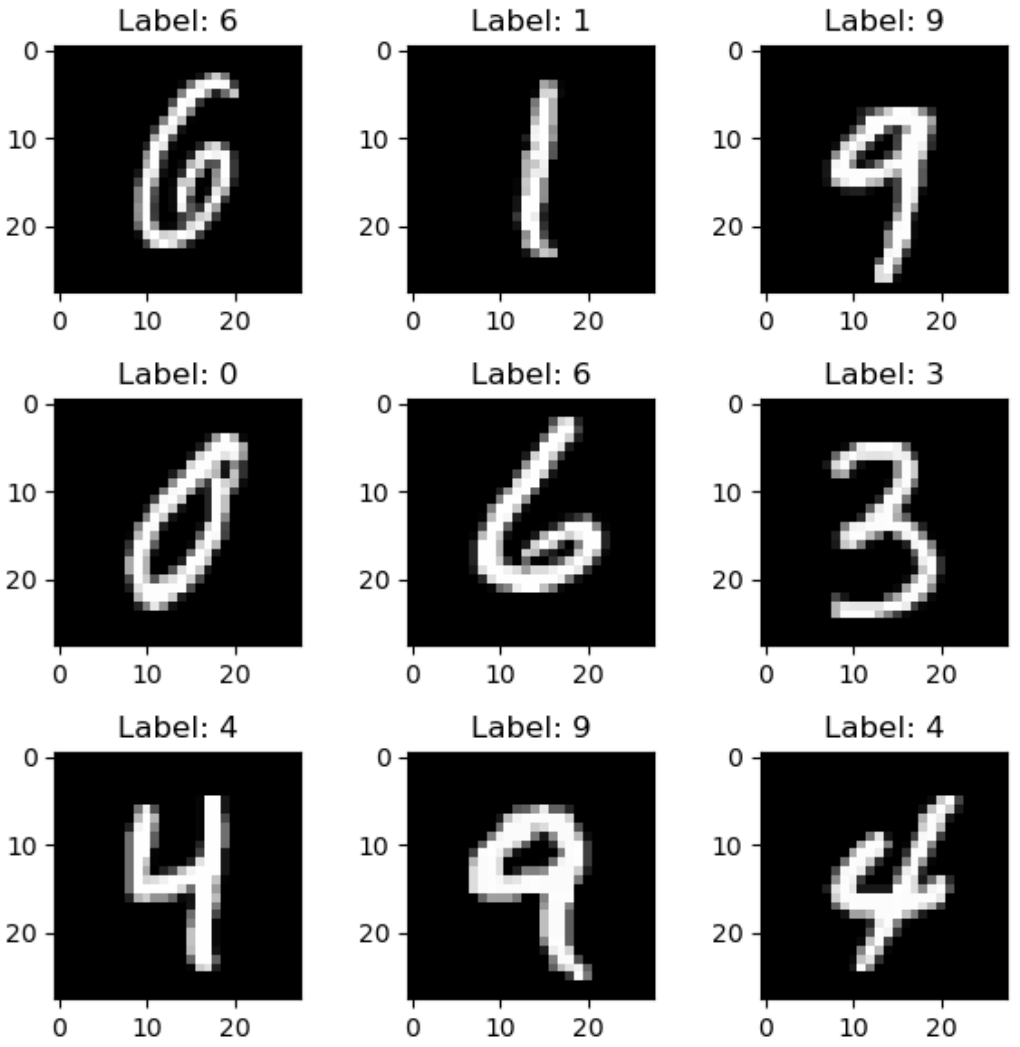
70,000 handwritten digits,
28 x 28 gray-scale pixels
Feature vector dim d = 28 x 28 = 784

10 class labels y ∈ {0, 1, 2,…. 9}

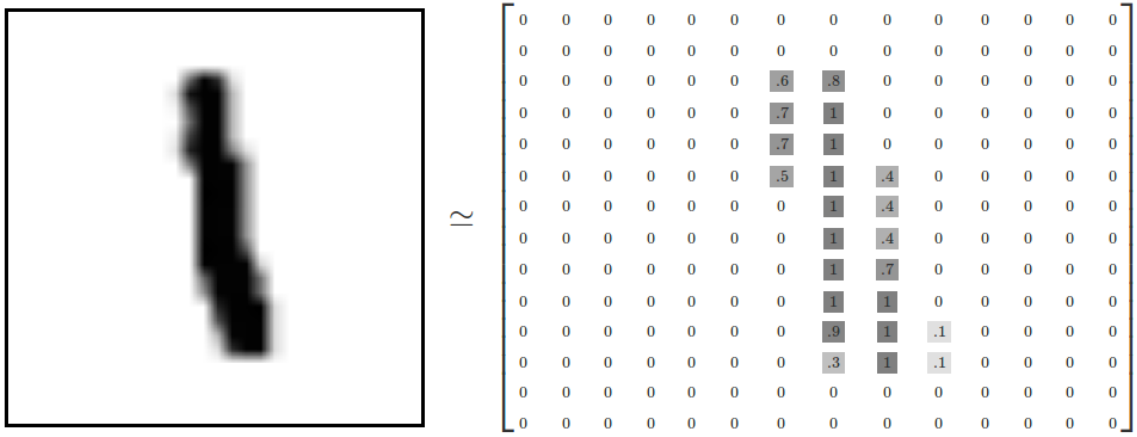
In these slides we shuffle data, and
pick 20% for train, 80% for test

Used in Homework 1 (with 75% for train)

Data originally created by
Yann LeCun (Turing Award winner)
in 1998 for ML research.
See: Le Cun, Bottou, Bengio, Haffner,
Proceedings of IEEE, 1998



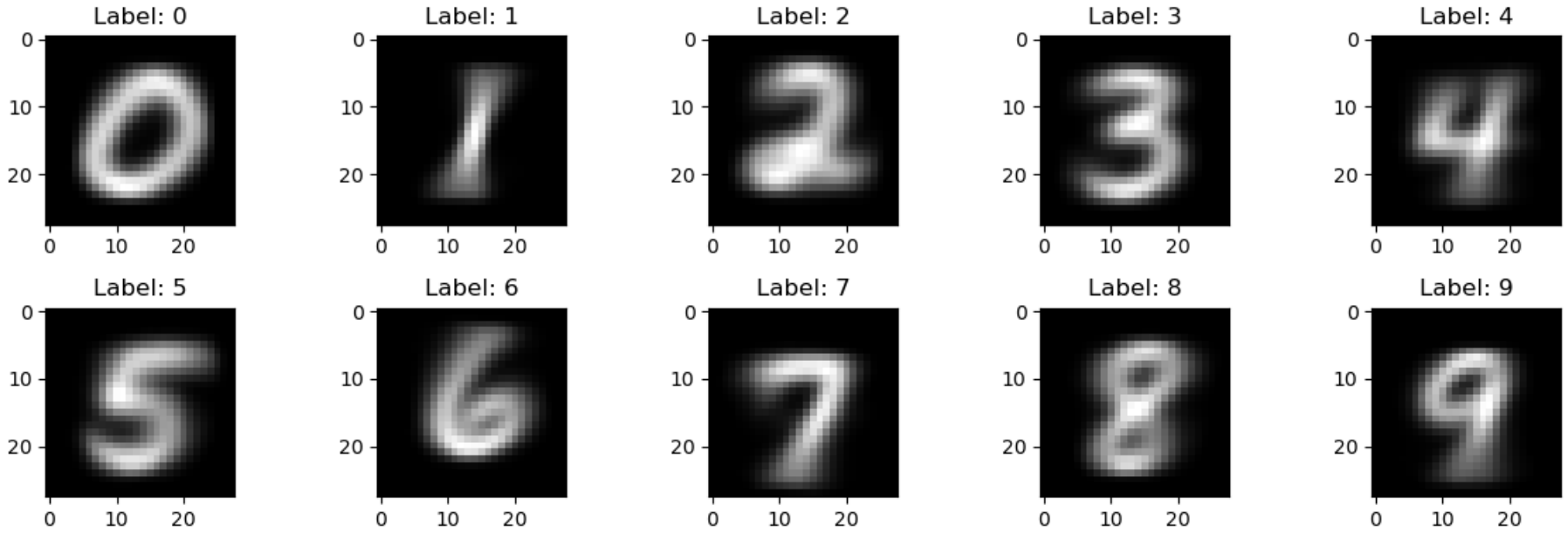
Representing Images as Feature Vectors



This m x m array of numbers is then written
out as a feature vector of length d x 1, where $d = m^2$

For example, for MNIST (homework 2):
- images are 28 x 28 (i.e., m=28)
- the resulting feature vector (per image) has $d = 28^2 = 784$

Fitted Centroids per Class

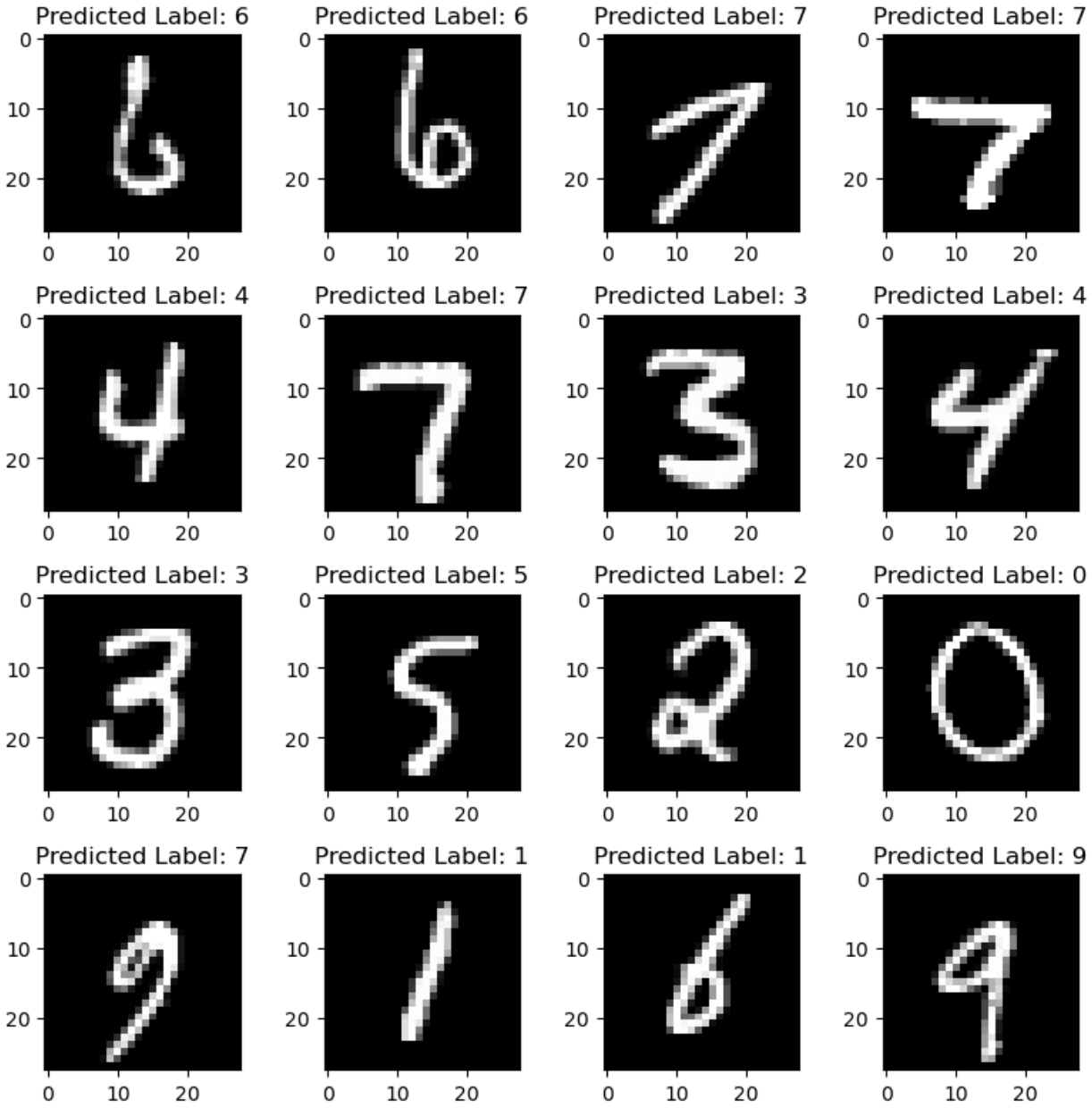


Note that the centroids are ”blurry”: why?

Predictions on Test Examples

(from NC model fit to 20% of data)

Accuracy:
About 80% on both training and test data



Questions?

Summary and Wrapup

- In classification problems we want to predict a discrete label
  - Compare with regression, where we want to predict a real-valued number
  - e.g. predicting the species of a penguin from various measurements

- Geometric concepts in ML
  - Data as points in some space
  - Norms, distances, centroids

- Nearest Centroids classifier
  - Very simple model
  - Predict the label corresponding to the nearest class centroid

Questions?
(Outside after lecture)