Today

- (Ch 11) Learning to classify
  - Nearest neighbors classifier
  - Evaluating a classifier
  - Naïve Bayes classifier

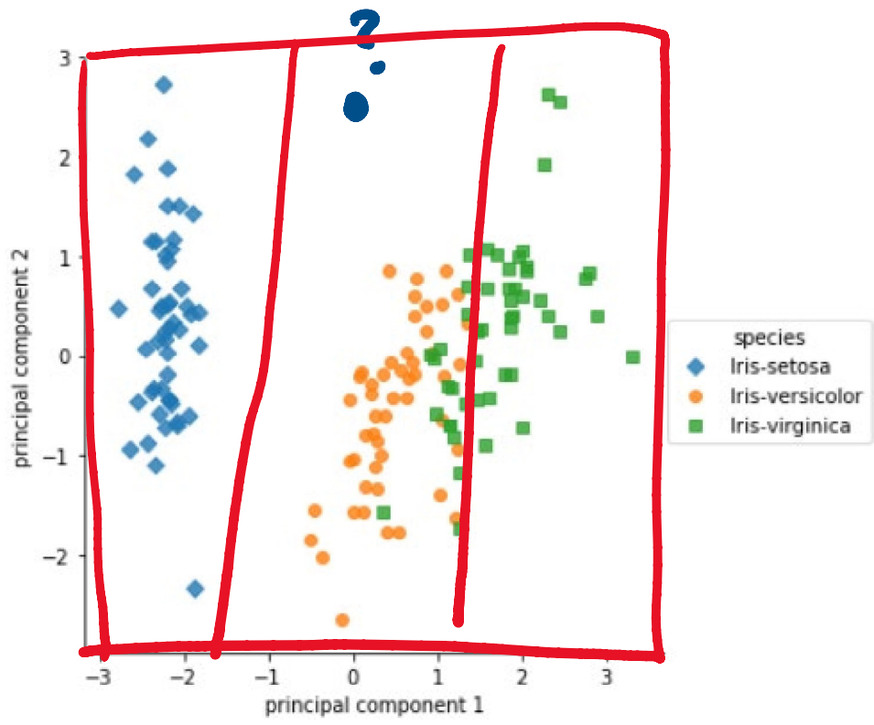
The next two lectures

- (Ch 11) Learning to classify
  - Support vector machine (SVM) classifier
  - Random forest classifier

Learning to classify

Given a set of **feature vectors** $\mathbf{x}_i$, where each has a **class label** y_i, we want to train a classifier that can map unlabeled vectors to labels

sepal length	sepal width	petal length	petal width	species
5.1	3.5	1.4	0.2	Iris-setosa
7.0	3.2	4.7	1.4	Iris-versicolor
6.3	3.3	6.0	2.5	Iris-virginica
⋮	⋮	⋮	⋮	⋮
7.8	3.0	5.1	1.3	?



Binary classifiers

- A binary classifier maps each feature vector to one of **two classes**

- For example, you can train a classifier to:
  - Look at a Twitter user's posts, time of posting, followers, etc.
    to predict whether the user is a bot
  - Look at a credit card transaction's amount, merchant, time, country, etc.
    to predict whether it is fraudulent
  - Look at a segment of DNA's sequence of bases
    to predict whether the segment is coding or non-coding
    (that is, whether the segment of DNA has a known biological function)

Multiclass classifiers

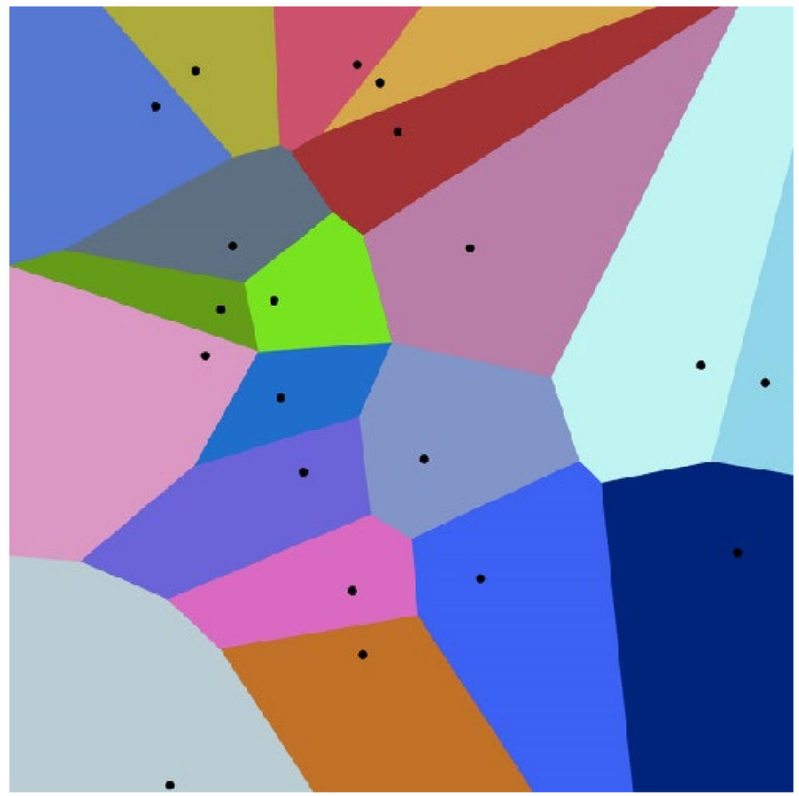
- A multiclass classifier maps to one of **three or more** classes

- For example, you can train a classifier to:
  - Look at a patient's current symptoms, lab test results, medical record, etc. to predict a diagnosis
  - Look at an image of a ZIP code to predict which ZIP it is

Nearest neighbors classifier

- Given an unlabeled feature vector $\mathbf{x}$
  - Calculate the distance from $\mathbf{x}$ to each labeled feature vector $\mathbf{x}_i$
  - Find the closest labeled $\mathbf{x}_i$
  - Assign the same label to $\mathbf{x}$

- Practical issues
  - We need a distance metric
  - We should first standardize the data
  - Classification complexity grows linearly in number of labeled feature vectors



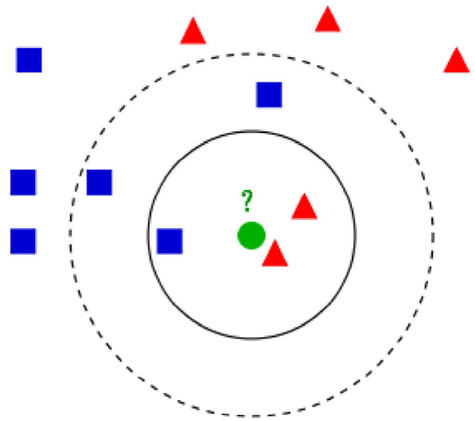
Source: Wikipedia

Variants of nearest neighbors classifier

- In k-nearest neighbors, the classifier:
  - Looks at the k nearest labeled feature vectors $\mathbf{x}_i$
  - Assigns a label to $\mathbf{x}$ based on a majority vote

$k=3 \Rightarrow \bigcirc := \triangle$

$k=5 \Rightarrow \bigcirc := \square$



Source: Wikipedia

- In (k, l)-nearest neighbors, the classifier:
  - Looks at the k nearest labeled feature vectors $\mathbf{x}_i$
  - Assigns a label to $\mathbf{x}$ if at least l of them agree on a label

How do we know if our classifier is good?

- We want the classifier to avoid making classification mistakes on unlabeled data that we will only see at run-time

- Problem 1: some mistakes may be more costly than others

  We can tabulate different types of error and define a **loss function**

- Problem 2: we will never know the true labels of the run-time data

  We must separate the labeled data into a **training set** and a **validation/test set**

Performance of a binary classifier

- A binary classifier can make two types of mistake
  - **False positive**: the classifier assigns a positive label when the truth is negative
  - **False negative**: the other way

- Sometimes one type of error is more costly
  - Pregnancy test
  - Death penalty trial

- We can tabulate the performance in a **class confusion matrix**

$$\begin{bmatrix} 15 & 3 \\ 7 & 25 \end{bmatrix}$$

		Predicted class: P	Predicted class: N
Actual Class	P	True Positives (TP) 15	False Negatives (FN) 3
	N	False Positives (FP) 7	True Negatives (TN) 25

Source: rasbt.github.io

Binary classifier with 0-1 loss function

- A **loss function** assigns costs to mistakes
- The 0-1 loss function treats FPs and FNs the same
  - It assigns loss 1 to every mistake
  - It assigns loss 0 to every correct decision

		Predicted class: P	Predicted class: N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

- Under the 0-1 loss function

$$\text{accurracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad \text{and} \quad \text{error} = 1 - \text{accuracy}$$

- The baseline accuracy is 50%, which we get by classifying randomly

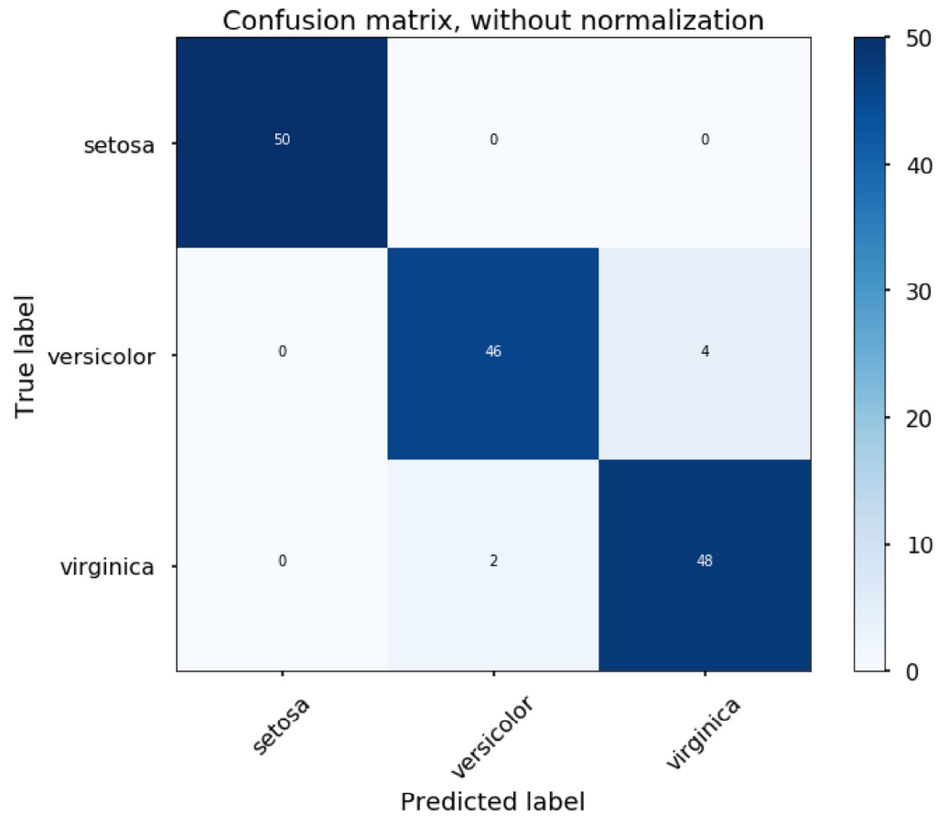
Performance of a multiclass classifier

Assuming there are c classes:

- The class confusion matrix is $c \times c$
- Under the 0–1 loss function

$$\text{accurracy} = \frac{\text{sum of diagonal terms}}{\text{sum of all terms}}$$

- The baseline accuracy is $\frac{1}{c}$



Source: qingkaikong.blogspot.com

Training set vs. validation/test set

- We expect a classifier to perform worse on run-time data than on the labeled data used for training
  - Sometimes it will perform much worse: an effect called **overfitting**
  - An extreme example: a classifier that classifies correctly if the input feature vector is in the training set, but otherwise makes a random guess
- To protect against overfitting, we separate the labeled data
  - The **training set** is for training the classifier
  - The **validation/test set** is for evaluating the performance on unused data
- It is common to reserve 10% to 20% of the data for validation

Cross-validation

- If we don't want to "waste" labeled data on validation, we can use **cross-validation** to see if our training methodology is sound
- Split the labeled data into training and validation sets in multiple ways
- For each split (called a **fold**)
  - train a classifier on the training set
  - evaluate its accuracy on the validation set
- Average the accuracy to evaluate the training methodology

Naïve Bayes classifier: a probabilistic method

- Training
  - Use the training data $\{(\mathbf{x}_i, y_i)\}$ to estimate a probability model $P(y|\mathbf{x})$
  - Assume that the features of $\{\mathbf{x}\}$ are conditionally independent given the class label y

$$P(\mathbf{x}|y) = \prod_{j=1}^{d} P\left(\mathbf{x}^{(j)}|y\right)$$

naïve

- Classification
  - Assign the label $\arg\max_y P(y|\mathbf{x})$ to feature vector $\mathbf{x}$

Naïve Bayes model

$$\arg\max_y P(y|\mathbf{x})$$

$$= \arg\max_y \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})}$$

Bayes rule

$$= \arg\max_y P(\mathbf{x}|y)P(y)$$

P(x) does not depend on y

$$= \arg\max_y \left[\prod_{j=1}^{d} P\left(\mathbf{x}^{(j)}|y\right)\right]P(y)$$

Naive assumption

$$= \arg\max_y \left[\sum_{j=1}^{d} \log P\left(\mathbf{x}^{(j)}|y\right)\right] + \log P(y)$$

The final expression avoids numerical issues related to tiny probabilities

Modeling the prior and the likelihoods

- Model the prior $P(y)$ based on the frequency of y in the training set
  - For a binary classifier, this model is a Bernoulli random variable

- Model each likelihood $P\left(\mathbf{x}^{(j)} \middle| y\right)$ by:
  - Selecting an appropriate family of distributions
    - Normal for real-valued numerical data
    - Poisson for counts in fixed intervals
    - Etc.
  - Fitting the parameters of the distribution using MLE from Chapter 9

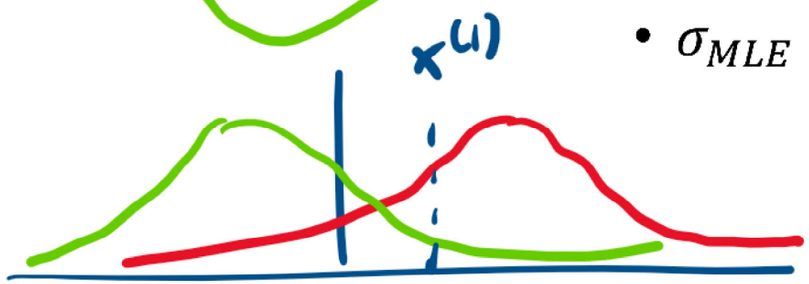
Naïve Bayes training example

Training data

$\mathbf{x}^{(1)}$	$\mathbf{x}^{(2)}$	y
3.5	10	1
1.0	8	1
0.0	10	−1
−3.0	14	−1

Modeling $P(\mathbf{x}^{(1)}|y)$ as normal

- $P(\mathbf{x}^{(1)}|y=1)$
  - $\mu_{MLE} = \frac{3.5+1.0}{2} = 2.25$
  - $\sigma_{MLE} = 1.25$
- $P(\mathbf{x}^{(1)}|y=-1)$
  - $\mu_{MLE} = -1.5$
  - $\sigma_{MLE} = 1.5$



Modeling $P(\mathbf{x}^{(2)}|y)$ as Poisson

- $P(\mathbf{x}^{(2)}|y=1)$
  - $\lambda_{MLE} = \frac{10+8}{2} = 9$
- $P(\mathbf{x}^{(2)}|y=-1)$
  - $\lambda_{MLE} = 12$

Modeling $P(y)$ as Bernoulli

- $P(y=1) = \frac{2}{4} = 0.5$
- $P(y=-1) = 0.5$