Introduction to Machine Learning

Mark Johnson

Department of Computing
Macquarie University

Suggested readings

- Chapter 6 of the NLTK book, especially the sections headed:
  - Supervised classification
  - Gender Identification
  - Choosing The Right Features
  - Document Classification

Outline

MACQUARIE UNIVERSITY

Data mining

- **Data mining** is the process of *automatically extracting information from large data sets*
- These data sets are usually *so large that manually examining them is impractical*
- The data sets can be *structured* (e.g., a database) or *unstructured* (e.g., free-form text in documents)
  - **Text data mining** uses *natural language processing* to extract information from *large text collections*
  - Quantitative data mining extracts information from numerical data
  - It's also possible to *integrate quantitative and qualitative information sources*

Business applications of data mining

- Data mining permits businesses to *exploit the information present in the large data sets* they collect in the course of their business
- Typical business applications:
  - in *medical patient management*, data mining identifies patients likely to *benefit from a new drug or therapy*
  - in *customer relationship management*, data mining identifies customers likely to be *receptive to a new advertising campaign*
  - in *financial management*, data mining can help *predict the credit-worthiness of new customers*
  - in *load capacity management*, data mining predicts the fraction of customers with airline reservations that will actually *turn up for the flight*
  - in *market basket* and *affinity analysis*, data mining identifies *pairs of products likely (or unlikely) to be bought together*, which can help design advertising campaigns

Challenges in data mining

- Diverse range of data mining tasks:
  - software packages exist for standard tasks, e.g., affinity analysis
  - but *specialised data mining applications require highly-skilled experts* to design and construct them
- Data mining is often *computationally intensive* and involve advanced algorithms and data structures
- Data mining may involve *huge data sets* too large to store on a single computer
  - often requires *large clusters* or *cloud computing* services

Machine learning

- **Machine learning** is a branch of *Artificial Intelligence* that studies *methods for automatically learning from data*
- It focuses on *generalisation* and *prediction*
  - typical goal is to *predict the properties of yet unseen cases*
  - $\Rightarrow$ split training set/test set methodology, which lets us estimate accuracy on *novel test data*
- Data mining can use machine learning, but it doesn't have to:
  - E.g., "who is the phone system's biggest user?" doesn't necessarily involve machine learning
  - E.g., "which customers are likely to increase their phone usage next year?" does involve machine learning

Statistical modelling

- **Probability theory** is the branch of mathematics concerned with *random phenomena* and *systems whose structure and/or state is only partially known*
  - ⇒ probability theory is a *mathematical foundation of machine learning*
- **Statistics** is the science of the *collection, organisation and interpretation of data*
  - A **statistic** is a function of data sets (usually numerically-valued) intended to *summarise the data* (e.g., the *average* or *mean* of a set of numbers)
- A **statistical model** is a mathematical statement of the *relationship between variables that have a random component*
  - many machine learning algorithms are based on statistical models
  - statistical models also play a central role in natural language processing

Statistics vs machine learning

- Statistics and machine learning often use *the same statistical models*
  - ⇒ very strong cross-fertilisation between fields
- Machine learning often involves data sets that are *orders of magnitude larger* than those in standard statistics problems
  - Machine learning is concerned with algorithmic and data structure issues that statistics doesn't deal with
- Statistics tends to focus on *hypothesis testing*, while machine learning focuses on *prediction*
  - **Hypothesis testing:** *Does coffee cause cancer?*
  - **Prediction:** *Which patients are likely to die of cancer?*

Outline

MACQUARIE
UNIVERSITY

Supervised classification problems

- In a *classification problem* you have to *classify* or *assign a label* y to each *data item* x
  - in *movie review classification task*, the data items are movie reviews, and the labels are *pos* or *neg*
  - in *Reuters news classification task*, the data items are news reports from Reuters, and the labels come from a set of 20 labels, such as *takeover*, *mining*, *agriculture*, etc.
  - in *name gender task*, data items are *first names* and the labels are *female* or *male*
- In order to do this, you're given *labeled training data*, i.e., a collection $D = ((x_1, y_1), \ldots, (x_n, y_n))$ of data items x_i and corresponding *label* y_i
  - each data item x_i in training data has a *label* y_i $\Rightarrow$*supervised* learning problem
  - in *movie review classification task*, training data consists of 1,000 movie reviews, each of which is rated *pos* or *neg*
  - in *Reuters news classification task*, training data consists of 10,000 news articles, each of which is labeled *takeover*, *mining*, *agriculture*, etc.
  - in *name gender task*, training data consists of 7,000 names and their genders *female* or *male*

MACQUARIE UNIVERSITY

Labeled data for name gender classification

- Goal: *to predict the gender of a first name*
- Python code to access the lists of first names

```
>>> import nltk
>>> from nltk.corpus import names
```

- The files that contain the female and male names

```
>>> names.fileids()
['female.txt', 'male.txt']
```

- The first few female and male names

```
>>> names.words('female.txt')[:10]
['Abagael', 'Abagail', 'Abbe', 'Abbey', 'Abbi', 'Abbie', 'Abby',
>>> names.words('male.txt')[:10]
['Aamir', 'Aaron', 'Abbey', 'Abbie', 'Abbot', 'Abbott', 'Abby',
```

MACQUARIE UNIVERSITY

K-nearest neighbour classifiers

- K-nearest neighbour classifiers are a simple but sometimes very effective kind of *supervised classifier* algorithm
- They don't need much maths to understand
- We'll use them to learn about general properties of machine-learning classifiers

K-nearest neighbour classifiers

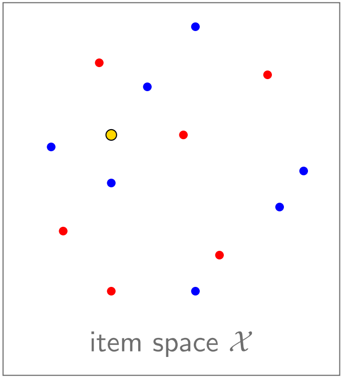
- A *k-nearest neighbour classifier* requires:
  - *labeled training data* $D = ((x_1, y_1), \ldots, (x_n, y_n))$
  - a *distance function* $d(x, x')$ that returns the "distance" between any pair of data items x and x'
  - the number k of *nearest neighbours* to use in classification

MACQUARIE UNIVERSITY

K-nearest neighbour algorithm – informal description

K-nearest neighbour algorithm – informal description

To classify a data item x:

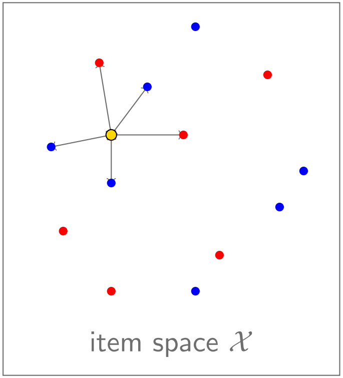


colour indicates label $\mathcal{Y}$

K-nearest neighbour algorithm – informal description

To classify a data item x:

- set N to the *k-nearest neighbours* of x in D
  - the k-nearest neighbours of x are the k training items in D with the *smallest $d(x, x')$ values*

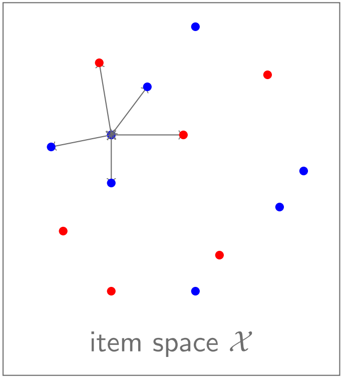


colour indicates label $\mathcal{Y}$

K-nearest neighbour algorithm – informal description

To classify a data item x:

- set N to the *k-nearest neighbours* of x in D
  - the k-nearest neighbours of x are the k training items in D with the *smallest $d(x, x')$ values*
- count how often each label y' appears in N

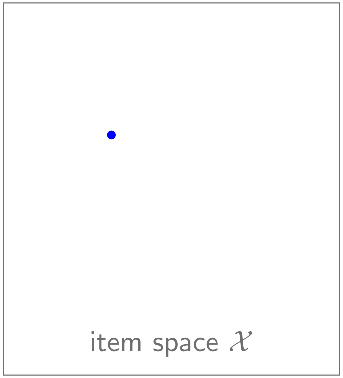


colour indicates label $\mathcal{Y}$

K-nearest neighbour algorithm – informal description

To classify a data item x:

- set N to the *k-nearest neighbours* of x in D
  - the k-nearest neighbours of x are the k training items in D with the *smallest $d(x, x')$ values*
- count how often each label y' appears in N
- return *the most frequent label* y in the k-nearest neighbours N of x as the predicted label for x



colour indicates label $\mathcal{Y}$

Evaluating classifier accuracy

- Any classifier can be viewed as a *function* f that maps a *data item* x to a *label* $\widehat{y} = f(x)$
    - we use the *hat* in $\widehat{y}$ to indicate that this is an *estimate* of y
- Evaluate classifier's performance using *labeled test data* $T = ((x_1, y_1), \ldots, (x_n, y_n))$
    - run classifier on each x_i to compute *predicted label* $\widehat{y_i} = f(x_i)$
    - *compare* the predicted labels $\widehat{y_i}$ with the *gold labels* y_i from the test data by *counting* the number m of *correctly predicted labels*

$$m \;=\; \sum_{i=1}^{n} [\![y_i = \widehat{y_i}]\!]$$

      where $[\![\text{Condition}]\!]$ is 1 if Condition is true, and 0 if Condition is false
    - return the *accuracy* of the classifier $a = m/n$
- The *accuracy* is the *fraction of the predicted labels that are correct*

The *precision* and *recall* of a classifier give a more detailed picture of a classifier's mistakes

Testing on training data over-estimates accuracy!

- What's the accuracy of a 1-nearest neighbour classifier *on the training data?*
  - assuming every data item is closer to itself than any other data item . . .

  ⇒ *perfect accuracy on training data*
- But in general *you won't get perfect accuracy on data items that aren't in the training data*
- *Evaluating a classifier on its training data over-estimates its accuracy.*
- Since we want to use our classifier to label new data items . . .

⇒ *It's essential to test on data items that aren't in the training data*

Training, development and test data

- Test data should differ from training data in order to accurately predict classifier's accuracy on novel data items
- Often classifiers have *adjustable parameters* that should be *tuned to optimise classifier's accuracy*
    - with k-nearest neighbour classifiers, select k that optimises classifier accuracy
- These parameters should be tuned on labeled data *different from the training and the test sets*

$\Rightarrow$ Tune on a *development data set* disjoint from the training and test data

- For supervised classification, divide your labelled training data into separate *training*, *development* and *test* portions

Preparing name gender data in Python

```
import collections, random, re
import nltk
from nltk.corpus import names

data = ([(name,'male') for name in names.words('male.txt')]
        +[(name,'female') for name in names.words('female.txt')])

random.seed(348)     # everyone's random shuffle will be the same
random.shuffle(data)

test = data[:500]
dev = data[500:1000]
train = data[1000:]
```

```
>>> import wk04a
>>> len(wk04a.train)
6944
>>> wk04a.train[:10]
[('Guillemette', 'female'), ('Milzie', 'female'), ('Clementina', 'fe
('Kikelia', 'female'), ('Lyssa', 'female'), ('Helise', 'female'),
('Armstrong', 'male'), ('Isobel', 'female'), ('Matteo', 'male')
```

MACQUARIE UNIVERSITY

Using features to define the distance function

- The k-nearest neighbour algorithm works with any distance function ...
- but how well it works depends on the distance function.
- It's often easy to define distance in terms of *features*
  - a *feature* is a *function* from data items x to values
  - here we'll work with string-valued features
- Examples for name gender classification:
  - the `suffix1` feature is the last letter of the name
  - the `suffix2` feature is the last two letters of the name
- Given a set of features and their values, let's define the distance between two names to be *the number of differing feature-value pairs* for the names
- There are many other reasonable ways to define distance
  - E.g., perhaps some features should be *weighted* more than others

Example: distance function for name gender classifier

- The `features` function returns the set of feature-values for a word

```
def features(word):
    return set([('suffix1', word[-1:]), ('suffix2',word[-2:])])
```

- This produces output such as:

```
>>> features('Christiana')
set([('suffix2', 'na'), ('suffix1', 'a')])
>>> features('Marissa')
set([('suffix2', 'sa'), ('suffix1', 'a')])
```

- The `suffix1` feature has the same value for both names but the `suffix2` features have different values
  $\Rightarrow d(\texttt{'Christiana'}, \texttt{'Marissa'}) = 2$

MACQUARIE UNIVERSITY

Processing the data into features

- `labeledfeatures` maps (name,label) pairs to (feature-value set,label) pairs

```
def labeledfeatures(data):
    return [(features(word),label) for (word,label) in data]
```

- Use this to prepare feature-value versions of `train`, `dev` and `test`

```
testfeatlabels = labeledfeatures(test)
devfeatlabels = labeledfeatures(dev)
trainfeatlabels = labeledfeatures(train)

>>> train[:2]
[('Guillemette', 'female'), ('Milzie', 'female')]
>>> trainfeatlabels[:2]
[(set([('suffix1', 'e'), ('suffix2', 'te')]), 'female'),
 (set([('suffix1', 'e'), ('suffix2', 'ie')]), 'female')]
```

MACQUARIE UNIVERSITY

Calculating the distance between names in Python

- Because `features` produces *sets* of feature-values, we can easily compute their *symmetric difference*
  - the *symmetric difference* between two sets are the elements that appear in one *but not in both*

```
>>> features('Marissa') ^ features('Christiana')
set([('suffix2', 'na'), ('suffix2', 'sa')])
>>> len(features('Marissa') ^ features('Christiana'))
2
```

- We can use this to *compute the distance between two feature-value sets*

```
def distance(s1, s2):
    return len(s1 ^ s2)
```

- We can use `distance` as a "distance measure" for a *k*-nearest neighbour classifier

```
>>> distance(features('Christiana'), features('Marissa'))
2
>>> distance(features('Christiana'), features('Martin'))
4
```

MACQUARIE UNIVERSITY

Building a *k*-nearest neighbour classifier in Python

- A classifier is a *function* that maps data items x to labels y

```
def make_classifier(traindata, distancefn, k=1):
    def classify(x):
        neighbours = sorted(traindata,
                            key=lambda xy: distancefn(x, xy[0]))
        return most_frequent(y for x,y in neighbours[:k])
    return classify
```

  This code constructs and *returns a function* which classifies data items.

- We can run this classifier as follows:

```
>>> import kNN
>>> classifier = kNN.make_classifier(trainfeatlabels, distance,
>>> classifier(features('Neo'))
'male'
>>> classifier(features('Adelie'))
'female'
```

Finding the most frequent value in a sequence

- We count how often each value occurs, and select the one with the highest count

```
import collections

def most_frequent(xs):
    return collections.Counter(xs).most_common(1)[0][0]
```

- The `collections` library has a `Counter` class that makes this easy

```
>>> import collections
>>> cntr = collections.Counter(['a','b','r','a'])
>>> cntr
Counter({'a': 2, 'r': 1, 'b': 1})
>>> cntr.most_common(1)
[('a', 2)]
>>> cntr.most_common(1)[0]
('a', 2)
>>> cntr.most_common(1)[0][0]
'a'
```

Evaluating classifier accuracy in Python

- Recall: the *accuracy* of a classifier is the fraction of data items that it *labels correctly*
  - evaluate classifier on heldout data if you want to estimate its accuracy on novel data

```
def accuracy(classifier, evaldata):
    ncorrect = sum(1 for x,y in evaldata
                   if classifier(x) == y)
    return ncorrect/(len(evaldata) + 1e-100)
```

- We use this to evaluate a classifier's accuracy as follows:

```
>>> accuracy(classifier, devfeatlabels)
0.788
```

Movie review data in Python

- Python code to access the reviews

```
>>> import nltk
>>> from nltk.corpus import movie_reviews
```

- The labels or categories that we want to predict

```
>>> movie_reviews.categories()
['neg', 'pos']
```

- The reviews that have each label

```
>>> movie_reviews.fileids('neg')[:2]
['neg/cv000_29416.txt', 'neg/cv001_19502.txt']
>>> movie_reviews.fileids('pos')[:2]
['pos/cv000_29590.txt', 'pos/cv001_18431.txt']
```

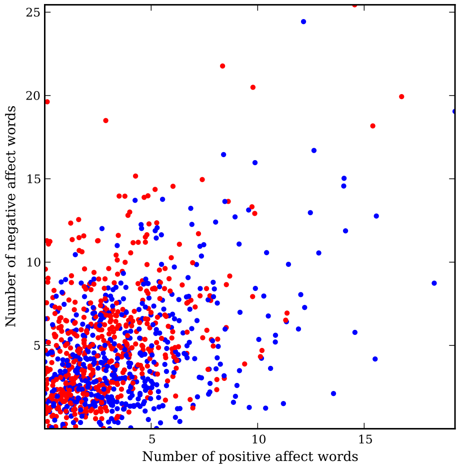
- The words in a review

```
>>> movie_reviews.words('neg/cv000_29416.txt')
['plot', ':', 'two', 'teen', 'couples', 'go', 'to', ]
>>> list(movie_reviews.words('neg/cv000_29416.txt'))[:100]
['plot', ':', 'two', 'teen', 'couples', 'go', 'to', 'a', 'church
```

MACQUARIE UNIVERSITY

Example: a simple distance function for movie review classification

- Positive affect words: *good, great, nice, liked, enjoyable, happy, best, outstanding, brilliant*
- Negative affect words: *bad,horrible,awful,hate, hated,terrible,sad,not,never*
- Red dots are `'neg'` reviews
- Blue dots are `'pos'` reviews
- A nearest neighbour classifier using just these features does terribly!



Defining distance in terms of features

- A convenient way to define a distance function is to:
  - define a vector of m *feature functions* $\boldsymbol{g} = (g_1, \ldots, g_m)$, where each g_j maps a data item $x \in \mathcal{X}$ to a *feature value*
  - use the vector of feature functions to map each x to a *vector of feature values*

$$\boldsymbol{g}(x) = (g_1(x), \ldots, g_m(x))$$

  - define the *distance function* in terms of these feature value vectors
  - If the features take *numerical values*, $d(x, x')$ can be the *sum of the squared differences* of the features

$$\begin{aligned} d(x, x') &= ||\boldsymbol{g}(x) - \boldsymbol{g}(x')||^2 \\ &= \sum_{j=1}^{m} (g_j(x) - g_j(x'))^2 \end{aligned}$$

MACQUARIE UNIVERSITY

Example: words as features in movie review classification

- Find the $m = 200$ most frequent words $\boldsymbol{w} = (w_1, \ldots, w_m)$ in the training data

```
>>> features = most_frequent_words(ml1.train, 200)
>>> features[:10]
['the', 'a', 'and', 'of', 'to', 'is', 'in', 's', 'it', 'that']
>>> features[100:110]
['how', 'people', 'then', 'over', 'me', 'my', 'never', 'bad', 'b
```

- Define $g_j(x) =$ number of times word w_j appears in review x, so $\boldsymbol{g}(x)$ is a vector of length 200
- The most frequent words are not the most information (c.f., Tf.Idf) $\Rightarrow$ might be better to *select features* somehow

Outline

k-nearest neighbour algorithms and other classification algorithms

- *Kernel-based classifiers* use a *similarity function* between training items and test items in much the way that k-NN does
  - *Kernel estimators* use a *weighted window* to place more weight on close items
- Most standard classifiers assume the data is defined in terms of *features*
- Classifiers such as *logistic regression* and *support vector machines* learn the *relative importance of each feature*
  - prior feature selection is less important $\Rightarrow$ "shotgun" feature design
  - probability theory is useful for understanding these algorithms
- k-NN is still used today because it can provide very good results with a good distance function
  - specialised data structures and algorithms for finding (approximate) nearest neighbours
- k-NN stores entire training data, which might be expensive
  - linear classifiers only store a weight for each feature, which may require less memory

Discrete versus continuous labels in machine learning

- Machine learning typically involves *learning to associate an item x with its label y*
  - If $\mathcal{X}$ is the set of possible items and $\mathcal{Y}$ is the set of possible labels, then we *learn a function $f : \mathcal{X} \mapsto \mathcal{Y}$*, i.e., that maps each $x \in \mathcal{X}$ to a $y = f(x) \in \mathcal{Y}$
- A **discrete** label set is one where the label set $\mathcal{Y}$ is a finite set
  - ▸ E.g., in a credit-rating application, $y = f(x)$ might be the rating of client x, so the label set might be $\mathcal{Y} = \{\mathrm{CreditWorthy}, \mathrm{NotCreditWorthy}\}$
- A **continuous** label set is one where the label set Y is a continuous set (usually a set of *real numbers*)
  - ▸ E.g., in a customer relationship management application where we are predicting the amount we expect to earn from various customers, $y = f(x)$ might be the amount we expect to earn from customer x, so $\mathcal{Y}$ is the set of real numbers
- We focus on *discrete label sets* here, as they have the most applications in information extraction and natural language processing

Supervised versus unsupervised training data

- Machine learning algorithms usually learn from training data.
- **Supervised training data** contains the labels y that we want to predict.
  - E.g., in *Part-of-Speech (PoS) tagging*, the training data may be a *corpus* containing words *labelled with their parts-of-speech*

Unsupervised training data does not contain the labels y that we want to predict.

  - E.g., in *topic modelling* we are given a large collection of documents without any topic labels. Our goal is to group them by topic (i.e., $\mathcal{Y}$ is a set of topics, and our goal is to learn a function f that maps each document x to its topic $y = f(x)$).

There are intermediate possibilities between supervised and unsupervised training data. **Semi-supervised training data** partially identifies the labels y, or identifies the labels on some but not all of the training examples.

  - E.g., in *PoS* tagging, we may be given a small corpus of PoS-tagged words, and a much larger corpus of words without PoS tags

Different types of machine learning algorithms

- The kinds of machine learning algorithms used *depend on whether the labels are discrete or continuous, and whether the data is supervised or unsupervised*

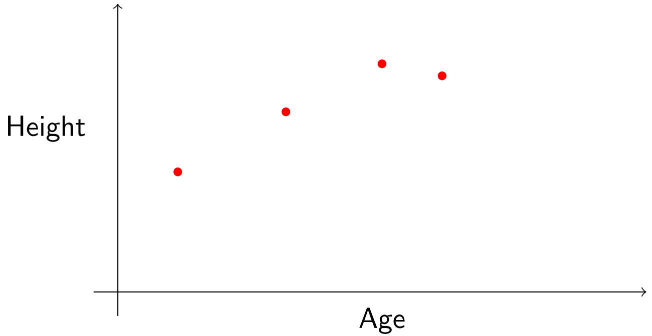
	Discrete labels	**Continuous labels**
Supervised data	*classification*	*regression*
Unsupervised data	*clustering*	*dimensionality reduction*

- We'll cover *classification* and *clustering* in this course

Outline

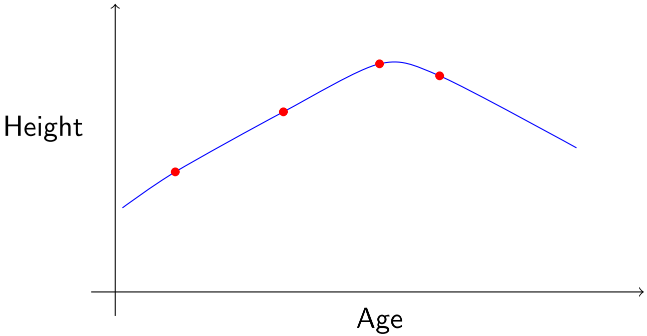
MACQUARIE UNIVERSITY

Why is machine learning hard?



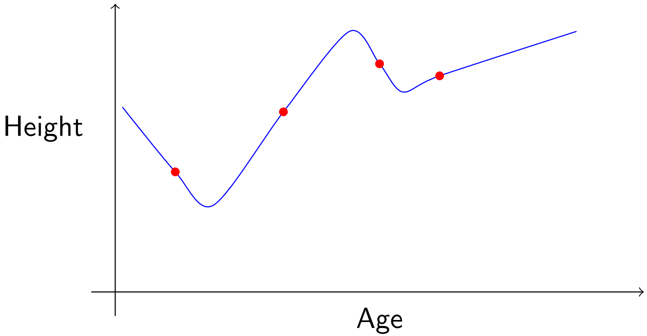
- There are an *infinite number of curves* that fit the data
  - even more if we don't require the curves to exactly fit (e.g., if we assume there's noise in our data)
- In general, *more data* would help us identify the correct curve better

Why is machine learning hard?



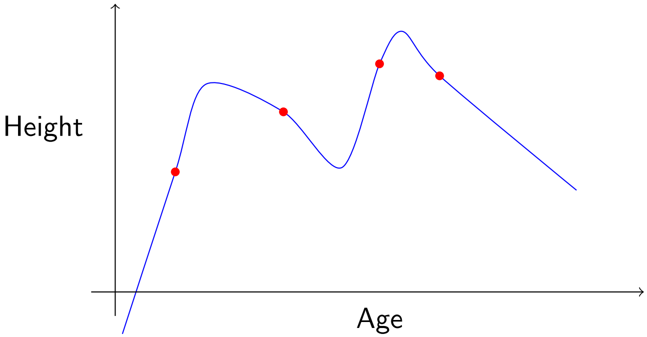
- There are an *infinite number of curves* that fit the data
  - even more if we don't require the curves to exactly fit (e.g., if we assume there's noise in our data)
- In general, *more data* would help us identify the correct curve better

Why is machine learning hard?



- There are an *infinite number of curves* that fit the data
  - even more if we don't require the curves to exactly fit (e.g., if we assume there's noise in our data)
- In general, *more data* would help us identify the correct curve better

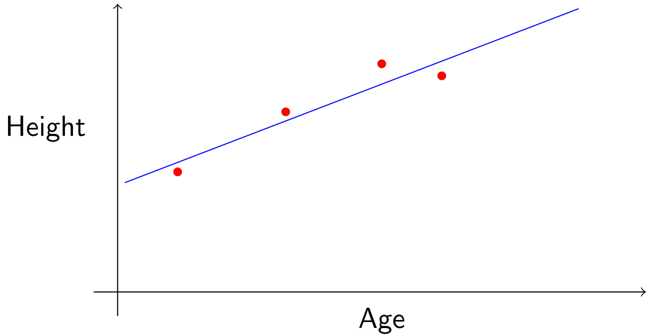
Why is machine learning hard?



- There are an *infinite number of curves* that fit the data
    - even more if we don't require the curves to exactly fit (e.g., if we assume there's noise in our data)
- In general, *more data* would help us identify the correct curve better

MACQUARIE UNIVERSITY

Why is machine learning hard?



- There are an *infinite number of curves* that fit the data
  - even more if we don't require the curves to exactly fit (e.g., if we assume there's noise in our data)
- In general, *more data* would help us identify the correct curve better

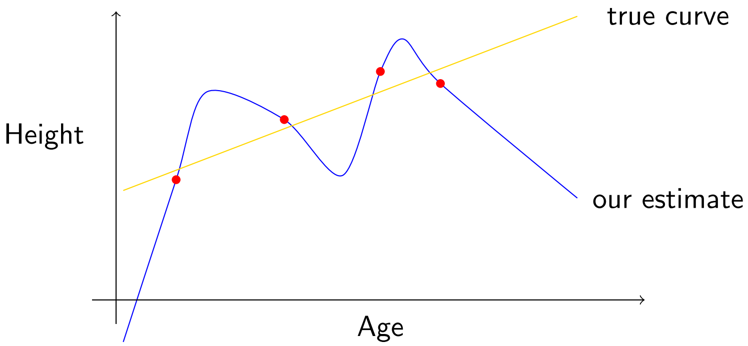
The “no free lunch theorem”

- The **“no free lunch theorem”** says there is no single best way to generalise that will be correct in all cases
  - ⇒ a machine learning algorithm that does well on some problems will do badly on others
  - ⇒ balancing the trade-off between the *fit to data* and *model complexity* is a central theme in machine learning
- Even so, in practice there are machine learning algorithms that do well on broad classes of problems
- But it’s important to understand the problem you are trying to solve as well as possible

Over-fitting and the bias-variance dilemma

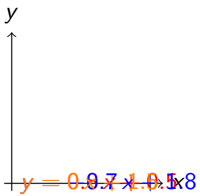
- Review the "no free lunch theorem"
  - many different functions are compatible with any finite data
  - need criteria to choose which one to use
- We'll see there are *two conflicting criteria* in choosing a generalisation
  - **Low bias:** the range of possible generalisations should be as broad as possible
  - **Low variance:** the error in the generalisation should be as low as possible
- There are techniques such as *the use of dev-test sets* and *cross-validation* that can sometimes help

Over-fitting the training data



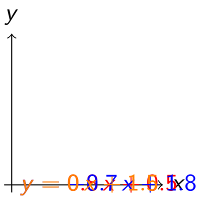
- Over-fitting occurs when an algorithm learns a function that is fitting noise in the data
- Diagnostic of over-fitting: *performance on training data is much higher than performance on dev or test data*

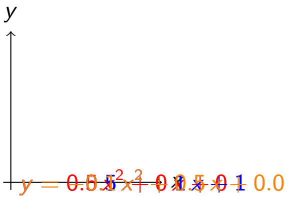
Simpler and more complex functions



- *Linear functions* have *two parameters* (b and c) and *define straight lines* $y = b\,x + c$
- *Quadratic functions* have *three parameters* (a, b and c) and *define parabolic curves* $y = a\,x^2 + b\,x + c$
- Every linear function is also a quadratic function, so *quadratic functions can describe a wider range of $x \mapsto y$ relationships than linear functions can*

Simpler and more complex functions





- *Linear functions* have *two parameters* (b and c) and *define straight lines* $y = b\,x + c$
- *Quadratic functions* have *three parameters* (a, b and c) and *define parabolic curves* $y = a\,x^2 + b\,x + c$
- Every linear function is also a quadratic function, so *quadratic functions can describe a wider range of $x \mapsto y$ relationships than linear functions can*

Bias-variance dilemma example

- Quadratic functions are *more expressive* than linear functions
  - ⇒ a quadratic function is likely to *come closer to the true function* than a linear function
- but quadratic functions have *one more parameter than linear function*
- with a fixed data set *it's not possible to learn 3 parameters as accurately as you can learn two parameters*
  - ▸ your estimates of the quadratic parameters will be *noiser* than your estimates of the linear parameters
  - ⇒ learning a quadratic function may produce *worse* performance

Bias and variance in machine learners

- The **bias** in a learning algorithm determines
  - the set of functions it fits to data
  - how it chooses a particular functions from that set
- A learner that fits linear functions has a higher bias than a learner that fits quadratic functions
- The **variance** in a learning algorithm is the degree to which noise in the training data affects the function it learns
  - a learner that learns complex functions with a large number of parameters usually has higher variance than a similiar learner that learns simple functions with a small number of parameters
- Ideally, we'd like a learner with *low bias* and *low variance*
- But in practice this isn't possible; lowering the bias raises the variance, and vice versa

Trading off bias and variance

- **Over-fitting:** Learners with low bias (and therefore high variance) can fit random fluctuations (noise) in the training data
- This often shows up as a *big difference between the accuracies on training data and on testing data*
- Many machine-learning algorithms have a parameter that trades off bias and variance
    - *in the k-nearest neighbour algorithm, the number of neighbours k used to label controls the amount of generalisation*
    - we can find the optimal value for such a parameter using *a held-out dev set* or *cross-validation on the training data*

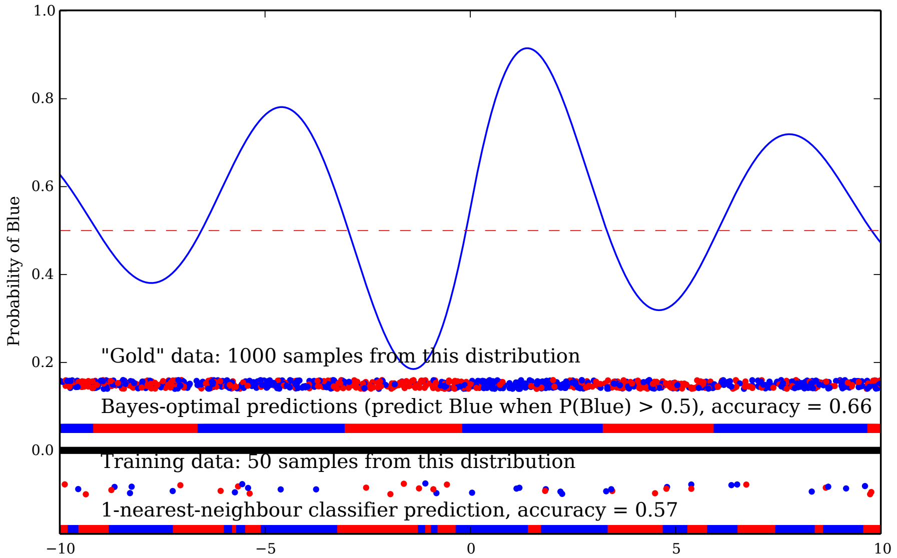
The Bayes-optimal classifier

- If the data is inherently non-deterministic (noisy) *no classifier will ever achieve perfect accuracy*
  - if we know the (true) probability of each label for the test items, the *Bayes-optimal classifier* picks the most probable label
- If we have to learn the label probabilities from data, our accuracy will in general be worse than the Bayes-optimal classifier
- Example: a biased coin, where probability of heads $\neq$ probability of tails
  - if we know the true probabilities of heads and tails, always bet on most probable outcome
  - but if we have to estimate these probabilities by observing a finite sample of throws, we may be unlucky and e.g., more heads may appear in our sample, even though tails is more probable (i.e., variance)

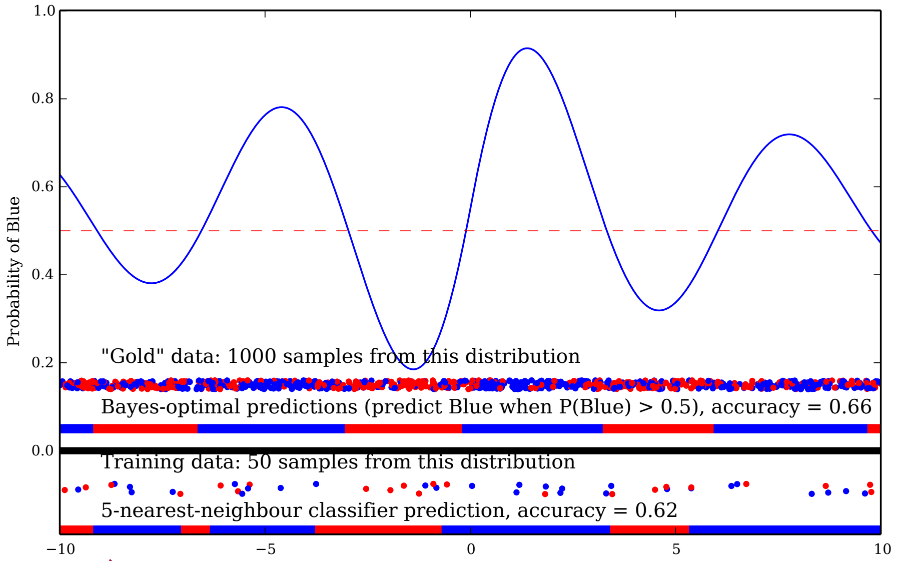
Bias and variance in k-nearest neighbour classifiers

- Two sources of error in classifiers
  - *bias:* restrictions on functions that model learns
  - *variance:* limited data $\Rightarrow$ model noise
- Different algorithms implement different trade-offs between bias and variance
- In a k-nearest neighbour classifier:
  - smaller values of k decrease bias and increase variance
  - larger values of k decrease variance and increase bias

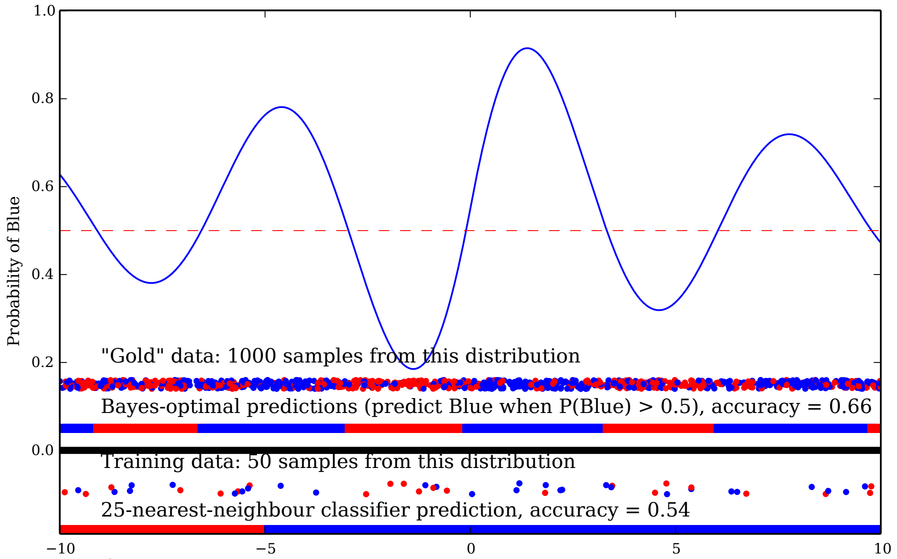
K-nearest neighbours; $N = 50, K = 1$



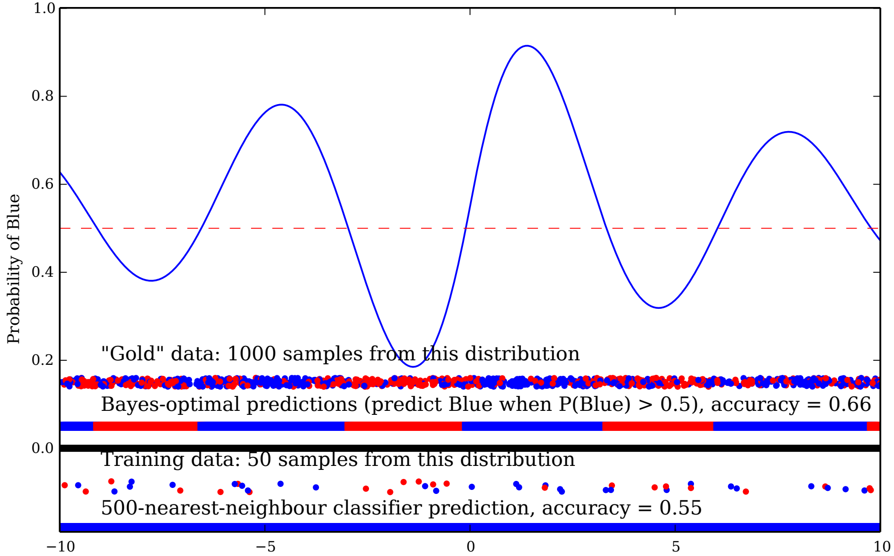
K-nearest neighbours; $N = 50, K = 5$



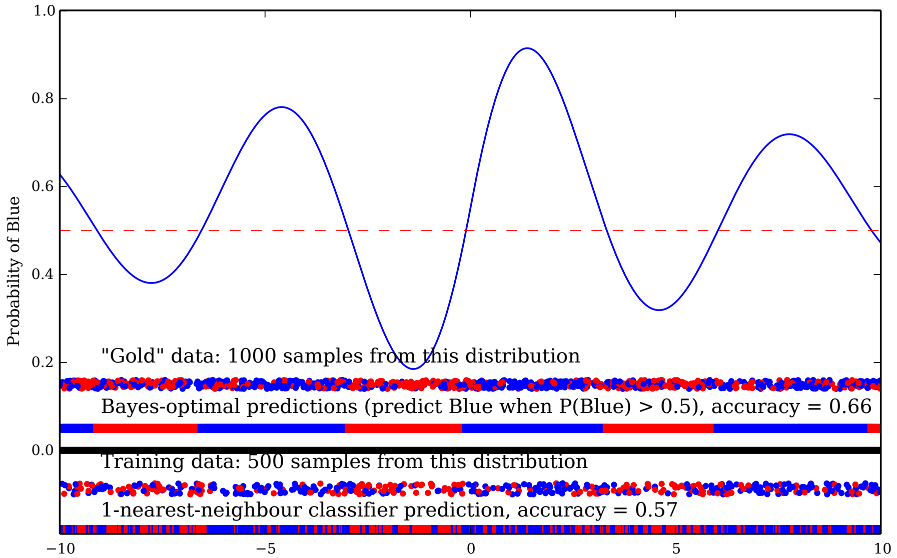
K-nearest neighbours; $N = 50, K = 25$



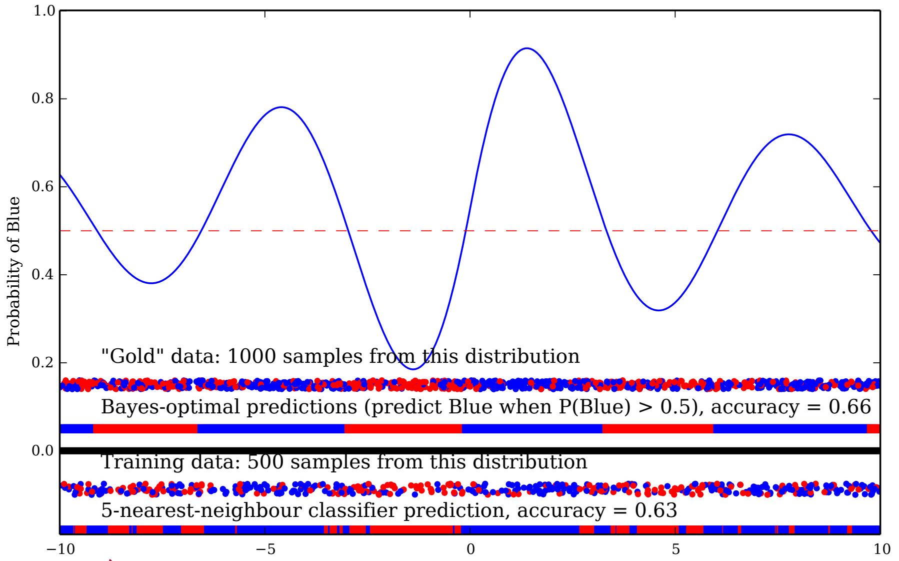
K-nearest neighbours; $N = 50, K = 50$



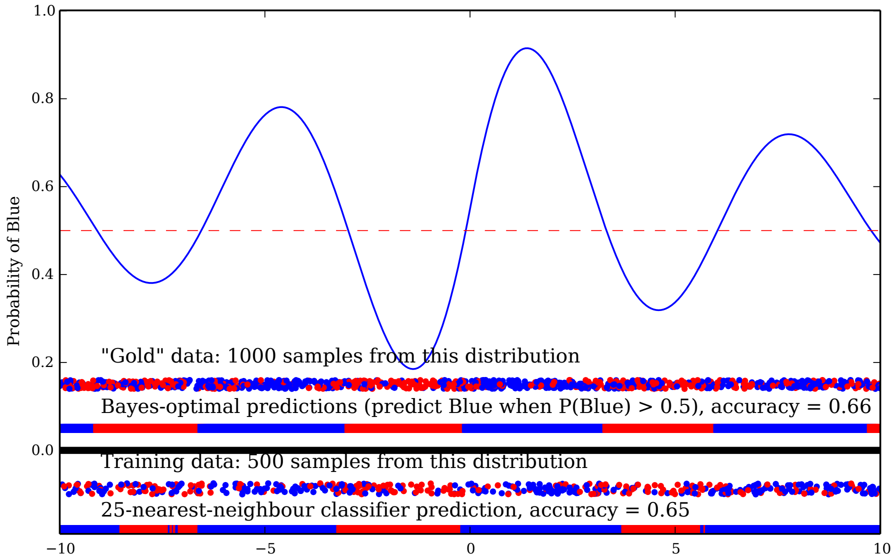
K-nearest neighbours; $N = 500, K = 1$



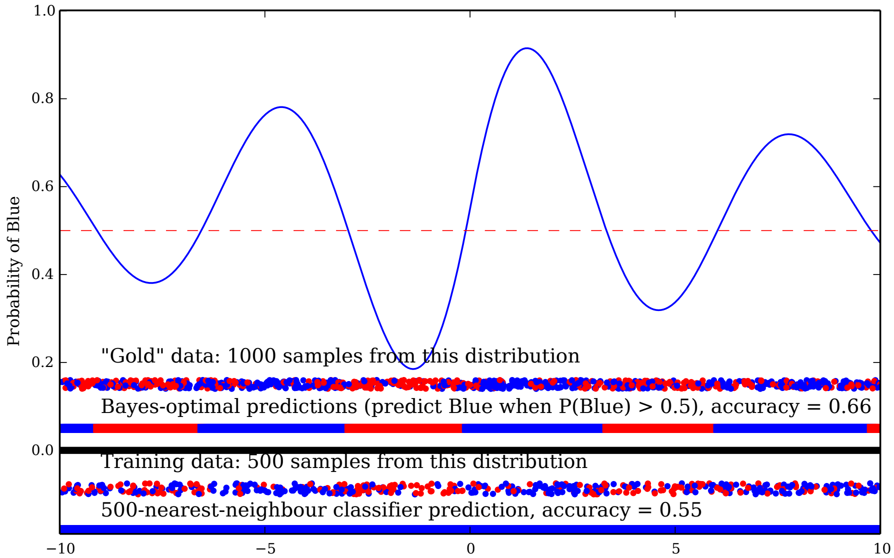
K-nearest neighbours; $N = 500$, $K = 5$



K-nearest neighbours; $N = 500$, $K = 25$



K-nearest neighbours; $N = 500$, $K = 500$



Outline

MACQUARIE
UNIVERSITY

Dimensions of machine learning

- *Supervised versus unsupervised machine learning*: are we given examples of the output we have to produce?
  - unsupervised machine learning is a kind of *clustering*
- *Discrete versus continuous outputs*:
  - *classification*: supervised learning with discrete outputs
  - *regression*: supervised learning with continuous outputs
  - *clustering*: unsupervised learning with discrete outputs
  - *dimensionality reduction*: unsupervised learning with continuous outputs

MACQUARIE UNIVERSITY

Fundamental limitations on machine learning

- *"No Free Lunch" Theorem*: many different hypotheses (functions) are compatible with any data set
- *Bias-Variance dilemma*: it's impossible to simultaneously minimise both bias and variance
  - the *bias* in a learner restricts the class of hypotheses it can form
  - the *variance* in a learner is the "noise" in its estimates
- This often manifests itself in *over-learning*

  $\Rightarrow$ important to separate *test data* from *training data*

K-nearest neighbour classifier

- *K-nearest neighbour classifier:* Label a test data item with the most frequent label of its k nearest neighbours in the training data
- The number of neighbours k controls the bias-variance trade-off
- The performance of a k-nearest neighbour classifier depends on how the *distance function* is defined
  - distance can be defined using *features* extracted from the data items