class: middle

# Data Science for Developers (Intermediate)

@DrPhilWinder

DrPhilWinder

http://WinderResearch.com

http://TrainingDataScience.com

---

## Schedule

- 09:00 Start
  - Model evaluation
- 10:15 Morning break
  - Evidence and Probabilities
- 12:00-13:00 Lunch
  - Dimensionality Reduction
- 14:15 Afternoon break
  - A Tour of SKLearn
- 16:30-17:00 End

---

## Information

Name: Phil Winder

Title: CEO Winder Research

Occupation: Engineer (cloud software and data science)

This training: First of three parts

Other training: visit https://WinderResearch.com/training

Online Training: visit https://TrainingDataScience.com

Tweet! @DrPhilWinder

Email: phil@WinderResearch.com

.center[Winder Research]

# Content and Questions

- I will email slides and workshops. To do this I need your email addresses.
- Please interrupt to ask questions. Data Science is all about asking the right questions.
- We're working on a cloud environment. You don't need to install anything (unless you want to).

# Setting Expectations

- This *is* difficult
- There *is* a lot of content here
- We *will* move fast (this is intentional)
- Data Science is *chaotic*; learning is not as easy as A.B.C.

The Goal:

1. Overview. Understand what you don't know.
2. Go away and continue learning.

*The goal is not to become an expert in 1/2/3 days*

# Models we will Learn Today

- Too many to mention

# Concepts we will Learn Today

- Model evaluation, numerical and visual
- How to think probabilistically, Bayesian statistics and Naive Bayes
- How to handle changes in data
- Dimensionality reduction
- A massive dose of classification and clustering algorithms.

---

1. Model Evaluation
   a. Numerical Model Evaluation
   b. Visual model evaluation
2. Evidence and probabilities
3. Changing Landscapes
4. Dimensionality Reduction
   a. Principal Component Analysis
   b. Self Organising Maps
   c. Manifold Learning
5. A Tour of SKLearn algorithms
   a. Classification
   b. Regression
   c. Clustering
6. Grand Challenge

---

name: what-makes-a-good-model

# What makes a good model?

These chapters introduces model evaluation.

- What makes a good model?
- How do we measure what a good model is?
- Why is it important?

???

Discussing evaluation metrics is made difficult because each domain has it's own set of metrics. It is impossible to suggest that one single metric is correct for all cases (no free lunch, again) because each domain has different requirements.

Instead we cover a range of topics that aim to highlight the most common forms of evaluation.

---

name: model-evaluation

# Model Evaluation

- If you were given three models, how would you choose which is best?

Key point: The best model is that which maximises your measure of performance

???

Imagine we had the task of evaluating which of three chosen models performed best. Ultimately, the best model is the one that maximises your measure of performance. But how do we define performance?

---

# Big Picture Goals

- Establish exactly what it is the business is trying to achieve

*People are quick to forget that they are being paid to achieve a goal.*

- The more concrete, the better.

Some examples include:

- Improve retention rate of customers
- Detect fraud with a Nuisance Alarm Rate of 1 per 1000 customers
- Classify which users will buy a product with 95% accuracy
- Improve sales by 5%

???

When knee deep in techniques and technologies, it is often easy to forget that we are being paid to achieve some business goal. It is important for both engineers and stakeholders to establish exactly what it is they are trying to achieve. You'd be surprised how often this is ignored.

So the first task is to establish what is needed. The more precise, the better. The requirement will direct the choice of evaluation statistic, sometimes even define it.

---

# Reporting Difficulties

- Businesses should not enforce the use of a summary statistic

*Businesses are often too far from the implementation details.*

- But be *very careful* about how your technical metric relates to the business metric.

*It is very easy to optimise for the wrong thing.*

???

However, businesses must not enforce the use of a statistic. Often the businesses definition will be too far removed from the implementation.

For example, a business might want to improve customer acquisition by improving a product. While developing models, we could measure the impact of developments by running versions of an algorithm and measuring the numbers of new customers.

But this would lead to a long development cycle, so in the interim, it might be better to chose a more technical measure of success.

But be warned, if you do chose a replacement metric, be sure to think very carefully about how it relates to

the business's definition. You might end up spending over-optimising athat doesn't improve the business.

---

# Evaluating Classifiers

To recap, a classification model takes an instance and attempts to predict its class.

For binary problems, the classes are "positive" or "negative". How should we measure the performance of a binary classifier?

First let's take a step back and examine what we mean by classes, performance and error.

---

## Positive and Negative

Positive and negative have different meanings in data science.

- A positive class means that an instance has some characteristic that we want to detect. We want to be alerted to the presence of a positive.
- A negative class means it is not something that is of interest to us. There is no cause for alarm.

*A positive cancer detection does not mean it is a good thing.*

???

For example, when we say that a cancer test has "come back positive", we don't mean that the result in itself is a positive thing. Just that the instance had some indications characteristic of cancer.

---

# Value and Performance

- Decisions usually have monetary value.

*Businesses run on cash. If you can use monetary values, use them.*

- Given that decisions have value, we can minimise the cost/maximise the value.

*We need to ask the business to place monetary objectives on decisions.*

???

We can usually attribute some value to each type of detection. For example, being able to accurately predict whether an account will be fraudulent in the future is a very valuable tool to have. So a positive prediction has a high value, but a negative prediction has little value.

Given that decisions have value, performance can be defined as how well our models maximise that value (or minimise the cost, to look at it another way).

This leads back to the original definition of the business requirement, but may require some translation. For example, if the business requirement was to prevent fraud, then we need to know how much fraudulent and non-fraudulent activities cost.

# Error

- Errors occur when systems do something undesirable.

*These are called false positives and false negatives*

- E.g. You get questioned at the airport for some unknown crime.

???

In data science, an undesirable event is when a model predicts positive, when in reality it is a negative, or it predicts negative when it was really a positive.

As you can imagine, the opposite, correct predictions, are called true positives and true negatives.

A false positive would be like when you got stopped at the airport for some supposed crime. A false negative could occur if a drugs smuggler walked through the airport without being stopped.

---

## Cost of Error

- Decisions have value. Errors have costs.

For example:

- A false negative on a medical test could cost a life.
- A false negative resulting in not showing an advert has some opportunity cost.

???

Since all decisions have value, there are costs involved when a model makes an incorrect prediction. The severity of the incorrect prediction differs by industry.

For example, the cost of a false negative on a medical test could be life threatening. However the cost of serving the wrong advert only consists of the lost opportunity cost.

The value of the prediction and the cost of the error must both be taken into account.

---

## Accuracy

Up to now, we have been using accuracy to measure all of our results.

Accuracy is defined as:

$$accuracy = \frac{\text{Number of correct decisions made}}{\text{Total number of decisions made}}$$

The accuracy is the number of occasions where the model made the correct decision.

This is the most common performance metric because it is easy to measure and easy to understand.

Despite its simplicity it has a range of drawbacks. To describe these it is easiest to use a confusion matrix.

---

## Confusion matrix

A confusion matrix is a table showing the degree of class confusion.

| | actual positive | actual negative |
|---|---|---|
| predict true | true positive | false positive |
| predict false | false negative | true negative |

It is constructed as an $n \times n$ matrix with columns labelled as actual classes and rows labelled with predicted classes. For a binary problem the table is of size $2 \times 2$.

The matrix separates the decisions made by the model, making it clear how one class is being confused for another.

---

## Unbalanced classes

- Very common to have classes that are very rare.

Skews of $1:100$ are common in fraud detection, and skews of greater than $1:10^6$ have been reported in other applications[1].

.bottom[ Attenberg, J., & Provost, F. (2010, July). Why label when you can search?: alternatives to active learning for applying human resources to build classification models under extreme class imbalance. ]

???

This is another important example of having to think very carefully about model performance.

It is very common to have one class that is very rare. It is uncommon to see equally balanced classes in most applications. For example, fraud occurs rarely. You will have very few examples of fraud in your sample.

Skews of $1:100$ are common in fraud detection, and skews of greater than $1:10^6$ have been reported in other applications[1].

Unbalanced classes are a serious problem for accuracy. Imagine a skew of $1:100$. If I create a model that predicts negative for every class, I will produce an accuracy of 99%!

This is known as the *base rate*. The base rate is very useful to bear in mind; use it to sanity check your models!

Furthermore, imagine two teams were creating competing models. If they used different data with a different skew, you wouldn't be able to compare the results because the base rates are different.

.bottom[

1. Attenberg, J., & Provost, F. (2010, July). Why label when you can search?: alternatives to active learning for applying human resources to build classification models under extreme class imbalance.

]

---

In this example there are 50 instances in our sample. 1 of them is an actual yes, the rest are no.

We could create a model that predicts negative for all samples. The result would look like:

| | positive | negative |
|---|---|---|
| yes | 0 | 0 |
| no | 1 | 49 |

$$accuracy = \frac{\text{Number of correct decisions made}}{\text{Total number of decisions made}} = \frac{49}{50} = 98\%$$

If we keep the same model, and now receive a sample that is balanced, the result would look like:

| | positive | negative |
|---|---|---|
| yes | 0 | 0 |
| no | 25 | 25 |

$$accuracy = \frac{25}{25} = 50\%$$

## Unequal cost/benefits

- Sometimes certain outcomes are more costly than others.

For example:

- The cost of missing cancer is high.

Key point: Ideally, we want to maximise the *expected profit*.

???

Another fundamental problem with accuracy as a metric is that it can misrepresent the cost/benefit of making a decision.

Again, let's imagine we had a sample with a $1:100$ skew and our model predicted negative for all results. Now imagine that this was a classifier for cancer detection.

The cost of making a false positive is worrying, but ultimately of little harm. The cost of making a false negative could be life threatening.

This is why there are many false positives with cancer diagnoses.

Ideally we should estimate the cost/benefit for each decisions a classifier can make, then we can aggregate to obtain a total *expected profit*.

---

name: numerical-model-evaluation

# Numerical Model Evaluation

- Every decision you make has some affect on the business.

*We need to talk in the same language as the business. Profit/Loss.*

???

After encountering the accuracy metric and see all it's flaws, you're probably wondering whether there is a better option.

We mentioned that optimising the cost/benefit of a result is better than optimising for accuracy. But why?

In nearly every context, the definition of a successful business is one that earns enough money to keep it afloat. Since your data science products are to be used to improve the business, it makes sense to optimise development by maximising the positive impact on the business.

In short, every decision your model makes has some affect on the business. You need to quantify what those costs/benefits are, then we can use the *expected value* method as an optimiser.

---

# Expected value

The expected value is the weighted average of the different possible outcomes, where the weight is the probability of occurrence.

For example, if you bet on the value of a standard die, where the probability of each value is $1/6$, then you have a $1/6$ chance of winning the prize.

Imagine that if you lost, you had to pay *10p*, but if you won, you won *£1*.

If we bet on rolling a 1, the expected value of this would be:

$$
\begin{aligned}
\text{expected value} &= p(o_0) \cdot v(o_0) + p(o_1) \cdot v(o_1) + \dots && (1)\\
&= \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 0.1 + \dots && (2)\\
&= 0.25 && (3)
\end{aligned}
$$

---

### Where do the values come from?

- Probabilities: can usually be estimated from the data. Sometimes there are well known laws and distributions.
- Values: must be acquired from other sources, usually the business or subject matter experts.

---

## Example

- Problem: My business has a new marketing product, but it's quite expensive. I want to send this marketing to people who are likely to then buy my product. I don't want to send it to people that aren't going to buy my product.
- Structure of the problem:
  - Cost of marketing: £9.
  - Profit from sale: £50.
  - Database of leads with some demographic data: $\mathbf{x}$.
  - Probability of someone buying my product is provided from a previous randomised experiment: $p(\cdot)$.
- Question: Are we going to make any profit at all? Who should we offer this product to?

---

- Two outcomes: buying and not buying.

Value of outcome 1: profit of sale - cost of marketing

Value of outcome 2: - cost of marketing.

$$
\begin{aligned}
\text{expected value} &= p(o_0) \cdot v(o_0) + p(o_1) \cdot v(o_1) + \dots && (5)\\
&= p_b(\mathbf{x}) \cdot (£50 - £9) + (1 - p_b(\mathbf{x})) \cdot (-£9) && (6)\\
&= && (7)
\end{aligned}
$$

???

To answer this question, we can use the expected value method. Recall that the expected value is the weighted sum of all possible outcomes.

The expected value if someone buys is the profit of the sale minus the cost to send the marketing materials.

The expected value if someone doesn't buy is simply the cost of sending the marketing materials.

Hence, the expected value equation looks like:

---

The first question to answer is whether or not we are going to profit. We can ask mathematically, is this equation greater than zero?

$$
\begin{aligned}
& && (8)\\
p_b(\mathbf{x}) \cdot (£50 - £9) + (1 - p_b(\mathbf{x})) \cdot (-£9) &> 0 && (9)\\
p_b(\mathbf{x}) \cdot (£50 - £9) - (1 - p_b(\mathbf{x})) \cdot £9 &> 0 && (10)
\end{aligned}
$$

We can rearrange that equation to provide a decision rule:

$$
\begin{aligned}
& (11) \\
p_b(\mathrm{x}) \cdot (£50 - £9) - £9 - p_b(\mathrm{x})) \cdot £9 > 0 & (12) \\
p_b(\mathrm{x}) \cdot (£50 - £9) - p_b(\mathrm{x})) \cdot £9 > £9 & (13) \\
p_b(\mathrm{x}) \cdot £41 - p_b(\mathrm{x})) \cdot £9 > £9 & (14) \\
p_b(\mathrm{x}) \cdot £32 > £9 & (15) \\
p_b(\mathrm{x}) \cdot £32 > £9 & (16) \\
p_b(\mathrm{x}) > \frac{£9}{£32} & (17) \\
p_b(\mathrm{x}) > 0.28 & (18)
\end{aligned}
$$

In other words, we will profit if we send out the marketing materials to people that have higher than a 28% chance of responding.

---

# Expected Value Method for Model Evaluation

We've just seen an example of how the expected value method can provide a rule that tells us when to use a model, but how can we use it to compare models?

We can extend the expected value method to measure the performance of a classifier with the help of the confusion matrix.

---

## Expected Outcomes of a Model

All the information we need to generate the probability of individual outcomes are provided in the confusion matrix. For example, consider the following confusion matrix:

| | p | n |
|---|---|---|
| Y | 12 | 3 |
| N | 5 | 30 |

Recall that the columns represent the model's prediction and the rows represent the actual class.

---

Now we need to calculate the probabilities of all the various outcomes.

Total number of samples: $50$.

$$p(Y, \mathrm{p}) = \frac{12}{50} = 0.24 \tag{20}$$

$$p(Y, \mathrm{n}) = \frac{3}{50} = 0.06 \tag{21}$$

$$p(N, \mathrm{p}) = \frac{5}{50} = 0.1 \tag{22}$$

$$p(Y, \mathrm{n}) = \frac{30}{50} = 0.6 \tag{23}$$

Next we need the values for each outcome.

## Costs/Benefits

Unlike the class probabilities (which are estimated from the labelled data), we can't estimate the costs or benefits. These must come from the business or a domain expert. Ideally they would be thoughtfully crafted, but more often than not, estimates are good enough.

Let's look at our marketing example again.

- A true positive is a customer that was offered the product and bought it. The benefit is then the profit from the product, £50, minus the cost it took to send the materials, £9, £41.
- A true negative is a customer that was not offered the product and was therefore unable to buy the product.
- A false positive is someone who was given the marketing materials but did not buy the product. Hence the cost of such an error is £9.
- A false negative is where a customer wasn't given the marketing materials so they could not buy the product, but would have bought the product if it was sent. In this case no money was spent, but nothing was gained. The total cost is £0. (Although there is some opportunity cost here, we could have made some money but we didn't.)

We can represent this cost benefit data, $b(\mathrm{x})$, in a matrix:

| | p | n |
|---|---|---|
| Y | 41 | -9 |
| N | 0 | 0 |

Finally we can compute the expected value in the normal way:

$$\text{expected value} = p(\mathrm{x}) \cdot b(\mathrm{x})$$

And we would find that the expected value is:

$$\text{expected value} = 0.24 \times 41 - 0.06 \times 9 = £9.3$$

- Some false positives which affect the expected value.

*So we could tune the algorithm to accept more false negatives in order to minimise the false positives*

???

In other words this classifier has some false positives which negatively impacts the expected value. There are also false negatives but this does not affect the expected value.

You can see how this affects how we might chose to tune the algorithm. We might tune it to accept more false negatives in order to minimise the false positives.

Furthermore, you can also see how we would compare two models. The one with the highest expected value wins.

---

However, there is still the issue of imbalance. Let's consider two models. When using balanced data, both models have the same accuracy (80%), although the results are subtly different.

| Model A | p | n |
|---|---|---|
| Y | 50 | 20 |
| N | 0 | 30 |

| Model B | p | n |
|---|---|---|
| Y | 30 | 0 |
| N | 20 | 50 |

---

If we now imagine that Model A was trained on an unbalanced sample, say 25% positives, 75% negatives, then the confusion matrices become:

| Model A | p | n |
|---|---|---|
| Y | 25 | 30 |
| N | 0 | 45 |

| Model B | p | n |
|---|---|---|
| Y | 30 | 0 |
| N | 20 | 50 |

| Model B | p | n |
|---|---|---|

If we calculate the standard expected value calculation, then the results are:

$$\text{Model A} = 0.25 \times 41 - 0.3 \times 9 = £7.55$$

$$\text{Model B} = 0.3 \times 41 - 0 \times 9 = £12.3$$

So model B appears to be better than model A. However, all is not as it may seem...

# Towards Bayes

So what we need to do is factor out the influence of the class skew. And to do this we can turn to basic probability.

Recall that our class probabilities took the form $p(\text{Y} \cap \text{n}) \ldots$.

Probability states that we can use the identity: $p(\text{Y} \cap \text{n}) = p(\text{n}) \cdot (\text{Y}|\text{n})$.

*We will explain this better when we talk about probabilities again in a later section.*

Then the expected profit becomes:

$$\begin{aligned} & (24) \\ \text{expected profit} = & p(\text{p}) \cdot p(\text{Y}|\text{p}) \cdot b(\text{Y} \cap \text{p}) + p(\text{p}) \cdot p(\text{N}|\text{p}) \cdot b(\text{N} \cap \text{p}) + & (25) \\ & p(\text{n}) \cdot p(\text{Y}|\text{n}) \cdot b(\text{Y} \cap \text{n}) + p(\text{n}) \cdot p(\text{N}|\text{n}) \cdot b(\text{N} \cap \text{n}) & (26) \end{aligned}$$

And factoring out the $p(\text{p})$ and $p(\text{n})$:

$$\begin{aligned} & (27) \\ \text{expected profit} = & p(\text{p}) \cdot [p(\text{Y}|\text{p}) \cdot b(\text{Y} \cap \text{p}) + p(\text{N}|\text{p}) \cdot b(\text{N} \cap \text{p})] + & (28) \\ & p(\text{n}) \cdot [p(\text{Y}|\text{n}) \cdot b(\text{Y} \cap \text{n}) + p(\text{N}|\text{n}) \cdot b(\text{N} \cap \text{n})] & (29) \end{aligned}$$

Note the first term, it's the probability of each class. I.e. we've factored out the influence of skew.

Let's repeat the previous example again. We can calculate the mess of probabilities straight from the confusion matrix.

| Model A | p | n |
|---|---|---|
| Y | 25 | 30 |
| N | 0 | 45 |

$$T = 100,\ p = 25,\ n = 75$$
$$p(\mathrm{p}) = 0.25,\ p(\mathrm{n}) = 0.75$$
$$p(\mathrm{Y}|\mathrm{p}) = \frac{25}{25} = 1$$
$$p(\mathrm{Y}|\mathrm{n}) = \frac{30}{75} = 0.4$$

---

| Model B | p | n |
|---|---|---|
| Y | 30 | 0 |
| N | 20 | 50 |

$$T = 100,\ p = 50,\ n = 50$$
$$p(\mathrm{p}) = 0.5,\ p(\mathrm{n}) = 0.5$$
$$p(\mathrm{Y}|\mathrm{p}) = \frac{30}{50} = 0.6$$
$$p(\mathrm{Y}|\mathrm{n}) = \frac{0}{50} = 0$$

---

Model A:

$$\begin{aligned} & (30) \\ \text{expected profit} &= 0.25 \times [1 \times 41] - 0.75 \cdot [0.4 \times 9] & (31) \\ &= 7.55 & (32) \end{aligned}$$

---

Model B:

$$\begin{aligned} & (33) \\ \text{expected profit} &= 0.5 \times [0.6 \times 41] - 0.5 \cdot [0 \times 9] & (34) \\ &= 12.3 & (35) \end{aligned}$$

---

Now we can alter the values of $p(\mathrm{p})$ and $p(\mathrm{n})$ to set Model A's skew to $50:50$.

Model A:

$$\begin{aligned} & (36) \\ \text{expected profit} &= 0.5 \times [1 \times 41] - 0.5 \cdot [0.4 \times 9] & (37) \\ &= 18.7 & (38) \end{aligned}$$

So Model A expected profit per customer is now £18.70, compared to Model B's £12.30. That's quite a difference and caused by skew in the training data.

---

### Notes on Creating Cost/Benefit Matrices

- Make the signs of the costs/benefits consistent. It will get confusing. For example your task might be to minimise costs, which would mean costs would be positive.
- Watch for double-counting. Double-counting is where you add a cost, which is also a benefit (or vice versa). For example, the benefit of acquiring a customer might be £100. But you might also put the cost of not acquiring a customer as -£100. Hence, the improvement of acquiring a customer is £100 - (-£100), which clearly doesn't make sense. Make sure you only count it once.

---

# Other evaluation metrics

We've already seen accuracy, which is:

$$\text{accuracy} = \frac{\text{correct predictions}}{\text{all instances}} = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + FP + TN + FN}$$

???

We've already seen accuracy, the proportion of correct predictions, but there are many other evaluation metrics. The ones that are used are usually domain specific, but most of them are built from combinations of the different rates: TP, FP, TN and FN.

---

### Nuisance alarm rate

One metric I like is the False Positive Rate (a.k.a. probability of false alarm):

$$\text{false positive rate} = \frac{\text{false positives}}{\text{all negatives}} = \frac{FP}{N} = \frac{FP}{FP + TN}$$

This is common in detection domains and is often differentiated over time and often called the Nuisance Alarm Rate (NAR). For example, "with this model we would expect 5 nuisance alarms per day".

---

### Precision and Recall

These are often used in informational retrieval type tasks like text classification.

Precision can be interpreted as how tightly grouped results are. E.g. with arrows and a bow, you might not be very accurate, but each arrow lands in the same place so you are very precise. Another way of defining it is "what proportion of items were relevant".

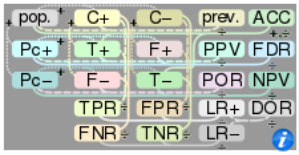
$$\text{precision} = \frac{\text{true positives}}{\text{all predicted yes}} = \frac{TP}{TP + FP}$$

Recall (a.k.a. true positive rate) can be interpreted as "out of the all the relevant items, what proportion were selected".

$$\text{recall} = \frac{\text{true positives}}{\text{all positives}} = \frac{TP}{TP + FN}$$

There a whole load more. If you ever need to recap what they are: visit wikipedia: https://en.wikipedia.org/wiki/Confusion_matrix

| | | True condition | | | |
|---|---|---|---|---|---|
| | Total population | Condition positive | Condition negative | Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$ | Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$ |
| **Predicted condition** | Predicted condition positive | **True positive** | **False positive** (Type I error) | Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$ | False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$ |
| | Predicted condition negative | **False negative** (Type II error) | **True negative** | False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$ | Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$ |
| | | True positive rate (TPR), Recall, Sensitivity, probability of detection = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$ | False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$ | Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$ | Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$ |
| | | False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$ | True negative rate (TNR), Specificity (SPC) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$ | Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$ | $F_1$ score = $\frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}}$ |




Key point: If you can't use a business metric, pick a technical metric that represents your problem.

# Baselines

Being good scientists, we want a hypothesis to validate against.

When developing models, always provide a baseline to compare against.

Baselines are good for weeding out unnecessary models. Models cost money. Implementation time, running time, maintenance time, performance, etc. Removing marginal models is well worth the effort.

Baselines often come from a very simple rule, or from prior work.

For example, a simple rule for our marketing exercise might be to "target all customers that have already spent x amount of money". If you can't beat that, then you might as well stick with the simple rule.

More generic baselines exist, as we have already seen.

- Majority classifier: The most common class
- Average of all past decisions: E.g. if a user has rated two stars, a good baseline is to predict two stars again.

???

A majority classifier is one that simply classifies as the most common class. This is a very good test to make sure you're not having issues with imbalanced datasets.

For models with history, another simple baseline might be to make a decision based on the average of all

past decisions. E.g. If a user has always rated two stars, betting that they will rate this new one two stars too is probably a good baseline.

Finally, the best baselines often come from domain expertise. Indeed, entire algorithms could be defined by someone that has studied the data for years.

---

class: center, middle

# Workshop: 11-numerical-model-evaluation

---

# Visual model evaluation

- Numerical scores are necessary for optimisation.

But visual representations are far more intuitive.

???

Numerical scores are absolutely necessary for optimising algorithms.

However, a visual representation of the data can be far more intuitive for both you and your stakeholders.

For example, it is very easy to see problems with your calculations if you plot them. And your stakeholders are unlikely to know what recall is.

Instead, lets investigate some visual methods of model evaluation.

---

# Ranking

- Most algorithms output some kind of score

For example:

- Probabilities
- Distances
- Errors

We can alter the *threshold* of that score to generate a new confusion matrix.

E.g. Logistic classification: We usually pick 50% to be the threshold; but that's not necessary.

Draw this

???

Draw this

Very often, algorithms output a score, where the score represents some sort of confidence in the result. Sometimes this can be a probability estimate (e.g. logistic regression), other times it is simply a measure of

how close the sample is to the predicted class (k-means).

When this happens, there is no single confusion matrix. Instead, it depends on where you *threshold* the output.

For example, you could take the first, best score and compute the confusion matrix. It is likely to be accurate, but will not lead to many positive hits.

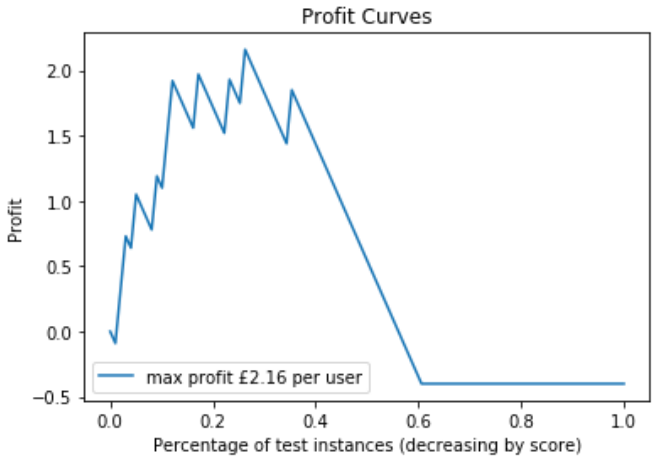
Instead we can move the threshold down through the best scores and generate a confusion matrix for each result.

That way, we are using the scores rank to determine which results to target first.

---

# Profit curves

The logical extension to ranking is to calculate the expected value for each rank. That way we can easily find the point at which we would become most profitable.




---

# ROC plots and curves

- Profit curves are great, because everyone can understand them.

But sometimes it's hard to get profit/loss values.

Instead we have to use a technical metric like accuracy, etc. and plot that.

- Receiver Operating Curves (ROC) plot the *false positive rate* vs. the *true positive rate*

???

Profit curves are great, because they are perfectly aligned with a business's needs, but they can be difficult to generate because of requirements of the expected value method.

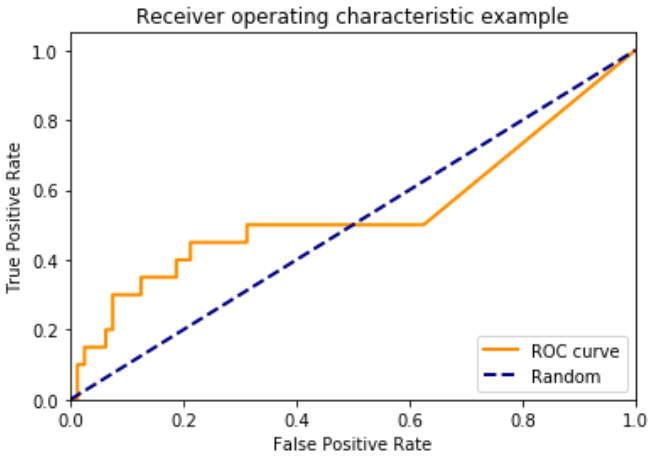
True population statistics may not be known and cost/benefits might be hard to estimate.

Despite this we can continue to visually compare models using some numerical evaluation of the confusion matrix, and rank in the same way.

This is called a Receiver Operating Characteristic (ROC) curve. *Sorry about the name, it originated in times when radio was a thing!*

ROC curves are created by ranking each sample and plotting the false positive rate on the x axis against the true positive rate on the y axis.

---




There are some important points on the curve...

???

- (0, 0) represents the point where no positive classifications are made and no negative classifications are made. This is because the "threshold" is higher than the highest sample.
- (1, 1) represents the point where all samples are declared positive, i.e. we positively identify everything, but also get all the rest wrong.
- In between these two points we have a compromise between including samples and not including false positives.
- A perfect classifier will have a point at (0, 1). I.e. no false positives, all true positives.
- A random classifier will have a straight line between (0, 0) and (1, 1)

---

## AUC

- We can summarise a ROC plot by measuring the *area under the curve* (AUC)

Key point: This is a very good metric because it accounts for class biases.

???

Visually, this is grand. But often we want to summarise the ROC results into a summary statistic.

The most often used metric is the *area under the curve* (AUC).

This is simply the area... under.. the... curve. It removes some of the detail about a classifier (e.g. are ROC curves skewed?), but generally it is a very robust measure of performance.

For the previous example, the AUC is 0.53.

---

## Cumulative Response

- Cumulative response curves are easier to understand.

The plot the *proportion of included samples* vs. *true positive rate*.

Key point: this doesn't represent false positives well, but is easier to understand

???

If you're not confused over what ROC curves are, well done.

But most stakeholders will find it hard to interpret what ROC curves mean to them and what on earth an AUC is.

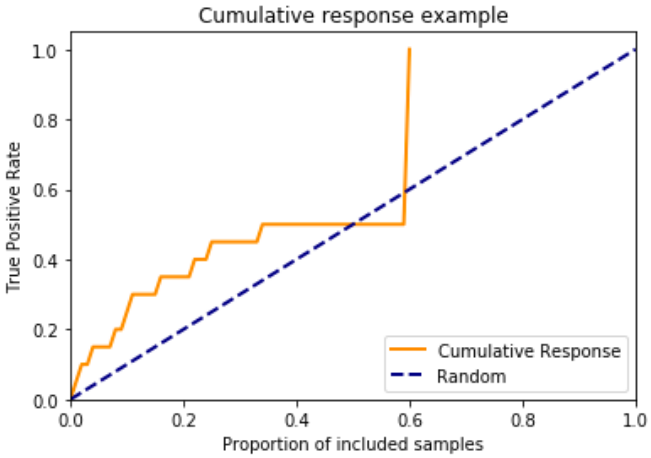
It is a data scientists responsibility to distill the result into something understandable.

One common method is to plot the same true positive rate on the y-axis, but then plot the proportion of the population on the x-axis.

The result is something that looks similar (and doesn't represent false positives very well) but is much easier to explain. This is called the *cumulative response*

---

- The y-axis in the percentage of "hits", the number of positive outcomes.
- The x-axis is the proportion of samples included for that positive outcome.




---

## Lift Curve

Another minor extension commonly used is to divide the cumulative response curve by the line representing random chance.

This results in a similar plot except the y-axis represents the relative improvement against random chance.

---

### Warning!

- Both the cumulative response and lift curves only plot positives!!

This means that if we balance classes, or have more positives than is realistic, it will have better than expected performance.

*If in doubt, use a realistic example. Or just stick to a ROC curve.*

???

A quick warning. Both the cumulative response and lift curves do not factor out the proportion of positive examples (we are including only the positives over the threshold).

This means that the resulting y-axis is dependent on the number of positive instances in the dataset.

If the the number of positives don't represent the true population, then the y-axis is wrong.

For example, in fraud, people often use equal numbers of positive and negative instances.

If we plotted the cumulative response curves, then they would include far fewer negatives than are actually seen in real life.

Be very careful.

> *If in doubt, use a realistic sample.*

---

class: center, middle

# Workshop: 12-visual-model-evaluation

---

---

name: evidence-and-probabilities

# Evidence and probabilities

- Thinking probabilistically helps you reason about your data
- But only works well when it makes sense to think about probabilities. E.g. not regression.

In this section:

- Probability part duex
- Bayes rule
- Bayesian classification
- Evidence lift

???

There is a another very popular framework for data science problems. Thinking probabilistically about a problem allows us to ask some very interesting questions.

Using a probabilistic method is not just another classifier, though. Through the development of a model we will generate statistical models for each component of the data.

This is another way of characterising the data, like we did when we looked at unsupervised classification in the first training course.

In this section we will be covering:

- Probability
- Bayes Rule
- Bayesian Classification
- Evidence Lift

---

# What's the Big Idea?

- We've been solving problems by brute force

*This is not scientific and might not be reasonable in custom solutions*

???

Up to now, we've generally be solving all our data science problems with brute force. We've been provided with a fixed problem, some data, and we try several algorithms and pick the best one.

This works well for well defined simple examples, but doesn't scale very well into real, custom solutions.

Instead, we can alter our perspective of the problem to one that isn't simply "classify this".

---

- Try to change your thinking about what features are.

*Features are manifestations of some property of the instance. The things we know about that instance are represented in the features.*

- Features are just evidence pointing towards a target

*If we knew how much each of those features corresponded to the problem, we could combine the (best?) features together and probabilistically estimate the value of the target.*

- It is our job to figure out how to combine the evidence to draw a conclusion.

# Combining evidence

- So how do we combine evidence?

*The first step in any experimental trial is to try and establish the population's base rate.*

For example, if we were trialling a new advert, we could show the advert to a random sample of the population.

Then we could say that "1 in 1000 random people shown the advert bought our product".

---

- Next: Try an beat the base rate

*Use the features of the data to improve the probability that a person will buy our product*

Eg. probability of becoming a customer, C, is 1/1000. So $p(C) = 0.0001$.

- We want to estimate the probability of C *given* some evidence $\mathrm{E}$.

*This is written as*
$p(C|\mathrm{E})$.

- How do we calculate that?

???

Typically the next step would be to try and increase that hit rate. And we could do that by only showing the advert to people that we think are going to buy.

In other words, we need to use each instance's features to estimate which instance is most likely to buy.

The issue is, how do we estimate this probability? If we have some statistics for someone with *exactly* the same features, then we could use a lookup table. But this is unlikely and also misses the point.

We want to generalise. What are the general themes in the features? What makes a good customer?

Armed with this information we can make much more convincing arguments.

---

# Joint probability

We've already seen this, but let's just go over it one more time.

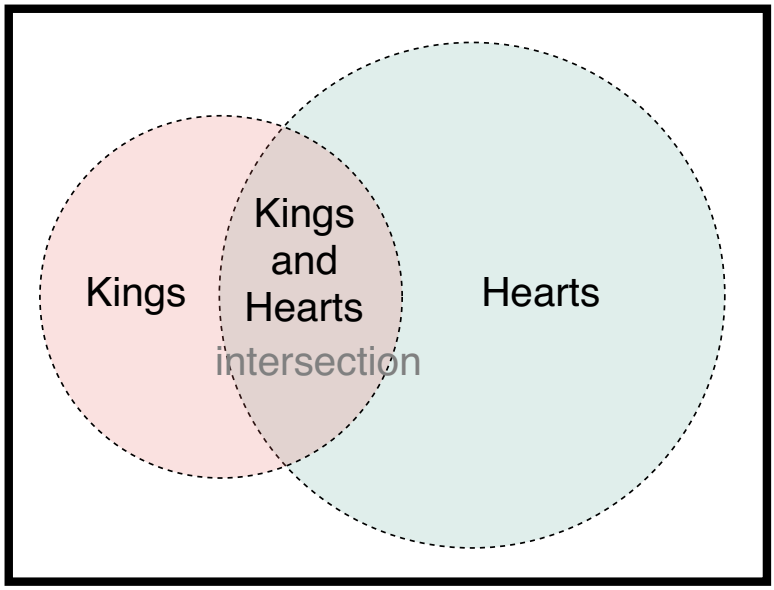
If we have two *independent* events, $A$ and $B$ the the probability of a joint event is:

$p(A \cap B) = p(A) \times p(B)$

For example, given two dice, the probability of getting two ones is $1/6 \times 1/6 = 1/36$.

---

class: center, middle




---

However, imagine they weren't independent. Imagine that once I threw one die, the other would always result in the same score as the first. In this case, the two dice are dependent on each other.

In this case, we want to know $p(A \cap B)$ *given* that $B$ has already occurred. This is known as the *joint probability of dependent events*:

$p(A \cap B) = p(B) \times p(A|B)$

---

- We're introducing causality

$p(A|B)$ is not the same as $p(B|A)$.

???

Note that we're introducing a causality here. We're saying that A is conditional on B. I.e B must happen before A can happen.

Because of this causality, $p(A|B)$ is not the same as $p(B|A)$. Hence $p(A \cap B)$ is only equal to $p(B \cap A)$ if they are independent.

For example, imagine the probability that someone would purchase something, P, was dependent on one of their features, their age for example, A.

The probability of their purchase given their age, $p(P|A)$, is not the same as the probability of their age given a purchase, $p(A|P)$. This can be proven mathematically too. Just rearrange the equation.

---

## More Examples

- Fake dice example

$$p(D_1) = 1/6, p(D_2|D_1) = 1$$

$$p(D_2 \cap D_1) = p(D_1) \times p(D_2|D_1) = 1/6 \times 1 = 1/6$$

If they were independent then $p(D_2|D_1) = 1/6$

???

Let's convince ourselves of that equation a bit more. You'll understand why in a moment. ;-)

So let's use the dependent dice example now. The probability of throwing a number on one die is $1/6$. But what is the probability of throwing the same number given than the first die has been thrown? Well, in our fake dice example, the probability is $1$.

In other words, we know exactly what the second die is going to show given the first has been thrown. So let's pop that into our equation:

$p(A \cap B) = p(B) \times p(A|B) = 1/6 \times 1 = 1/6$

Now imagine the two dice are independent. The first throw is the same, $1/6$. But now, because the second die is independent of the first, we don't know what that number is going to be. So, the probability is again $1/6$.

$p(A \cap B) = p(B) \times p(A|B) = p(B) \times p(A) = 1/6 \times 1/6 = 1/36$

In other words, if they are independent, the equation collapses back to the joint probability.

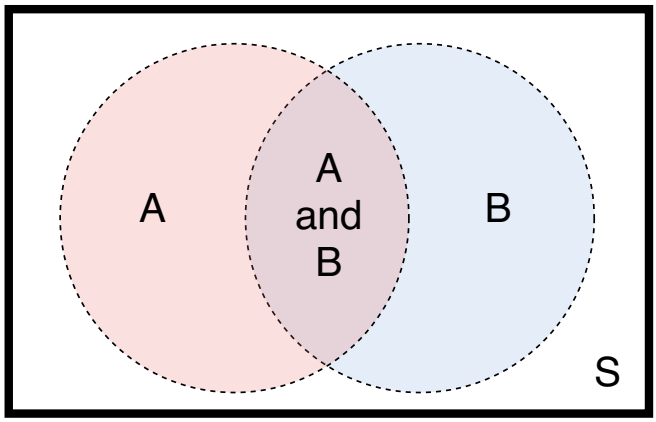
---

## Intuition

If we rearrange the equation, this makes more sense. This is called conditional probability:

$$p(A \cap B) = p(B) \times p(A|B)$$

$$p(A|B) = \frac{p(A \cap B)}{p(B)}$$




---

# Bayes Rule

$p(A \cap B) = p(B) \times p(A|B)$

Notice how arbitrary the choice of ordering is in this equation. $p(A \cap B)$ is not conditional. We could have easily written:

$p(B \cap A) = p(A) \times p(B|A)$

So in other words:

$p(A) \times p(B|A) = p(A \cap B) = p(B \cap A) = p(B) \times p(A|B)$

And therefore:

$p(A) \times p(B|A) = p(B) \times p(A|B)$

---

If we divide by $p(A)$:

$p(B|A) = \frac{p(B) \times p(A|B)}{p(A)}$

Wooh! You've just derived one of the most important mathematical theories of the 18th century!

---

## Working With Bayes

- Very important equation; will change your perceptions

Let's use more intuitive symbols. Let $H$ be a hypothesis and $E$ be some evidence.

$p(H|\mathrm{E}) = \frac{p(H) \cdot p(\mathrm{E}|H)}{p(\mathrm{E})}$

???

Again, let's take our time working with the bayesian equation. I'm not overstating by saying that this can alter your perception of probabilities and life more generally.

Let's give the letters more intuitive names. Let's call our ultimate requirement, the hypothesis, $H$ and let's call the evidence, $\mathrm{E}$ (bold because it is likely to be a vector). Then the bayes formula becomes:

$p(H|\mathrm{E}) = \frac{p(H) \cdot p(\mathrm{E}|H)}{p(\mathrm{E})}$

In words, the probability of a hypothesis, $H$, given some evidence, $\mathrm{E}$, is equal to...

the probability of the evidence given the hypothesis, $p(\mathrm{E}|H)$...

times the probability of the hypothesis $p(H)$ ...

divided by the probability of the evidence $\mathrm{E}$.

The reason why this is so powerful is that it means we can turn one conditional probability that is difficult, if not impossible to measure into a set of probabilities that are easier to measure.

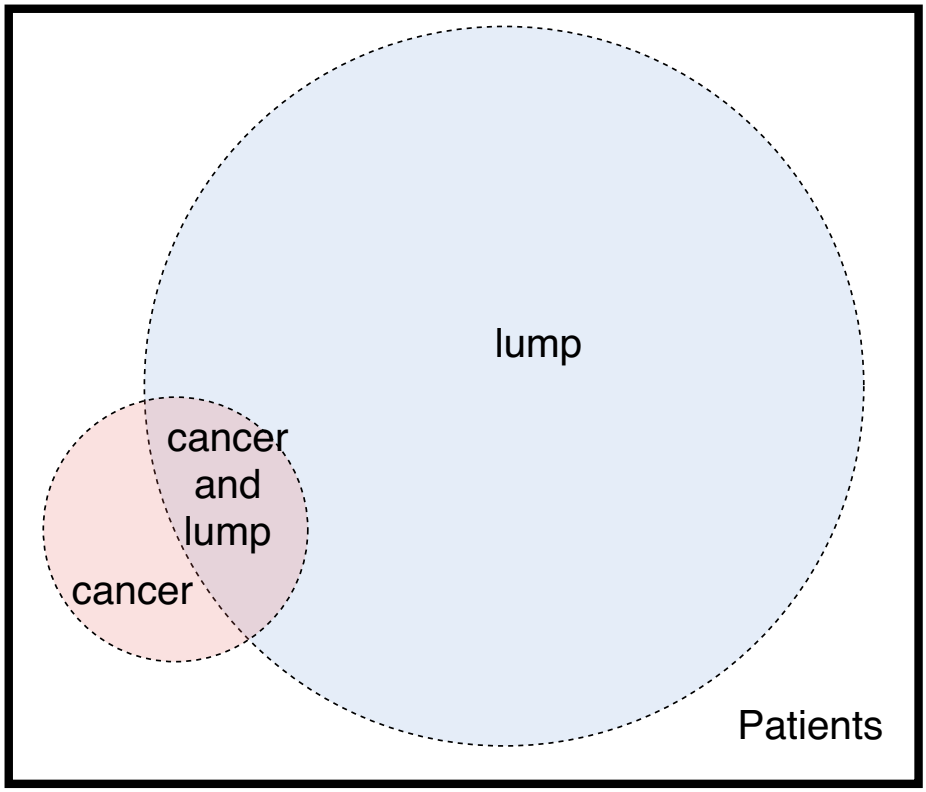
The classic, and most revealing, example is that of cancer detection. Let's walk through an example.

---

## Bayes Example: Cancer diagnosis

$$p(cancer \cap lump) = p(lump) \times p(cancer|lump) \tag{39}$$

$$p(cancer|lump) = \frac{p(cancer \cap lump)}{p(lump)} \tag{40}$$




???

Imagine you are a doctor and a patient has walked into your office with a lump. You immediately think that this might be cancer (because the most natural thing to think is the worst case).

So what you want to know is what is the probability that the person has cancer (your hypothesis) given that they have a lump (the evidence).

In order to calculate $p(cancer|lump)$ directly, we can go back to our standard conditional joint probability:

We would need to know the probability of anyone having a lump and the probability of anyone having a lump and having cancer.

But this is next to impossible to calculate. How many people with cists or lesions or bites have we seen? Do bruises count?!

---

Instead, we can use bayes:

$$p(cancer|lump) = \frac{p(cancer) \cdot p(lump|cancer)}{p(lump)}$$

Now all we need is:

- Probability of cancer in the population
- Probability that someone has a lump, given that they have cancer
- Probability of a lump is just a normalisation factor

???

Now all we need to estimate is the probability of cancer in the general population (we can look that up) and the probability that someone has a lump, given that they have cancer. Again, we can ask an expert to provide that.

(Intuitively, we know that breast and testicular cancer is very common, so rates are likely to be high and we know that cancer rates over a lifetime are quite high - around 1/3, but not that common on a day to day

basis).

Finally, we still need an estimate of the probability of having a lump. This can still be hard to estimate. Thankfully, quite often we don't need it, as we will see, because we are comparing models to the same population, so it cancels out.

---

## Example: Cancer Diagnosis

- Given a positive test for breast cancer, what are the chances that you actually have cancer?

*You will be surprised by the result.*

$$p(cancer|diagnosis) = \frac{p(cancer) \cdot p(diagnosis|cancer)}{p(diagnosis)}$$

- $p(cancer) \approx 1\%$
- $p(diagnosis|cancer) \approx 80\%$
- $p(diagnosis)$ = $p(cancer) \times TP + p(notcancer) \times FP$
- $= (0.01 \times 0.8) + (0.99 \times 0.1) = 0.107 \approx 0.1$

???

An even more insightful example is that of cancer diagnosis.

If you have received a positive test for breast cancer, what is the chances that you actually have breast cancer?

Most people, when they get a positive test, immediately think they have cancer, but this isn't actually true.

$p(cancer|diagnosis) = \frac{p(cancer) \cdot p(diagnosis|cancer)}{p(diagnosis)}$

The probability that you have breast cancer is around 1% - $p(cancer)$. The probability that you get a positive diagnosis given that you have cancer is around 80% - $p(diagnosis|cancer)$.

The probability of a positive diagnosis, $p(diagnosis)$, is the probability that someone gets a positive, irrespective of whether they have cancer or not. I.e. This is $TP + FP$.

For mammograms, the typical FP is around 10%. And this is in a population where 1% have cancer.

So
$p(diagnosis) = (p(cancer) \times TP) + (p(not_cancer) \times FP) = (0.01 \times 0.8) + (0.99 \times 0.1) = 0.107 \approx 0.1$
.

---

Hence:

$p(cancer|diagnosis) = \frac{p(cancer) \cdot p(diagnosis|cancer)}{p(diagnosis)} = 0.01 * 0.8/0.1 = 0.08 = 8\%$

Key point: If you get a positive mammogram result, you only have about 8% chance of actually having breast cancer.

*If we have 100 people in a room, around 1 of them will have breast cancer. But 10 of them if tested would get a false positive.*

???

Let's think about it intuitively.

If we have 100 people in a room, around 1 of them will have breast cancer. But 10 of them if tested would get a false positive.

So, the probability of you having cancer given that you have a diagnosis is only about 1/11.

And this is all due to the errors in the mammogram.

However, before you start turning down surgery, this doesn't take into account any other evidence, like lumps. And people normally go for more tests before they have surgery.

And, because the surgery isn't life threatening and doesn't cause to much discomfort post-surgery, the tests are designed to be overly sensitive and simple to perform.

Still... Remember this if you ever here statistics about false positives...

---

# Applying Bayes' Rule to Data Science

Back to the data.

With a slight reformulation we can apply Bayes rule to classes:

$p(C = c|\mathbf{E}) = \frac{p(C=c) \cdot p(\mathbf{E}|C=c)}{p(\mathbf{E})}$

- $p(\mathbf{E}|C = c)$ is the probability of seeing the evidence, $\mathbf{E}$, when the class is $c$

*We probably haven't seen exactly the same evidence, so can't estimate probabilities.*

???

Here, we're asking "what is the probability that this instance belongs to class $c$ given the evidence.

$p(C = c)$ is known as the "prior" probability of the class. I.e. from the general population, what proportion of instances belong to class $c$.

This could be calculated from the data, or provided by some prior belief. This is an exceptional way of introducing theory into your model.

$p(\mathbf{E})$ is the probability of of seeing the evidence, $\mathbf{E}$, in the population. This too can often be calculated from the data.

$p(\mathbf{E}|C = c)$ is the probability of seeing the evidence, $\mathbf{E}$, when the class is $c$. Like before, it is often difficult to calculate this, because it relies on seeing instances with exactly the same set of features, which is unlikely.

So how do we get around this?

---

# Conditional independence

We get around this by assuming that each feature is conditionally independent of each other. I.e. $e_0$ is independent of $e_1$ given class $c$.

Let's look at the tricky part of the equation more closely:

$p(\mathrm{E}|C=c) = p(e_1, e_2, e_3, \ldots|C=c)$

We know that if they are dependent, then that would result in a load of:

$p(e_1, e_2, \ldots|C=c) = p(e_1|C=c) \cdot p(e_2, \ldots|e_1, C=c)$

???

Add $|C$ to the end of the standard conditional probability equation:
$p(A \cap B) = p(B) \times p(A|B) => p(AB|C) = p(B|C) \times p(A|BC)$

And trying to calculate $p(e_2, \ldots|e_1, C=c)$ would be just as difficult.

---

If, however, we assume that features are independent, then we can just use the simple indepdendent joint probability, which results in:

$$\begin{aligned} p(\mathrm{E}|C=c) &= p(e_1, e_2, e_3, \ldots|C=c) && (41)\\ &= p(e_1|C=c) \cdot p(e_2|C=c) \cdot \ldots && (42) \end{aligned}$$

*We have lots of examples of evidence from a class (or we can model it), this is much easier.*

???

In other words, we now need values for the probability of seeing an individual $e_i$ in a class. The chances of seeing a similar $e_i$ in a class is much higher which makes the problem much easier.

---

# Naive Bayes Classifier

This is the basis for the naive bayes classifier. It's "naive", because the independence assumption.

$p(C=c|\mathrm{E}) = \frac{p(C=c) \cdot p(e_1|C=c) \cdot p(e_2|C=c) \cdot \ldots}{p(\mathrm{E})}$

You may still question how "easy" it is to obtain $p(\mathrm{E})$.

???

For many problems it isn't an issue at all, because when making the classification decision we can simply pick the highest result. I.e. the normaliser at the bottom doesn't make any difference to the result, so we can omit it.

---

If we do need proper probabilities for each class (this is quite useful) then it is still fairly easy to calculate because usually classes are independent and mutually exclusive. E.g. you can't both have a positive and negative cancer diagnosis. Hence:

$$
\begin{aligned}
p(\mathrm{E}) &= p(\mathrm{E}c_0) + p(\mathrm{E}c_1) + \ldots && (43)\\
&= p(\mathrm{E}|c_0) \cdot p(c_0) + p(\mathrm{E}|c_1) \cdot p(c_1) + \ldots && (44)\\
&= p(e_0|c_0) \cdot p(c_0) + p(e_1|c_0) \cdot p(c_0) + \ldots + p(e_0|c_1) \cdot p(c_1) + p(e_1|c_1) \cdot p(c_1) + \ldots && (45)\\
& && (46)
\end{aligned}
$$

---

So the full equation becomes:

$$p(C = c|\mathrm{E}) = \frac{p(C=c) \cdot p(e_1|C=c) \cdot p(e_2|C=c) \cdot \ldots}{p(e_0|c_0)p(e_1|c_0)\ldots p(e_k|c_0)p(c_0) + p(e_0|c_1)p(e_1|c_1)\ldots p(e_k|c_1)p(c_1) + \ldots}$$

???

Although that looks messy, we're simply summing up how much each bit of evidence weights towards a particular class, multiplied by a class prior.

Once you code it up, it's not so bad.

Finally, we would choose a class by picking the class with the best probability (or score if you're not calculating the denominator).

The only caveat with this step is that the "best" depends entirely on how the data is distributed. If it's gaussian, then the best is the highest. If it's not, the you might need to think more carefully.

In practise, assuming a gaussian usually works pretty well.

---

# Pros/Cons

- Simple (if you ignore the probability math!)
- Effective at incorporating evidence from sources other than the data
- Works suprisingly well, even when the evidence isn't independent
- Is easy to make it update "on-line". Simply add the new data point.
- No training time, just a calculation

Cons:

- Be careful if using the actual probabilities if evidence is not independent. If two features were correlated, naive bayes would effectively double count it. It would report that you have twice as much evidence, when you really don't. (e.g cost/benefit analysis)

---

# Evidence "Lift"

In the previous session we consider the idea of lift. Given some base rate, how does some new model raise the probability of success.

We can apply that here too. Given some base rate, what is the increase in hit rate given some new evidence. In other words, we begin with:

$p(c|\mathrm{E}) = p(c)$

---

If we make the assumption that the evidence is completely independent of class (i.e. not conditionally independent on $c$), then if we had two new bits of evidence we can recalculate with:

$$p(c|\mathrm{E}) = p(c) \cdot \frac{p(e_0|c)}{p(e_0)} \cdot \frac{p(e_1|c)}{p(e_1)}$$

We can generalise to:

$$p(c|\mathrm{E}) = p(c) \cdot lift(e_0) \cdot lift(e_1) \cdot \ldots$$

Where:

$$lift_c(e) = \frac{p(e|c)}{p(e)}$$

---

For example, imagine a marketing funnel where we are interested in people buying a product.

If a customer visited a page, that would lift them by $x\%$. If they downloaded a trial, that would lift them by more $\%$:

$$p(buy|\mathrm{E}) = p(buy) \cdot \mathrm{lift}(e_p) \cdot \mathrm{lift}(e_t)$$

If they then made a call, we could multiply by another lift.

---

class: center, middle

# Workshop: 20-evidence-probabilties

---

name: changing-landscapes

# Changing Landscapes

.right-column[ 

]

.left-column[

In 2009 a data science competition was held by Netflix. The winners received $1 million.

They developed a sophisticated set of algorithms that was better able to provide recommendations for Netflix viewers.

]

---

*The model was never used!*

- Netflix stopped being a mail-order company
- The model was massively complex and hard to implement

???

However, by the time the competition was won, Netflix stopped being a mail-order company and was moving towards streaming content.

With streaming services, if people were recommended bad movies, they could easily stop it and start another one. This is far easier than having to send the DVD back in the post.

So by the time the competition was won, there was only a need for a "good-enough" algorithm. And the winning algorithm was so complex it would have cost a fortune to implement in production.

The model was never implemented!

---

# Changing Conditions

Throughout this course, we have only considered static, stationary datasets.

This short chapter:

- Consider what can change when in production
- How to make your models more robust

???

In this chapter we'll consider what it is like to implement a model in production, and what we can do to make sure that our models are actually used!

For example, if you enter a data science competition, you won't have the test dataset available to use. That is used to score entries.

So we need to consider what might change in the data, so we can make our models more robust.

---

# Robust Models

*Robustness* is a measure of how effective your model is when applied to another data set.

Datasets comprise of signal, features improve the evidence of a prediction, and noise, spurious data that does not add or even obfuscates evidence.

- When we train models, we train on signal and noise

???

When you train models on data you will inevitably learn features from both the signal and noise. We have already discussed overfitting, which is the practice of fitting a model that so closely that we accidentally learn features of the noise, which does not help identify the signal.

---

- Already seen techniques to avoid overfitting

But we can do more:

- Intentionally add noise
- Intentionally remove the amount of information being trained upon

*These techniques are more useful for highly complex models*

???

We've already seen some techniques to avoid overfitting. Namely train-test-validation set splitting and cross-validation. These are both commonly-used and widely acknowledged techniques.

To be frank, you should always apply these techniques to every problem. There is no reason why you should not.

But there is another concept too. One that is used heavily in deep learning.

It is to actively add noise, or reduce the amount of information going into the training process.

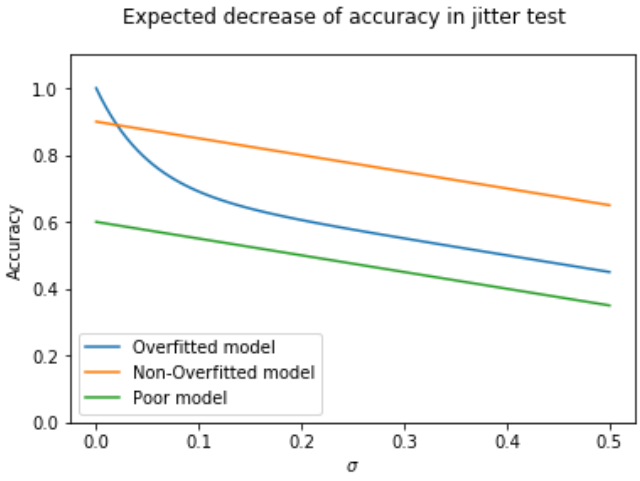
The idea is that by randomly leaving out data or randomly adding noise, you're forcing the model to make generalisations, rather than relying on very specific, obtuse features of the data.

These techniques are mostly useful for models that can model high complexities. I.e. there isn't much point applying these techniques to logistic classifiers because they aren't able to model complex processes in the first place. This means it tends to only be used in deep learning.

---

A similar concept is used to test a model:

- Add noise to the test dataset; watch how it behaves




???

But there is another, related technique. We can first train a model. Then, when testing, we can randomly add noise to the test dataset. If we see performance drop dramatically, then we know that we have overfit and or trained upon the noise.

However, if we see a gradual, graceful degradation of performance, then we know that the loss in performance is purely due to less information.

---

# Risk factors

Overfitting is the first, and possibly most important risk in your model.

But if we imagine that we want to deploy our model for repeated use, then there are a range of other risks that you should be aware of.

---

## Cyclic Changes

- Features often shift over time.

Example cycles:

- Day/Night
- Weekday/weekend
- Financial crashes
- Seasons/Holidays

???

Models that generalise well may not account for changes in the future.

The most common types of change are those based upon cycles. For example seasonal cycles, or financial crashes.

Shifts might be over an even shorter timescale like those over night/day, evening/morning, weekday/weekend, etc.

It makes sense, where possible to attempt to factor out, or model these variations. Your models will typically be more accurate as a result.

---

## Incremental Changes

- Some changes are incremental in nature.

Examples:

- New data might be corrupted by your recommendations engine.
- Users might stop randomly viewing news articles

*This is known as selection bias*

???

For example, if you're working with customer data, then over time more data might become available as new features are found and stored.

Results might also change systematically. For example if you introduce a recommendations system, your users might stop randomly picking new products.

Recommended systems might fail because of a selection bias, i.e. your model keeps recommending the same thing over an over again because you recommended and they bought that thing in the past.

---

### Fundamental Data Changes

The worst case scenario is that some of your key features change or are no longer available.

For example, consider the situation where a new law prevents you from storing financial data on your system. If the financial data was a key feature, your model will suffer as a result.

For this reason it is recommended that you keep track of what you key features are.

---

# Mitigations

- Investigate all features. Keep a report (ipython notebook?) showing:
  - Most important features
  - How features are affected by time
  - How features might be affected in the future
  - Are the features particularly sensitive or risky?
- Investigate ways to mitigate against future changes.
  - Can you factor out or model the varying feature?
- Add automated robustness tests.
  - What happens when you add a trend to a feature?
  - What happens when you add jitter?

---

# Strategies to Improve Model Robustness

- Champion/Challenger: Run several different models at the same time. Continuously test which model performs the best. Use the one that wins.

Although be careful about switching too quickly. Confidence in a particular model is gained over time. The new model might just happen to be stuck in a local minima, but doesn't generalise well. - Continuous/Repetitive Integration: Continuously run automated tests against your model. You should be the first to know if a model is no longer performing adaquately. New data should be integrated into the automated tests. Tests should be run repeatedly with new data, even if the model hasn't changed. - Model History: Make sure you have the ability to regenerate a model from any point in time. You may need to go back or refer to an earlier version. Use version control.

---

class: center, middle

# Workshop: 30-changing-times

---

name: dimensionality-reduction

# Dimensionality Reduction

The number of dimensions affects:

- Computational cost
- Generalisation

*It affects some algorithms worse than others*

*Obvious non-functional limits, like maximum training time*

???

The number of dimensions in a dataset (generally equivalent to the number of features) greatly affects both computational and model performance.

Dimensionality affects some algorithms more than others, but generally the computational complexity increases at least linearly with the number of features, sometimes exponentially.

We want our models to train and predict as fast as possible. You will probably have some non-functional requirements stating some limits. E.g. you don't want to have to wait three days for a model to train.

Often you can overcome some computational complexity by throwing more processing power at the problem. BUt it is usually much easier to simply reduce the numbers of features.

---

- Models expend effort trying to optimise for each feature
- Some underlying features can overpower others

???

It can also affect model performance. Models aim to generalise rules that are able to predict certain outcomes. If you have many inputs, it will expend effort in trying optimise each feature to produce the output.

In the worst case, some noisy, large scale data will overpower smaller, more subtle, but possibly useful features simply due to the optimisation algorithm (remember scaling?).

Features can also be correlated, which means that some algorithms will overweight certain features because they seemingly have twice as much predictive power.

---

- Mental burden, simpler models are easier to understand and explain

*We must reduce the number of dimensions*

???

Finally, you want to reduce the numbers of features because it will ease the mental burden and improve the ability to debug.

For all these reasons and more, fewer features is usually better. Occams Razor.

One fundamental algorithm to reduce the amount of data is called principal component analysis.

In short, PCA attempts to map high dimensional data onto lower ones, by finding the dimensions with the most information.

We'll delve into the algorithm now.

---

# PCA

- Finds *orthogonal* components (i.e. independent features)
- Ranked by information content
- Pick the first X components to represent your dataset

???

Principal Component Analysis (PCA) is a decomposition technique that attempts to find an orthogonal set of components - data that are totally independent from one another. The components are then ranked by information content.

The idea is that you pick the first X number of components. These components then hopefully represent the vast majority of the variation in your data.

This means we can take a very high dimensional dataset, perform PCA to find the principal components, pick the first two, and voila, you have a two-dimensional dataset.

---

- We are intentionally throwing away data, there is no way to get it back

An important distinction to make at this point is that we are intentionally throwing away data.

Once you have performed PCA you will not be able to regenerate the original data.

But the idea is that you only throw away the uninformative parts, so you still have the same amount of informative power.

But data regeneration might be important in some domains.

---

The PCA algorithm is performed as follows:

1. Find the centre point
2. Calculate the covariance matrix
3. Calculate the eigenvectors of the covariance matrix
4. Orthonormalise the eigenvectors
5. Calculate the proportion of variance represented by each eigenvector

Let's recap that terminology...

---

## Covariance

This is the variance over multiple dimensions.

Each element in the matrix measure the variance between two distinct features.

So for example, if you had three features, then you would need a $3 \times 3$ covariance matrix.

Element [0, 0] would calculate the variance of itself. Element [0, 1] would calculate the variance between the zeroth feature and the first feature. Etc.

The diagonal, [0, 0], [1, 1], ... in this case, is the variance of each feature.

---

## Eigenvector

This measures the direction of the central axis of the data in question.

An eigenvector for two variables would require a matrix of size $2 \times 2$.

Each value represents a vector in the direction of the central tendency of the data.

Imagine a linear regression problem. The eigenvector would be a vector pointing in the direction of the line of best fit.

---

## Orthoganalisation and Orthonormalisation

*Orthoganalisation* is the process of finding two vectors that are orthogonal (at right angles) to each other.

For example, imagine the linear regression problem again. If we had one vector pointing through the data that traces the line of best fit (the eigenvector) then the orthogonal vector would be the one that is perpendicular to the eigenvector.

*Orthonormalisation* is the process that normalises the orthogonal vectors.

---

## Proportions of Variance (Eigenvalues)

Finally, the eigenvalues for each vector are calculated.

Eigenvalues are used to calculate the proportion of variance represented by each eigenvector.

---

## PCA Summary

In summary, the we use eigenvector analysis to establish the directions of most variance.

Eigenvectors go through a process to produce a set of orthogonal, normalised vectors.

The vector with the largest eigenvalue accounts for the highest amount of variance and is called the first principal component.

Subsequent components account for the rest of the variance.

---

## PCA Example

- We can project data onto these new axes

1. Find the principal components
2. Transform the data with a dot product
3. Only keep the first X number of dimensions

???

The key to PCA is that not only does it describe which orthogonal components account for the most variance, you can also project the original data onto the new principal components.

So you would transform your data by performing a dot product with the principal components, then throw away the dimensions that represent the low variance.

This is how you generate a low dimensional dataset from a higher one.

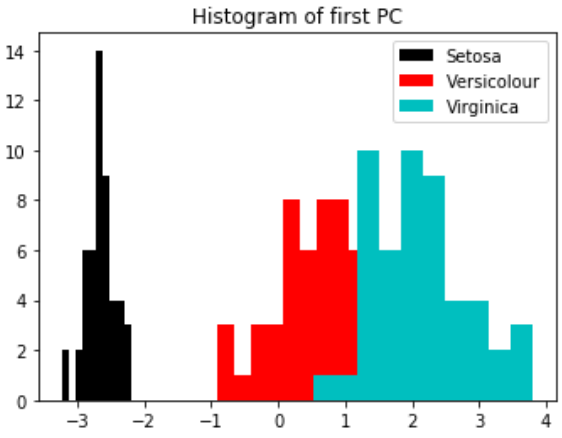
After all that mind bending, let's take a look at an example.

---

These images show the first and second principal components of the iris dataset.

You can see that there is much more variation in the first dimension (x on the 2d plot) than in the second.

.left-column[




]

.right-column[ 

]

---

name: self-organising-maps

# Self Organising Maps

Continuing the idea of dimensionality reduction, self organising maps (SOM) again aim to reduce the number of dimensions produced from a high dimensional dataset.

Despite the similar aims, the algorithm is quite different.

It's also hard not to talk about neural networks, because it's resemblance is uncanny.

Let's begin.

---

1. Start with a high dimensional input
2. Initialise a matrix
3. The matrix is connected to each feature by a set of weights
4. Find the cell that best matches each input (euclidean distance)
5. Update the cells (and neighbours) to better match the input

???

SOMs start with a high dimensional input. Imagine the iris dataset for example (not really high, but is easy to reason about).

The iris dataset has four dimensions: Sepal Length, Sepal Width, Petal Length and Petal Width, if you're interested in that sort of thing.

We initialise a matrix (in some way).

The matrix is connected to the input via a series of weights. Each value in the matrix has a set of weights. The weights are connected to the input data, where the number of weights are equal to the number of inputs.

We then find the value in the matrix that best matches the input (euclidean distance).

This is known as the best matching unit (BMU).

We then update the value in the matrix, and its neighbours, to better match the input.

---

Let's rewrite that a little more succinctly:

1. Randomise a matrix
2. Using the values of the weights, generate a prediciton of the next instance
3. Find the matrix value, pick the best match
4. Update the weights of that matrix value and it's neighbours

Obviously we have to repeat this process many times to reach an adaquate result.

But the result is that all inputs are (in some way) encoded in that matrix.

And clearly, again, this is a destructive process. You cannot get the original data back, but you can generate some typical examples.

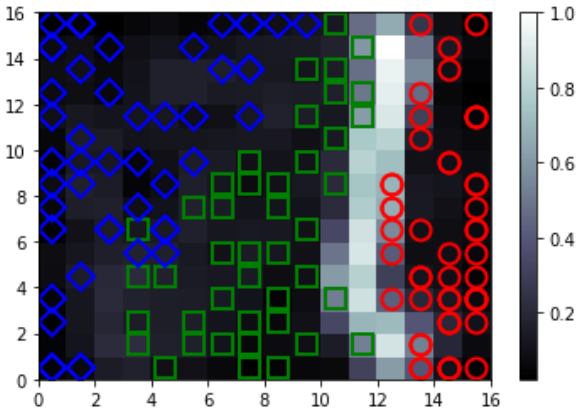
---

We haven't covered it yet, but this has a remarkable resemblance to neural networks. They too have weights that are summed into values in a matrix.

And a particular type of neural network learning, called auto-encoding, also uses a prediction of the input to adjust it's weight.

---

Below is the SOM for the Iris dataset.

.center[ 

]

- Pixel intensities represent the normalised distance to the input data. 0 is good.
- The markers represent observations transformed into the SOM domain
- You can see that they are clustered quite well.

*Recap: We've just mapped four dimensions into 3D, but this works just as well with many dimensions.*

???

The pixel colours represent the normalised distance to the data. So 1 means it doesn't represent the data at all. 0 means it perfectly represents the data.

On top of that we have plotted the result of transforming each instance onto the SOM. I.e. if the values of an instance were transformed into the SOM domain, where would it sit?

Each colour/shape represents one of the three target variables for the Iris dataset.

You can see that the data is separated pretty well.

---

## Issues with SOM

In reality, SOM isn't used that much. It hasn't even got an implementation in sklearn.

There are a few reasons for this:

- The parameters are hard to tune (pick the wrong values of grid size, iterations, sigma, learning rate, and it might not look very good)
- The parameters are inconsistent (slightly different parameters yield completely different results)
- It relies on an iterative optimisation, which means it's hard to prove that results are optimal
- Deep learning allows far more complex and in some cases, easier to understand interpretations which use a similar technique
- Other, out of the box techniques seem to work a little better (see next)

---

name: manifold-learning

# Manifold Learning

A manifold is an N-dimensional baseline. A line is a 1-D manifold. A surface is a 2-D manifold. A figure of eight is not a manifold because it crosses itself.

Manifold learning is the attempt to learn the (possibly nonlinear) manifold of a dataset, whilst trying to separate distinct examples.

Many manifold learning methods build upon PCA. For example, kernel-PCA can be thought of a simple manifold learner. In k-PCA the data is transformed by a kernel *before* PCA is applied.

---

There are quite a few implementations of manifold learning. See the sklearn documentation.

But implementations differ by what they are trying to achieve.

- Some try to represent the raw data in fewer dimensions (like PCA)
- More popular implementations try to increase the distance between dissimilar instances (a form of clustering)

???

Some implementations, like ISOMap, LLE and Spectral Embedding continue down the PCA path of trying to represent the raw data in fewer dimensions, with the first dimension representing the highest amount of information. Definitely investigate these if this is your goal.

Goals might include trying to reduce the amount of data required to represent a domain, e.g. audio.

But one of the more popular implementations instead tries to increase the distance between dissimilar instances.

For classification, this is exactly what we want to do, so let's look at this in more detail.

---

# t-distributed Stochastic Neighbour Embedding (t-SNE)

Stochastic Neighbour Embedding is a similarity measure between a pair of instances in high dimensional space.

- First, in high-dimensional space, it generates a probability that two points are equal

???

For each instance a density measure is produced, which measures the distance between two points, for all points, and is then normalised by the distance to all points.

Effectively each measure is the probability of the two points being equal, in high dimensional space.

---

- Second, it transforms the data according to some weights
- Third, it performs another probability estimate
- Fourth, it alters the weights to make the probabilities more similar

???

Now, the second step is is to transform the data (by weights) into a lower number of dimensions.

Initially the weights are random and the projections are also random.

Another density measurement is performed in the low dimensional space, the same as before.

Now the goal is to attempt to make the probability of two high dimensional instances being the same equal to the probability of two low dimensional instances being the same.

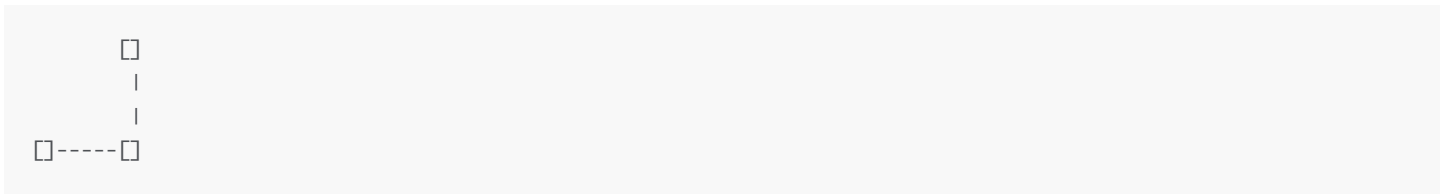
I.e. if they are separated in the high dimensional space, they should be separated in the low dimensional space.

Now all we have to do is calculate the gradient and perform a standard gradient descent algorithm to optimise the weights.

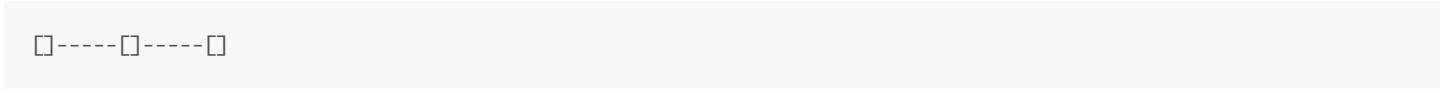
---

The issue with standard SNE is that when it projects the high dimension into the low dimension, it will retain the local separation, but warp the global separation.

The t-sne author describes it like this. Imagine you had three points. Two of the points are similar to a third, but not each other. In two dimensions the data would look like this:



When we map that into 1D, we can retain the local separation by doing this:



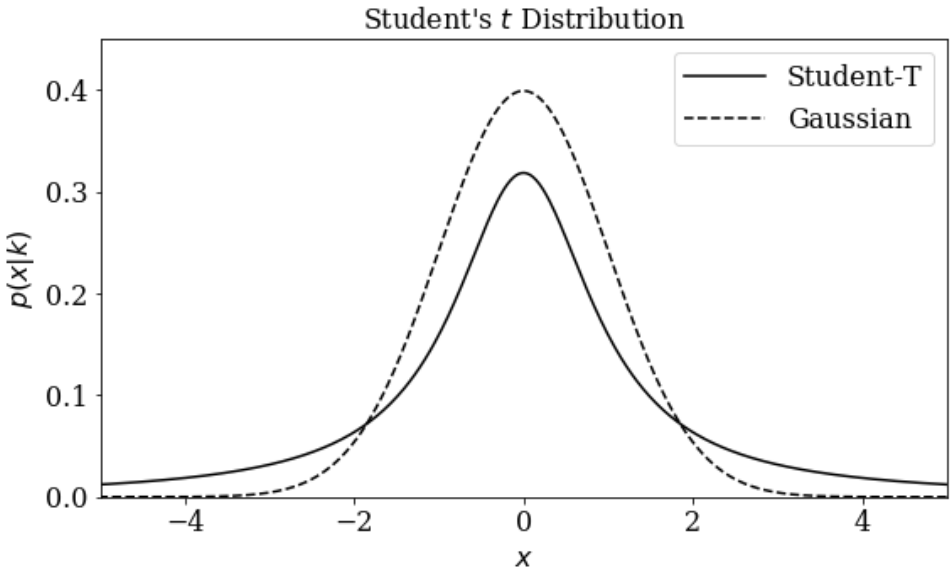
But notice that the global separation between the two dissimilar points has changed from $\sqrt{(2)}$ to $2$.

---

The author then renormalises the probabilities of similarity between two points according to a student-t distribution (a distribution with a fat tail).

As you can see, if two points were dissimilar, then they would be infinitely (well, very far) apart because of the near-divide-by-zero.

The student-t distribution allows points to be far enough to be clear, but not so far they're off the screen.




.center[ ]

---

So, to recap, it works by:

1. Project high dimensional space into low dimensional space using weights
2. Calculate the probability that two points are the same in high-dimensional space
3. Calculate the probability that two points are the same in low-dimensional space
4. Calculate the similarity between the two probability distributions (Kullback-Leibler divergence)
5. Calculate the gradient of the similarity and update the weights.
6. Repeat from step 1 until convergence.

That's a bit intense.

---

In short, it tries to keep instances that are far apart in high dimensional space far apart in low dimensional space. And it repeats until the statistics of the distances match.

Let's look at an example.

---

A selection from the 64-dimensional digits dataset

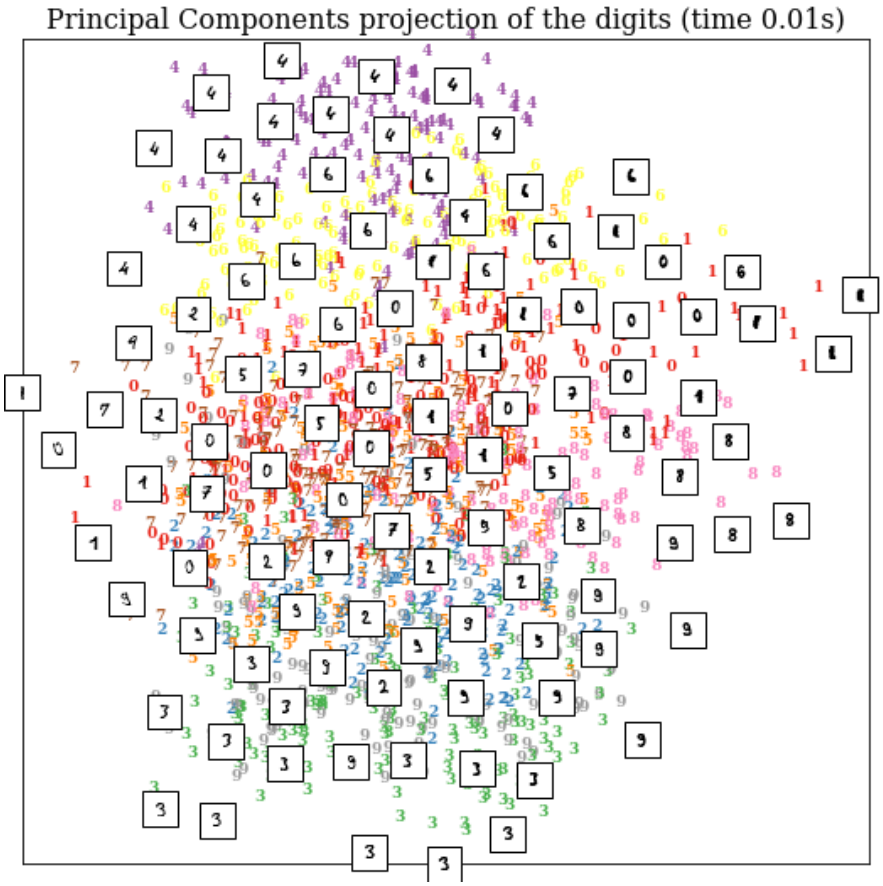
.left-column[ ]

.right-column[

If you haven't seen it before, this is the digits dataset. It is a series of $8 \times 8$ pixel images of handwritten numbers.

Let's perform PCA on this data.

]




.left-column[ ]

.right-column[

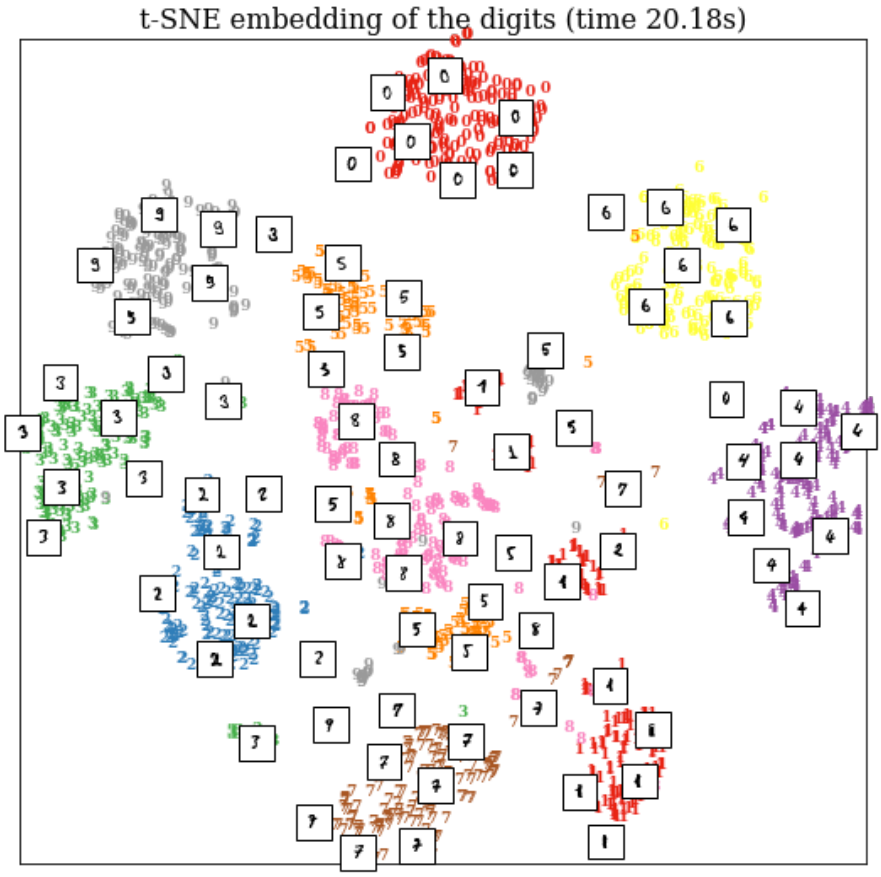
This is the result of the first two principal components of the digits dataset.

The different colours correspond to the diferent digits and an example has been plotted on top of each.

Notice the poor separation between classes.

This is because the data is highly nonlinear. PCA isn't able to capture the nonlinearities within the dataset.

]




.left-column[ ]

.right-column[

This is the result of the t-SNE algorithm.

Notice how disjoint all the classes appear.

You can see that the algorithm is trying hard to separate similar classes, even in this high 64-dimensional dataset. And on the whole it has done a good job.

There are still regions that haven't converged back together though, which indicates a slight issue with the chosen parameters.

]

---

## Pros/Cons with t-SNE

### Pros

- Works really well
- Simple to use (neglecting issues with the hyperparameters)

However, there are a number of caveats that you have to watch.

---

### Cons

- Distance between clusters is completely removed. Do not take any notice regarding the class separation.
- Inter-class variance is also trashed, because of the effects of being pushed out by other classes. As an example, look at the 0 class for PCA (which is linear, and preserves variance/spacing) and t-SNE. PCA's variance is much greater.
- Changing the parameters of t-SNE affects the results greatly
- Random noise often doesn't look random
- Topological data (data contained within data) is sometimes visible, sometimes not, depending on the parameters.
- Computationally expensive
- Is not consistent over time. Adding new/different data from the same problem with affect results.

---

### General Manifold Embedding Tips

- Always scale the data: Most embeddings are using euclidean distance as a similarity measure, so small scale features will naturally be closer than large scale features.
- Remove noise and/or useless features: Noise and/or useless features can trick the embeddings into thinking there is a manifold when there isn't.

---

# Others

- Linear discriminant analysis (finds manifolds that separate classes)
- Kernel PCA (apply a kernel before performing PCA - like kernel SVM)

---

class: center, middle

# Workshop: 40-dimensionality-reduction

---

---

name: sklearn-algorithms

# SKLearn algorithms

There's a lot of algorithms in SKLearn.

This is a fun chapter reviewing all (as many as possible) of them.

We've covered a lot over the past two days, but there are still more we haven't seen.

The objective here is to make you aware of them so if you do want to investigate further, you know where to look.

---

# What is Your Task?

Before we begin reviewing algorithms, we need to know where to start.

There are four standard data science tasks:

- Classification: what class does it belong to?
- Regression: what is it's value going to be?
- Clustering: I know there is structure, find it for me.
- Dimensionality Reduction: I've got too much data! And is there any structure?

It's interesting to note that the first two are typically supervised tasks. The latter are optionally unsupervised.

We've already discussed dimensionality reduction in depth earlier, so I won't cover it again here.

---

The next main decision is how much data do you have?

- 10: Compute a mean
- 100: Compute something linear
- 1000: Most classification algorithms
- 10,000: Something quite complex (e.g. kernel methods)
- 100,000+ observations: Stop and look at Deep Learning
- Otherwise: Continue investigating the data, improve features, impart knowledge, etc.

*Caveat: These numbers depend on the number of features*

???

If you have lots of labelled data, then stop now and look at deep learning. We'll cover that in the advanced class.

Otherwise continue trying to investigate the data, improve the features, impart knowledge and produce the best result you can.

Next we have a cute diagram showing some of the decision made during a discovery phase. This image isn't exhaustive, but gives a good idea of the process.

---

# Classification

Classification probably has the highest amount of diversity in terms of algorithms.

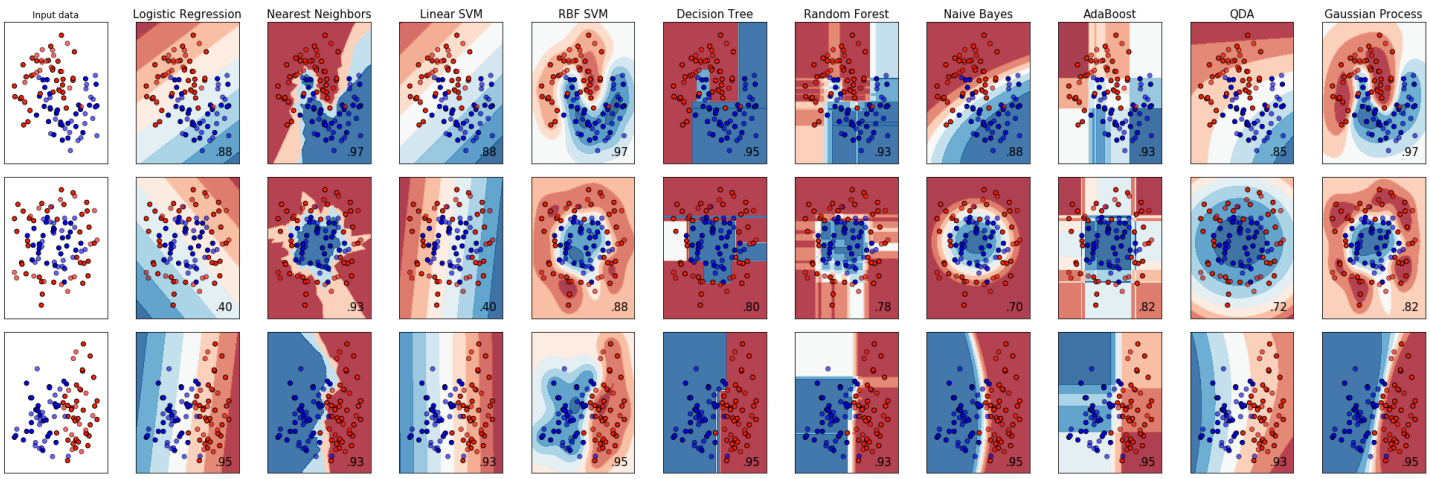
- Production code: Pick the simplest model that does the job
- Competitions: Go crazy, accuracy is all that matters.

So the first thing that you must decide is what is the purpose of you classification algorithm.

Once you have that then you can start to work your way through the algorithms.

*You will start to favour certain algorithms*

One of my favourites is a Random Forest, because it performs very well and is super easy to run.



There's even more that I haven't plotted. But this just about covers the main groups of algorithms.

The only one I have intentionally missed off is the neural network based classifiers. We'll be looking at them tomorrow.

We've already seen Logistic, SVMs, Forests and Naive Bayes, so I'm skipping them here.

---

## Boosting Algorithms

There are a class of classification and regression algorithms called boosting algorithms.

The general idea is:

- Start with a set of random classifiers
- For correct predictions, decrease weights
- For incorrect predictions, increase weights
- Combine the predictions and weights and regenerate new predictions
- Repeat until convergence

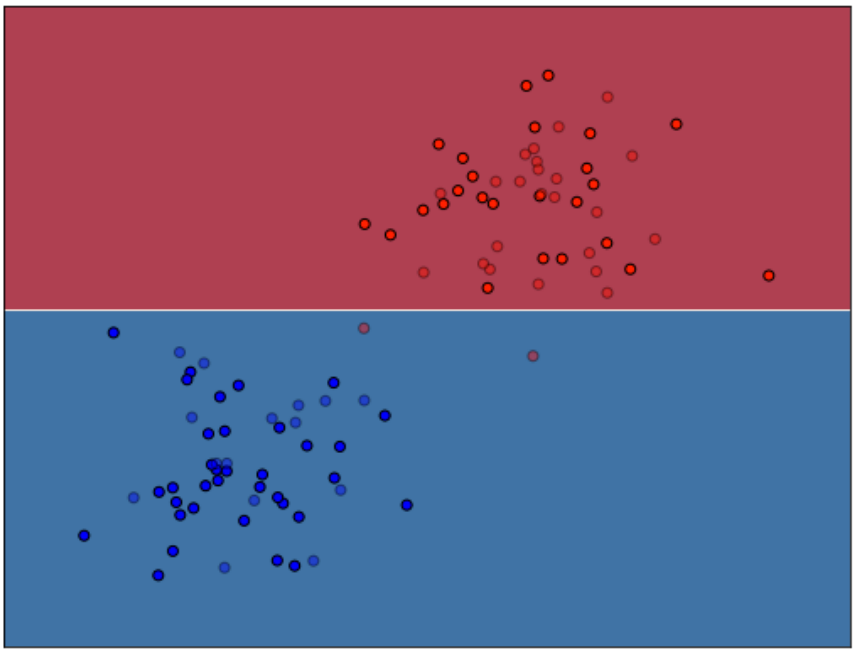
Over time the misclassifications obtain more weight, which means they have more and more infulence over the output.

Commonly the individual classifiers are decision trees, but that's not strictly necessary.

And because individual trees are used as the base classifiers this means that the model can overfit quite easily.

The main tuning parameters are the learning rate of the weights and the parameters of the underlying trees. E.g. how many, leaf size, etc. These would be tuned in the standard way.

---

.left-column[  ]

.right-column[

Here is the result of an a classification on some diagonal 2D data. The lighter samples represent the test data.

We can see that the algorithm has created a simple decision boundary across the middle of the data. This is testament to the underlying decision tree that was used to generate the classifications.

ADABOOST is generally considered to be one of the best "out-of-the-box" algorithms because the lack of need for tuning.

]

---

Pros:

- Hardly requires any tuning
- Can be used with many other types of algorithms
- A simple form of "ensemble" learning

Cons:

- Can overfit in some cases
- Sensitive to noise and outliers

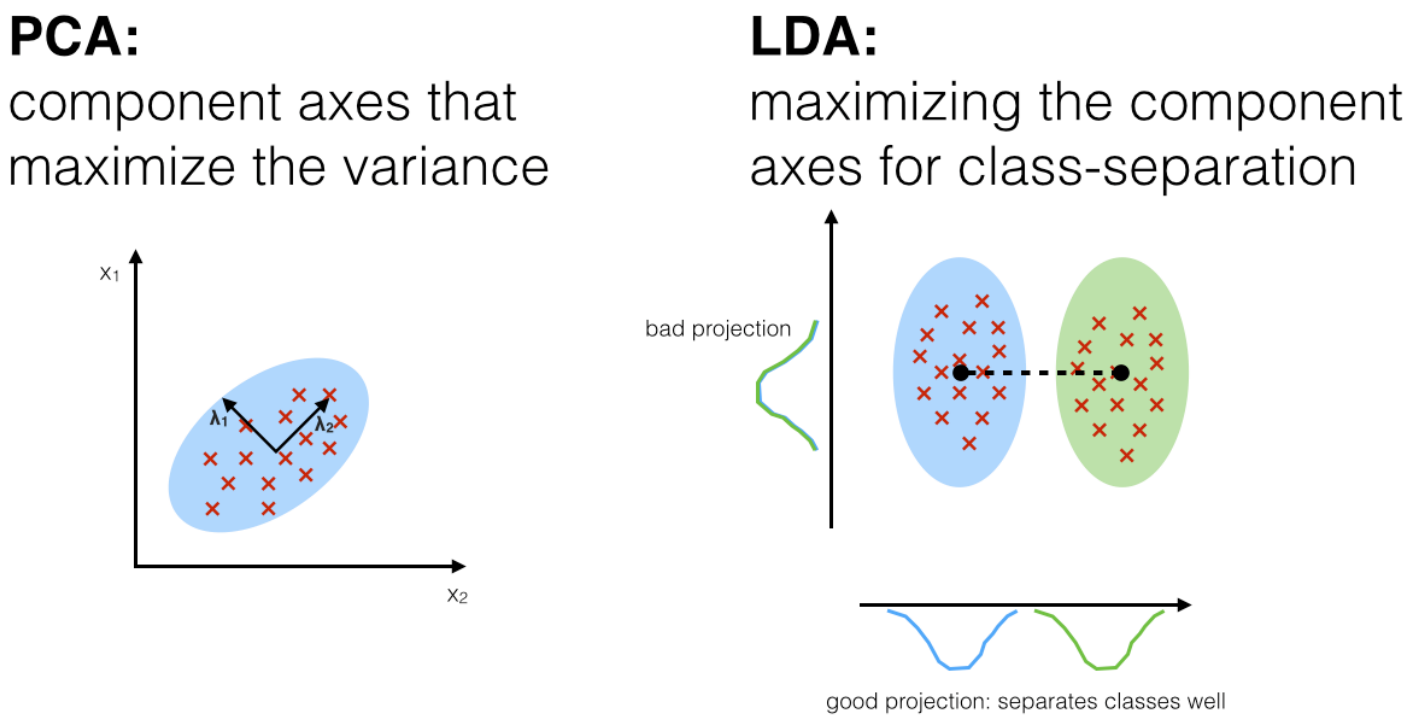
---

## Linear Discriminant and Quadratic Discriminant Analysis

This could have easily made it into the dimensionality reduction section, as it is also used for that.

This is very similar to PCA, except the goals are slightly different. PCA tries to remap the original data onto new dimensions, without losing information.

---

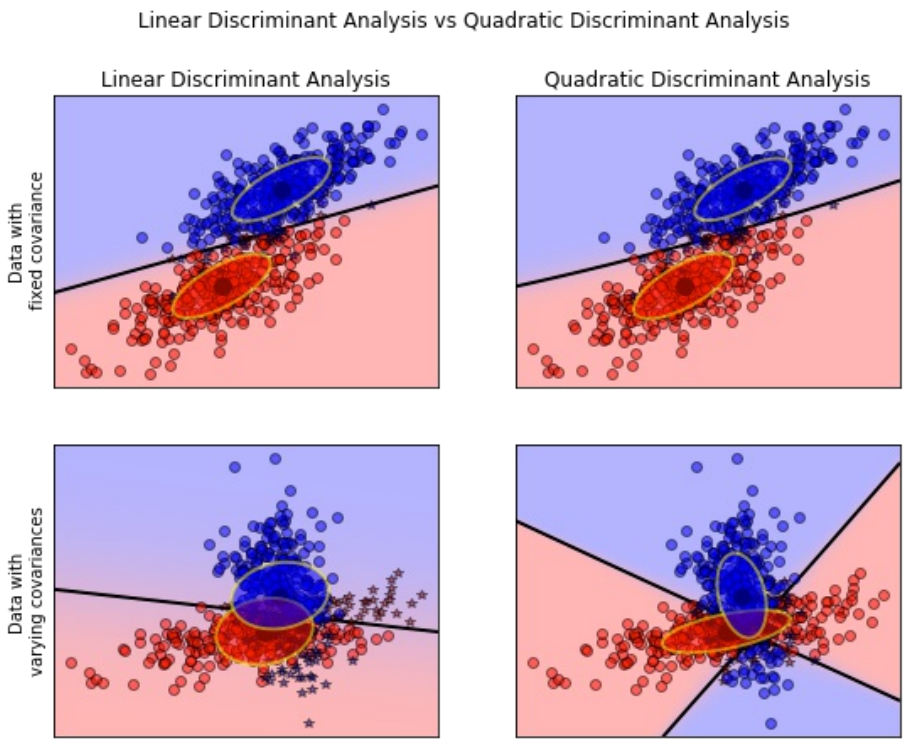



.bottom[ (With permission from Sebastian Raschka) ]

???

LDA and QDA attempts to pick the dimensions that are best able to separate the classes.

---




.center[ ]

LDA assumes equal covariances. So when there aren't equal variances, it doesn't perform well.

QDA allows for unequal covariances, which allows it to better fit more complex data.

The actual classification is performed by a bayesian fit of a gaussian.

---

## Classification through Gaussian Processes

- Allow you to simulate arbitrary functions as an infinite set of Gaussians
- For classification, very similar to Bayesian types
- Hugely flexible probabilistic classifier

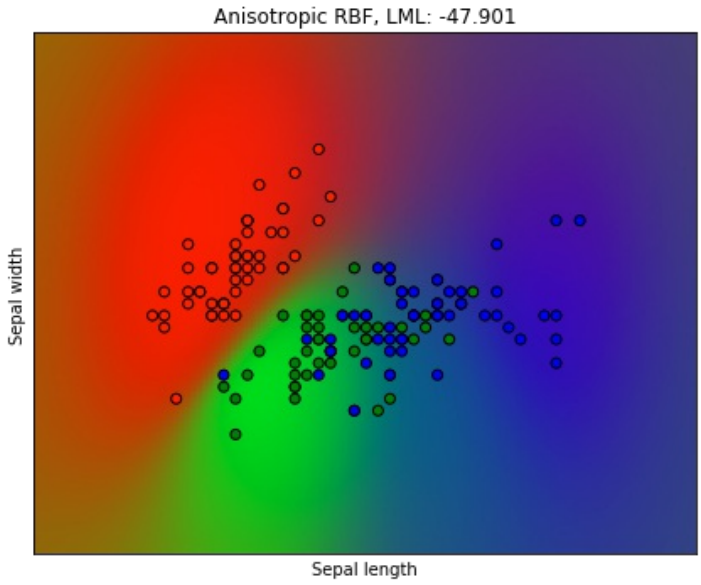
???

Gaussian Processes are pretty abstract mathematical concepts that allow you to simulate arbitrary functions as an infinite series of Gaussians.

But for classification, the result is very similar to Bayesian types of classification, with the added bonus of being able to choose your kernel with which to model your data.

Essentially this is a hugely flexible probabilistic classifier that attempts to model the densisty of your data in any number of dimensions.

---




.left-column[ ]

.right-column[

Here we're plotting the gaussian process classification estimates for iris dataset.

You can see the probability estimates for the various classes. Pretty incredible.

Pros:

- Probabilistic
- Can use different kernels/distributions, i.e. you can match them to your data

Cons:

- Slow

]

---

# Regression

There seems to be a dividing line between regression techniques, but it's hard to say specifically where that line is.

sklearn has a range of linear models, most of which you have already seen. But all of these tend to be quite primitive, in terms of the data that you're trying to predict.

---

In the real world, there are many regression tasks that use timeseries data.

For example:

- predicting sales
- stock market analysis
- anomaly detection

All of these have huge user bases, but tend to form their own disciplines and therefore there own libraries.

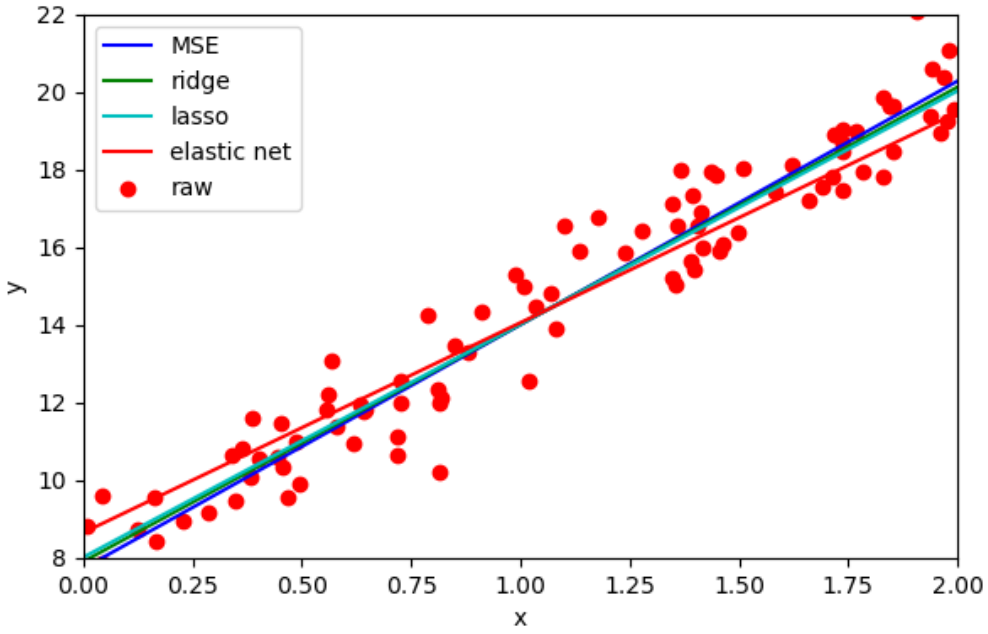
But this is how I would make the distinction:

- Time series analysis and prediction,
- Everything else.

---

## Linear Regression

We covered a lot of the linear regression algorithms in the first class, so I won't delve into them again here.




---

## Time Series Models

There is a whole subdiscipline regarding time series analysis.

Statsmodels has a lot of the basic models covered, autoregressive-moving-average types. Let's quickly cover them.

---

## (AR)(I)(MA) Models

Auto-Regressive means: "my next output depends on my previous input"

Moving-Average means: "my next output depends on the previous noise"

ARMA models are combinations of AR + MA models.

Integrated means: "my next output depends on a sum of the previous inputs"

There's a bit more to it, but that covers the general idea.

---

The key thing to remember is that (especially for ARMA models) the data must be stationary.

Stationary means that the data must:

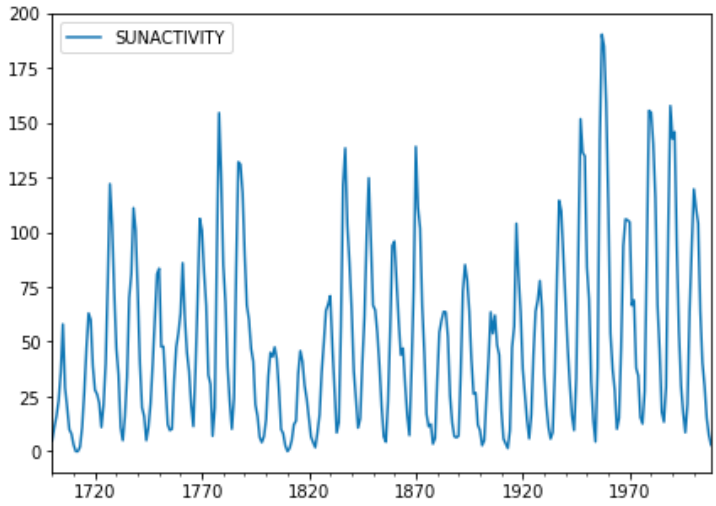
- Mean should be constant (not for ARIMA models)
- Variance should be constant
- Covariance should be constant (variance between samples should be constant - i.e. no variation in time)

---

As you can probably tell, this is quite a strict set of requirements. But it turns out that many processes can be massaged into this format.

If you can, the result is a very simple model, where the parameters probably tell you something important about the data of interest.

---




???

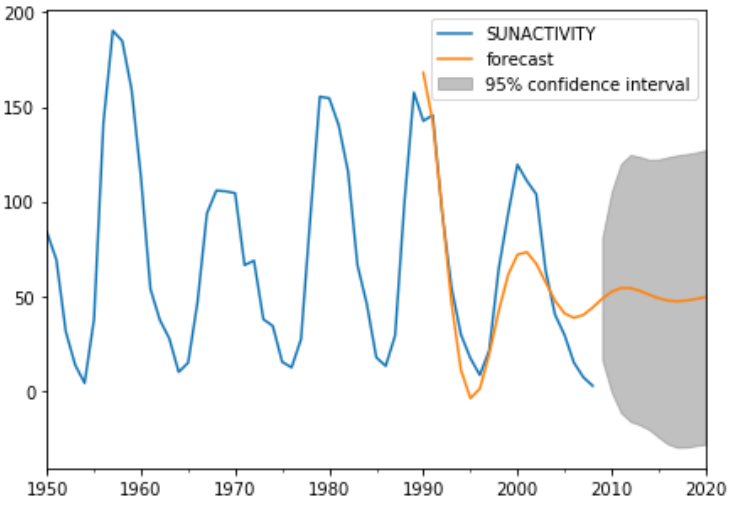
This is a sunspot activity dataset. It's interesting because it goes so far back in time.

But you can see that the variance is definitely not stationary, although the trend is.

So we'd expect to get the MA part about right, but all bets are off for accurately predicting the oscillation.

Finally, because there is little trend, there doesn't appear to be any reason to add integration to make it an ARMIA model.

---




.left-column[ ]

.right-column[

This is the result of an ARMA model for the sunspot data.

Like we predicted, the trend, the MA part, is about right. The prediction is good.

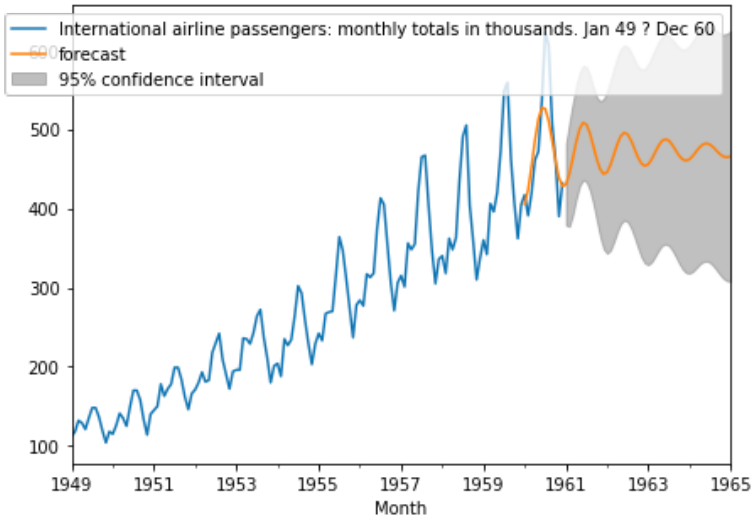
But it hasn't quite been able to predict the oscillation, the variation, because the variance isn't stationary.

]

---

.left-column[

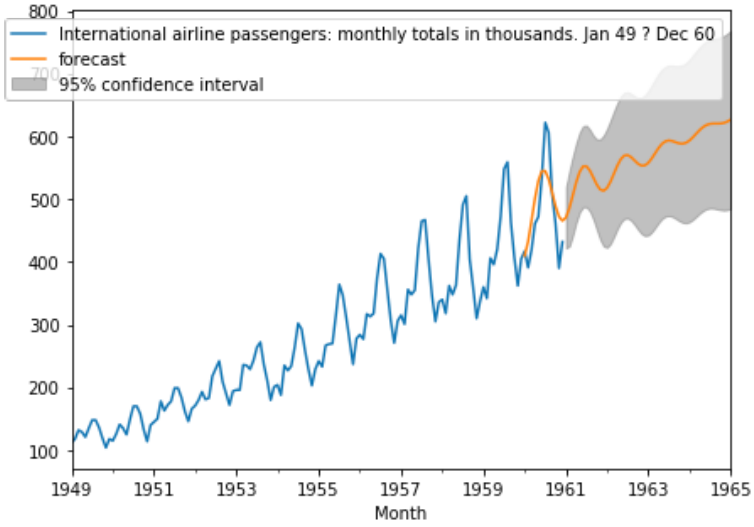
ARMA




]

.right-column[

ARIMA




]

???

This is the passenger airline data we saw previously. You can clearly see a trend in the data, so we must add an integration term for it to make sense (or detrend the data).

On the left, we have an ARMA model. On the right, an ARIMA.

You can see that with the ARIMA we're better able to track the trend, and results in a reasonable prediction.

---

## How do You Pick the Model Parameters?

The parameters that you enter into the ARMA/ARIMA models are called the "order" of the model.

They define how many previous samples to include when performing their sum.

- Typically the `I` component has values of 0-2
- For the `MA` and `AR` orders, we use two plots.

???

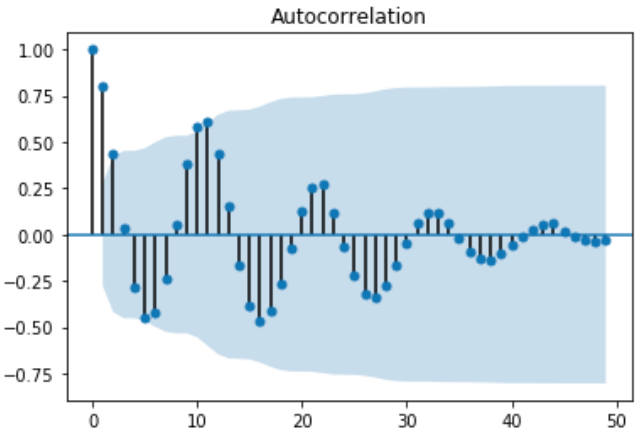
For example, the I component in the ARIMA model defines the number of time steps to include in the integration sum. 0 would mean that we add nothing to the next output. 1 would mean we add some proportion of previous input to the output (linearly increasing). 2 would mean we add two previous inputs to the current output (quadratic), etc.

Typically only values of 1 and 2 are used for the I component.

For AR and MA parameters, often two plots are used. The Autocorrelation plot and the Partial Autocorrelation plot.

---

.left-column[




]

.right-column[

]

???

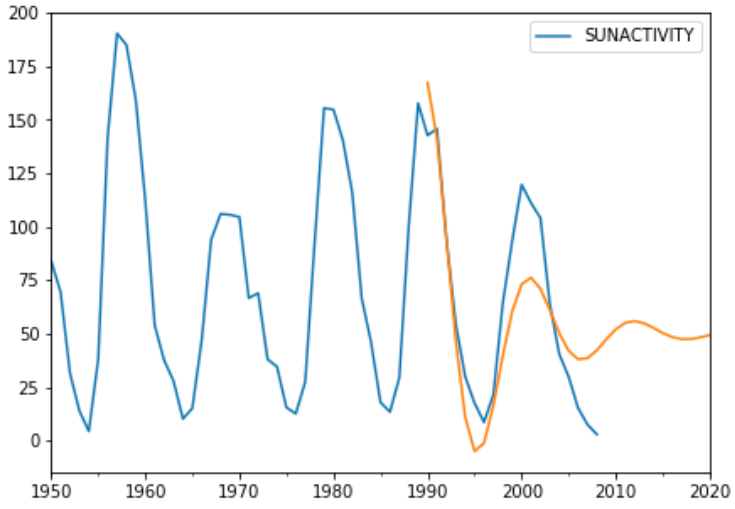
The autocorrelation plot is used to determine the MA order. The partial autocorrelation plot determines the AR order.

We can see that in the PACF plot on the right, there are clearly 2 or three components that are vastly larger than all other components. The shaded box represents a significance bounds.

The ACF plot on the left, however, has a couple of components at the start, but none of them are very significant compared to the rest.

This is indicating that a AR model of order 2 or 3 represents the data quite well, whereas the addition of a MA component won't add much.

---

.center[ 

]

This is the result of just fitting an AR model. It's about as good as previously, when we tried to fit a ARMA model.

We'll leave ARIMA models there for the moment. They are widely used and there is a lot of literature. Read up on it if you are interested.

---

## Hidden Markov Models

But the really impressive results come from Markov Chains, Hidden Markov Chains and Bayesian-inspired

Markov Chains.

Markov Chains are probabilistic processes that only depend on the state, not the whole history.

Complexity can be added with hidden states.

Arbitrarily complex Markov Chains can model even nonlinear timeseries. But their beauty lies in the fact that they are trained probabilistically. Meaning that they are robust against noise.

---

Now I was going to show a great example of predicting stock prices using a HMM, but unfortunately my (and it seems like everybody else's) source of data, Yahoo, just shut down it's API.

Yahoo used to provide a great API to get loads of historical market data, but after being sold to Verizon it appears to have gone poof!

If I haven't fixed this by the time you see it, apologies.

---

# Others

Some other advanced regressors that we have already seen in other guises:

- Bayesian Regression
- Gaussian Process regression
- Trees
- Boosting methods

---

# Clustering

We've also seen quite a few clustering algorithms already too.

We've learnt all about k-means and tree-based clustering (ward, agglomerative, birch).

But "you ain't seen nothing yet".

The next image is a nice comparison of lots of different classification algorithms together with some simulated datasets.

It gives you a good impression of what should be used when.

---

---

## Affinity Propagation

Affinity propagation is very cool because:

- You don't need to estimate the number of cluster (e.g. k-means)
- It doesn't enforce equal size clusters
- Only really one main parameter to tune the number of clusters

It works by having two matrices.

1. The first records which samples are the best "exemplars" of their cluster.
2. The second keeps track of which exemplar the current sample belongs to.
3. These two matrices are iteratively updated by continuously calculating how similar each instance is to each of the exemplars.

Over time similar instances will become more strongly tied with its exemplar. The algorithm stops when the clusters have converged.

---

We can see that the affinity propagation algorithm has chosen three clusters.

The lines point towards the chosen exemplar for each cluster.

---

### Cons

- It tends to require well scaled data
- Quite slow, don't use for lots of points
- Not very easy to understand the algorithm
- The parameter "preference" alters the results dramatically, although this can be tuned using the techniques we learnt in the previous clustering tutorial.

---

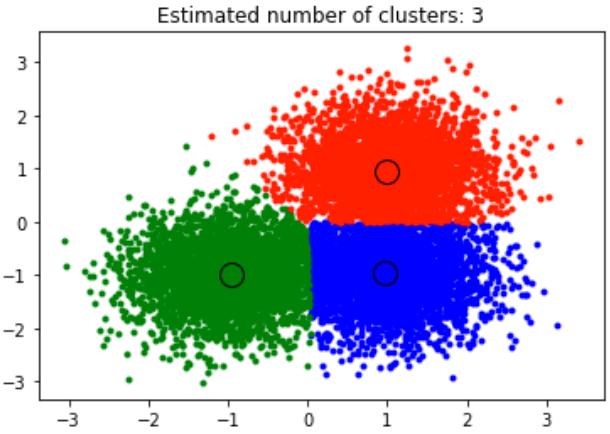
## Mean-Shift

The mean shift algorithm is quite simple and very similar to k-means and a gaussian mixture model.

It assumes that the data has been sampled from a probability density function. If there are dense regions then they correspond to a local maxima. It works in the following way:

1. For each point, define a window (kernel) around the point and compute the mean
2. Shift the window towards the mean and repeat until convergence.

---




.left-column[ ]

.right-column[

Pros:

- Easy to understand
- Works quite well with clusters that are poorly separated
- Don't need to specify the number of clusters
- Better than k-means because: use any kernel, don't need to choose number of clusters, hence no issues with bad initialisations.

Cons:

- Quite slow
- Data needs scaling and doesn't work well with clusters of different sizes and shapes
- Need to tune the window width
- Worse than k-means: is a lot slower, k-means is fast.

]

---

## Spectral

Spectral clustering is a little fancy.

First they calculate a similarity matrix (this is often called an affinity matrix).

Next they calculate the eigenvectors of the similarity matrix. A suitable number of eigenvectors are chosen to represent the data in a low-dimensional space.

Then k-Means clustering is performed on the low-dimensional space.

The crucial bit is automatically choosing the number of eigenvectors to use, then you also need to tune for the number of clusters in the k-means.

---

The huge benefit of this algorithm is the transformation of dimensions. It allows us to cluster really complex, non-parametric data.

But obviously you could do something like this manually if you transformed your data using PCA or some other dimensionality reduction technique.

---

.row.align-center[

]

This is an example of an image with some noise. In the image we've plotted several circles (i.e. similar parts of the image) and used spectral clustering to find those clusters.

???

You can see that the transformation has allowed us to handle really complex, multi-dimensional image data. Hence this technique is widely used in image fields, for example, for image segmentation.

---

## Pros:

- Works well even with very complex data (e.g. images)

- Check out the results of the comparison. It's possibly one of the best performing algorithms.

---

### Cons:

- Requires scaling
- Sensitive to disproportionate cluster sizes
- Still have to tune number of clusters
- A little unstable due to the transformation

---

## DBSCAN

Density-Based Spatial Clustering of Applications with Noise.

This is somewhat similar to k-NN or k-Means. The algorithm operates like so:

1. A max-similarity distance, `epsilon` in sklearn, defines how close instances have to be to be considered part of the cluster.
2. When there are `min_samples` considered to be close to each other, they form a *core sample* (a cluster).
3. It then continues to find instances that are close to this core sample, its *neighbours*.
4. Instances are only considered to be part of the *core sample* once the `min_samples` criterion has been reached.

???

Hence, this algorithm has the capability of identifying instances that are part of a cluster, but also not considered to be a *core* part of the cluster.

Furthermore, this also means that it can detect outliers for a particular cluster.

I get the impression than the authors wanted to include Densisty-Based and Clustering in the name of the algorithm, but needed a catchier acronym!

---

As you can imagine, whether samples are added to a cluster is highly dependent on `epsilon` and `min_samples`.

Often, `min_samples` is quite easy to choose after observing the data. But the fixed value for `epsilon` means that it doesn't cope well with clusters of varying variance.

---

- Makes a distintion between core samples and outliers
- Largely arbitrary, controlled by `epsilon`

???

From this exmaple we can see that the algorithm makes the useful distinction between instances that tend to be exemplars of the cluster, members of the cluster that aren't considered part of the core sample and outliers.

Although the distinction is largely due to the arbitrary choice of `epsilon`.

---

## Pros:

- Works well with non-parametric data
- Produces useful core, not-core, outlier distinctions that could be helpful in some use cases.
- Works reasonably well even with skewed, non-scaled data (assuming all clusters have a similar skew/scale
- Automatically chooses number of clusters

---

## Cons:

- Need to tune `epsilon` and, to a lesser extent, `min_samples`
- Doesn't work well with clusters of differing variance

---

## Gaussian Mixture

The last algorithm we're going to look at is also one of the most obvious.

Imagine you had a gaussian process that produced instances in any number of dimensions. How would you best cluster that?

By fitting gaussians of course!

---

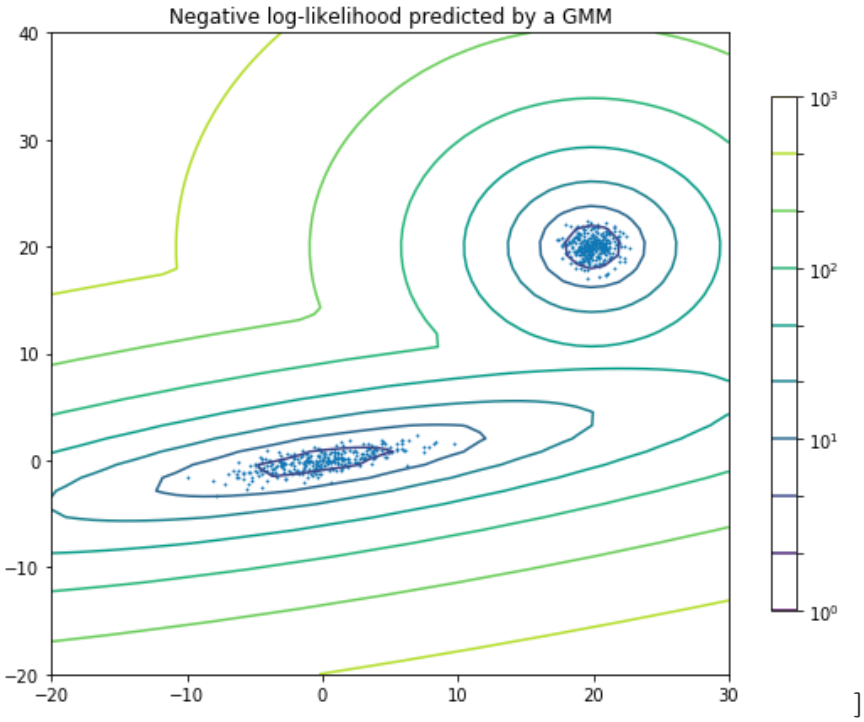
A gaussian mixture model is simply a mixture of gaussians. The goal is to select a number of gaussians and attempt to fit them to the data as best as possible.

The tricky part is that the algorithm is very flexible. It allows you to constrain the covariance to spherical, diagonal, tied, etc.

But the massive benefit is that you get a proper estimate of the probability of class membership because we are inherently fitting a probabilistic model to the data. And as we saw from the section on bayesian thinking, this can be very useful.

---

.left-column[




]

.right-column[

This is a pretty simple example, but it shows the results of fitting two 2D gaussians to the data.

The result produces two probabilities of an instance belonging to a particular class, based upon the shapes of the gaussians.

So, for example, if we were interested in calculating profit curves then we could introduce the probability of an instance belonging to a class into the profit curve calculation and provide better estimates for the final profit.

]

---

## Pros:

- Often better represents the underlying process that generates the data
- Produces reliable probability estimates
- Allows you to reduce cluster information to mean/variance measures.

---

### Cons:

- Doesn't work with non-gaussian data
- Need to choose number of gaussians

---

## Which Clustering Method?

- What are the characteristics of the data?

Probabilistic? Hierarchical? Non-Parametric?

Tip: Link the clustering method to the process that generated the data

???

Even though the process may be unsupervised, you often know what type of process generated the data.

Often you will know that the process was gaussian, or hierarchical or non-parametric.

If you don't know you can often research the subject and analyse the data to find out how the features were generated.

Once you have some definition of the process that generated the data, then you can pick one of the models that is best suited to that task.

For example, if the data is born out of a natural process, like heights or ages or something, then you would start with algorithms that inherently model data in a gaussian like way, like GMM or some sort of gaussian-based kernel density/affinity measure (k-means, mean shift, etc.).

---

- Model comparison

*You might find that certain algorithms do not work or cannot be justified*

Tip: Create a competition between models

???

Once you have established some algorithms to try, then you can create a competition based upon features of the algorithm and the results that they produce.

For example, if you have large numbers of instances, then that might be a critical factor when picking an algorithm. Slow algorithms wouldn't be feasible.

Once you have several algorithms to test, then you can compare and contrast the results both visually and numerically.

Visually by plotting results of interest. Numerically with some of the techniques we discussed in the

previous talk about clustering.

---

class: center, middle

# Workshop: 50-sklearn-algorithms

---

name: grand-challenge

# Grand Challenge Time!

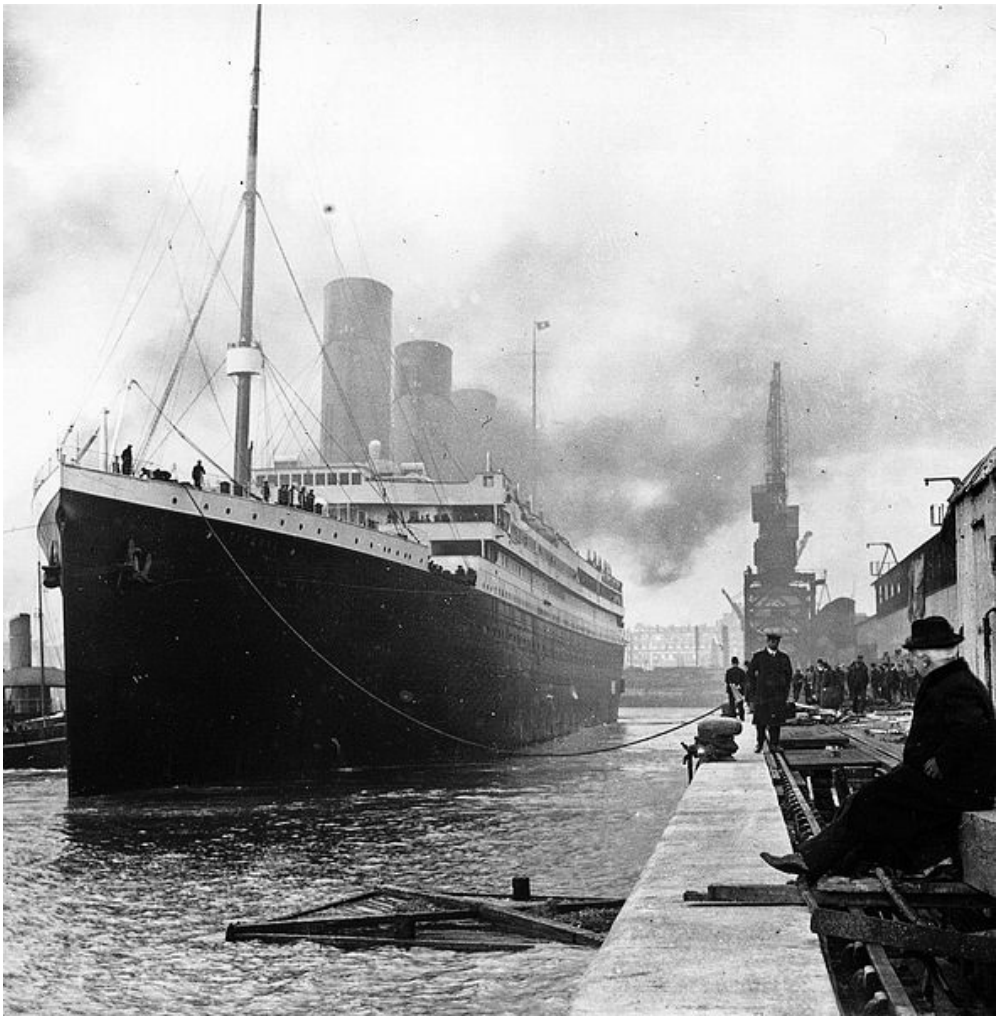
.left-column[

Today's challenge involves one of the most famous tragedies of the 20th century.

What makes it particularly interesting is the amount of skew within the data; the skews towards wealth, class and sex.

Your goal is to predict who survives the iconic maiden voyage of the RMS Titanic.

]

.right-column[  ]

---

This challenge was originally provided by Kaggle, the competitive data science website.

- We're all going to have our own mini-competition to see who can use the things we've learnt over the past few days.
- The winner gets bragging rights and will have to demonstrate how they achieved their result.

---

class: center, middle

## Workshop: 60-grand-challenge

---

# Review

- Numerical and visual model evaluation (critical)
- Dimensionality reduction
- A tour of machine learning algorithms

---

## End of Developer Data Science (Intermediate)

The advanced course covers: - Deep learning - Text - Ensemble methods

---

class: middle

## Thank You!

- If you need any help at all, get in touch.
- Know anyone that would like this training?

 @DrPhilWinder

 DrPhilWinder

 http://WinderResearch.com

 http://TrainingDataScience.com

.center[ Winder Research ]