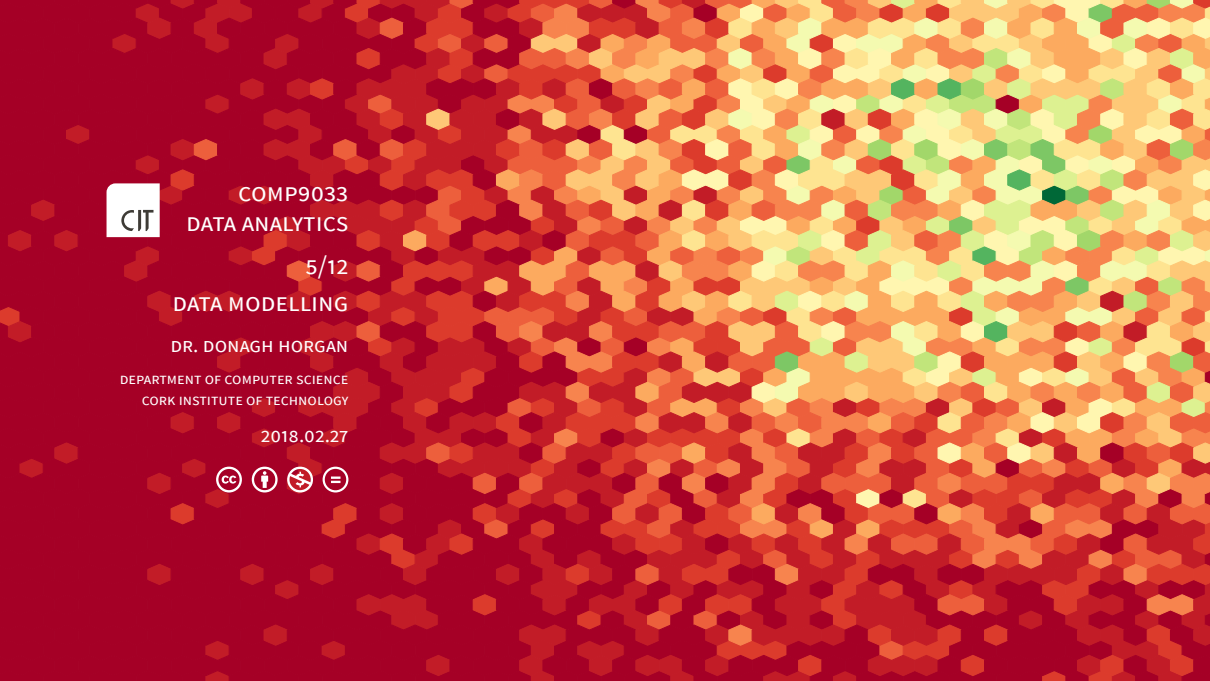


COMP9033
DATA ANALYTICS

5/12
DATA MODELLING

DR. DONAGH HORGAN

DEPARTMENT OF COMPUTER SCIENCE
CORK INSTITUTE OF TECHNOLOGY

2018.02.27

Overview

1. Cleaning data:
    - Duplicate data.
    - Missing data.
    - Outlying data.
    - Erroneous data.
    - How to deal with bad data.
2. Preprocessing data:
    - Converting categories to numbers.
    - Scaling data.
    - Feature generation.
    - Trend estimation.
3. Dimensionality reduction:
    - The curse of dimensionality.
    - Feature selection.
    - Principal component analysis.

1. Data modelling:
    - Rule-based models
    - Statistical models.
    - Machine learning.
2. Sources of model error:
    - Underfitting and overfitting.
    - Bias, variance and irreducible error.
    - The bias-variance trade off.
3. Cross validation:
    - Split size.
    - Exhaustive vs. non-exhaustive.
    - Cross validation techniques.
    - Stratification.
4. Model selection:
    - Choosing the optimal model.
    - Grid search.
    - Nested cross validation.

Data modelling

- Data modelling is a key step in the data analysis process:
  - It generally occurs *after* data sampling, exploration and cleaning/transformation.
  - In SEMMA and CRISP-DM, data modelling is the fourth step (see Lecture 01).
- A *model* is abstract representation of data.
- The key idea is that, if our data represents reality, and our model represents our data, then we can use our model to learn something that can be applied to new situations, *e.g.*
  - Model preferences and make recommendations/suggestions (Netflix/Spotify).
  - Model items that are commonly purchased together (Amazon/Tesco).
  - Model website quality and relevance (Google/Yahoo/Bing).

- Generally, the accuracy of a model depends on two factors:
  1. How well the model represents the data that we've gathered.
  2. How close that data is to real life.
- Consequently, if we use an appropriate modelling technique *and* we have good quality data[1], we can create models that are *very* accurate.
- Theoretically, if we had access to the whole population of data, and not just a limited sample, we could create a model with 100% accuracy.
- However, we are usually limited to modelling (relatively) small data samples, which often have quality issues, and so in practice we create models with a non-zero (though hopefully small) error rate.

[1] That is, the data itself is of good quality *and* we have sampled, cleaned and transformed it well.

- If we create a model with a large error rate, it can lead us to reach conclusions that do not generalise and we will make mistakes when interpreting new situations.
- However, models also offer many benefits:
  - They automate the process of interpreting data, and so can be more cost effective than manual (*i.e.* human) intervention
  - In some cases, they can be automatically updated when new data becomes available, *i.e.* they can be easy to maintain.
- Consequently, whether a given model should be used in a given situation depends on the cost, as well as the rate, of failure (*i.e.* risk versus return).

- Models are descriptions of data, and so *any* form of description is a valid model.
- However, in practice, only models that describe data well are useful.
- Typically, these are built by recognising patterns in, and/or determining the most important aspects of, the data.
- There are lots of ways to do this, but three commonly used techniques are:
  1. Rule-based modelling.
  2. Statistical modelling.
  3. Machine learning.

- Rule-based models summarise data using rules, *e.g.*
  - If the temperature is above 20 °C, ice cream sales will be high.
  - If an email contains the phrase “YOU WIN!!!”, it is spam.
  - If JVM heap usage grows to larger than 4 GB, there is a high probability of a service outage.
- Typically, this requires specific domain knowledge, *e.g.*
  - Business/marketing knowledge.
  - Spam/email user behaviour knowledge.
  - IT system administration experience.
- However, aside from this, rule-based modelling is straightforward to implement and easy to understand.

- Rule-based modelling can be useful in some situations, but is often too simplistic and is not robust to systematic change, *e.g.*
  - A threshold of 20 °C might be useful in Ireland, but may not work as well in another country because of differences in climate or culture.
  - Emails with the text “YOU’VE WON!!!” won’t be classified as spam without the implementation of an additional rule.
  - If the server memory is upgraded, then a real failure may now occur only when memory usage is in excess of 6 GB, and so the old rule will generate many false alarms.
- While it is possible to add or periodically adjust rules, this can be expensive (both in time and money) as human intervention is often required.

- Statistical models rely on the use of basic statistics to describe data, *e.g.*
  - If the temperature is 20% higher than average, then ice cream sales will be high.
  - If the *z*-score of JVM heap usage is greater than some threshold (*i.e.* a standard score test), then raise an alarm.
- Typically, statistics are computed directly from the data and so, often, no domain knowledge is required.
- They can also be automatically adapted to changes in behaviour, as new data becomes available.
- However, statistics are themselves quite simple descriptions of data, and often cannot capture more complex behaviour than deviations from a measure of central tendency or some other moment.

- *Machine learning* is a form of artificial intelligence concerning algorithms that allow computers to learn.
- It is similar to statistical modelling (the dividing line can be fuzzy), but typically *combines* statistical measures with algorithms and/or heuristics in order to maximise the information extracted from data.
- Like statistical modelling, it often does not require specific domain knowledge to work well and can be adapted to new situations as new data becomes available.
- Machine learning algorithms are generally classified as belonging to one of three categories:
    1. Supervised learning algorithms.
    2. Unsupervised learning algorithms.
    3. Semi-supervised learning algorithms.

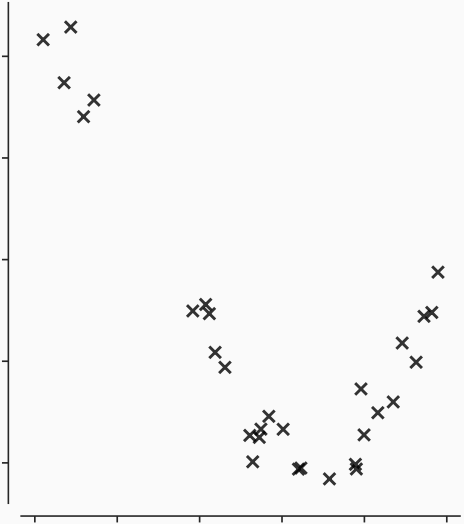
- Supervised learning algorithms extract information from data that provides some some form of feedback to indicate whether what is learned is “right” or “wrong”.
- This feedback can come in many forms, *e.g.* tags (yes/no, approved/rejected) or numerical measures (*e.g.* average error).
  - *Regression* algorithms require numeric feedback and can be used to predict new numeric values, given new explanatory variables.
  - *Classification* algorithms require discrete or categoric feedback and can be used to predict new category values, given new explanatory variables.
- Feedback is *required*, so usage is limited to data that can provide this.
- Typical examples include linear regression, nearest neighbours, decision tree classification.

- Unsupervised learning algorithms extract information from data without any form of feedback or ground truth.
- While they can be used to make predictions about new situations, the lack of feedback means that they usually aren't as effective as supervised learning algorithms.
- However, they are very useful in situations where no feedback exists.
- Typical examples include clustering and association rule mining.

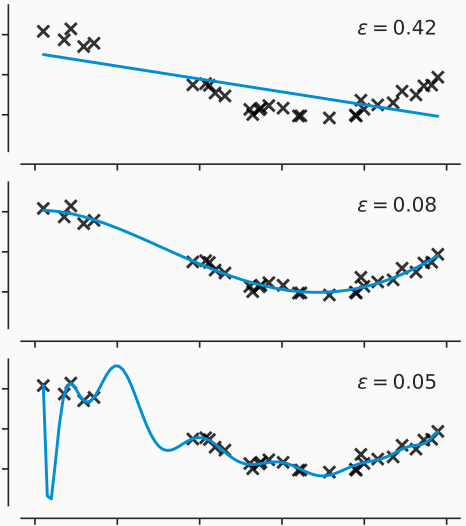
- Semi-supervised learning algorithms can *optionally* incorporate feedback to maximise learning.
- This balances the advantages of supervised and unsupervised learning.
- However, it is usually more difficult to design algorithms that optionally incorporate feedback, and so is less commonly used than other forms.

Sources of model error

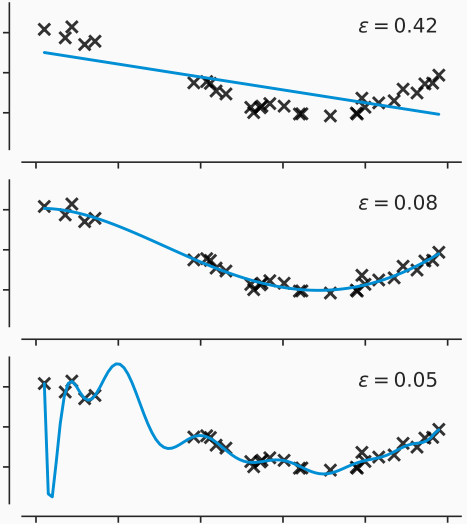
- Consider the problem of fitting a curve to the data shown to the right: what kind of curve should we fit?
- We could try a linear fit (i.e. $y = mx + c$) or we could try something more complicated.
- There are actually an *infinite* number of candidate curves we can fit.
- How can we know which is best?



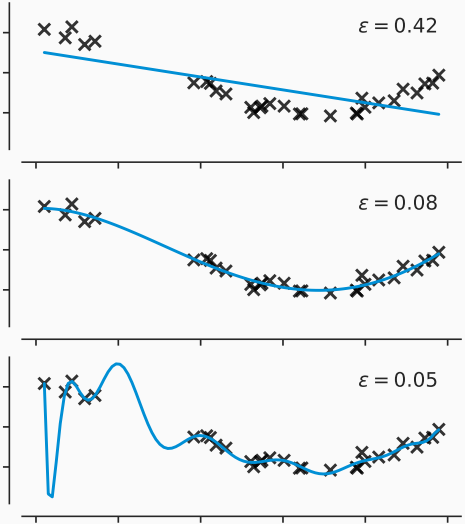
- The chart to the right shows three candidate fits for the data on the previous slide.
- The upper candidate is a simple straight line fit, and passes through none of the data points.
- The middle candidate is a more complex fit, passing through some of the data points and close to others.
- The lower candidate is a very complex line, and passes through nearly all of the data points.



- The upper candidate *underfits* the data, *i.e.* it does not capture enough detail.
- The lower candidate *overfits* the data, *i.e.* it captures too much detail.
- The middle candidate captures just the right amount of information from the data, enough that the fitted curve passes close to all of the points, but not so much that it contorts wildly in an attempt to fit through all of them.

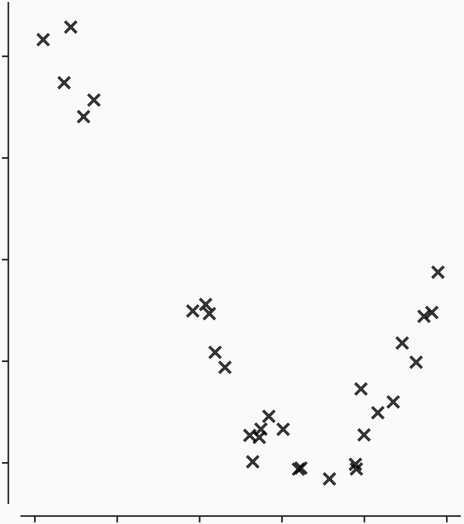


- Visually, the middle candidate appears to be the best fit.
- However, if we measure the average error (*i.e.* the distance) between the fitted lines and the data points, we will find that the lower candidate has the lowest error. Why?

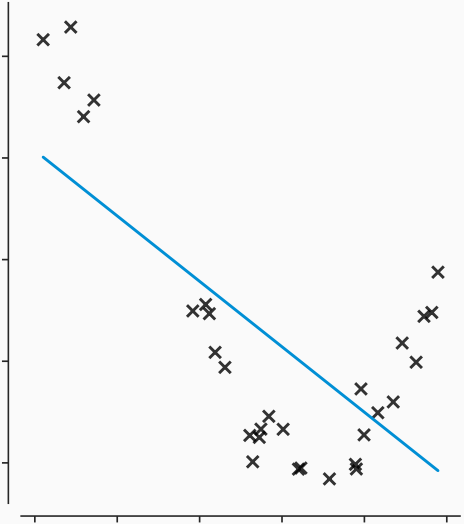


- The kinds of errors we encounter when building models are similar to the kinds of error we encounter when we try to fit curves.
- There are three specific varieties:
  1. Error from bias.
  2. Error from variance.
  3. Irreducible error.
- The total error for a model depends on all three, *i.e.*

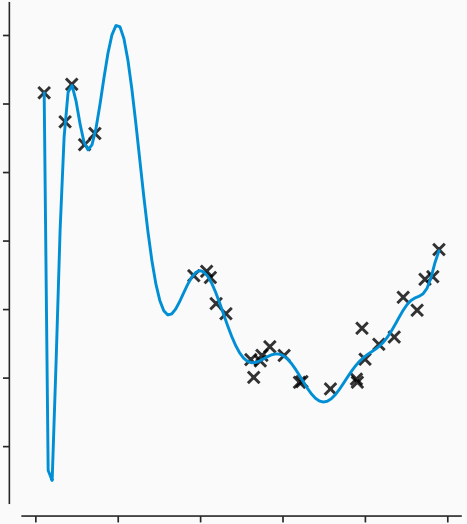
$$\epsilon_{total} \sim \epsilon_{bias} + \epsilon_{variance} + \epsilon_{irreducible}. \quad (5.1)$$



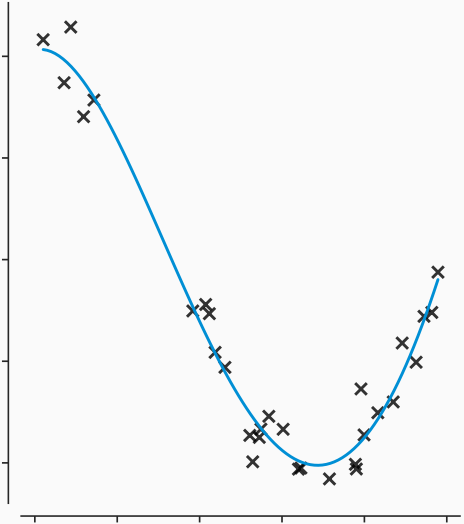
- Bias error is caused by building a model that is not specific enough to the training data.
- It is similar to the concept of underfitting in our curve fitting example.
- When a model suffers from bias error, it is usually wrong *on average*.



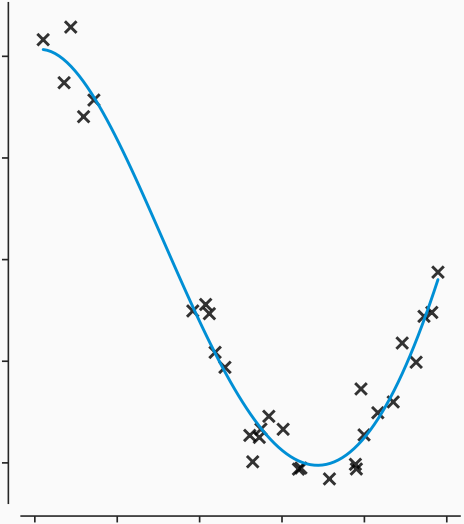
- Variance error is caused by building a model that is too specific to the training data.
- It is similar to overfitting in our curve fitting example.
- When a model suffers from variance error, small deviations from the training data cause large errors in the output.
- Consequently, the model is often wrong on new data.



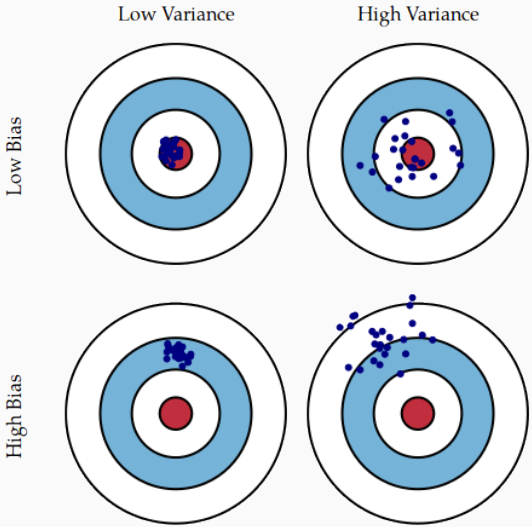
- Irreducible error is caused by the difference between the data samples and reality and has multiple causes, *e.g.*
  - Use of a poor sampling technique.
  - Measurement error when collecting data.
  - Naturally occurring noise.
- In our curve fitting example, irreducible error (due to noise) causes some of the data points not to lie on the correct fit line.



- Typically, we have no control over irreducible error as it depends on the quality of the data.
- However, in many situations, our data quality will be sufficiently high that the irreducible error is not significantly large.

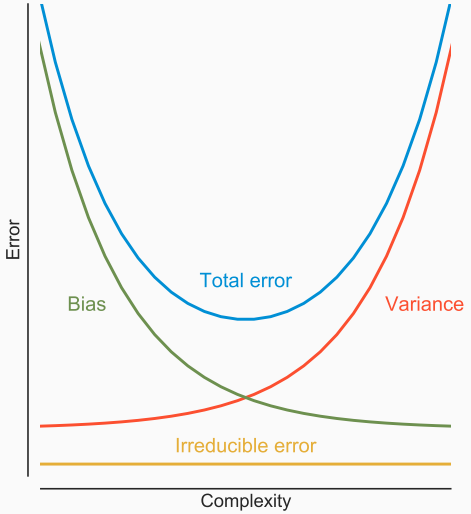


- Bias and variance error are actually statistical concepts:
  - Bias is a measure of the *central tendency* of the model error.
  - Variance is a measure of the *dispersion* of the model error.
- A good way to think about this is that bias measures the typical error of a model, while variance measures the variation in error from sample to sample.
- Ideally, we want to build models with both low bias and low variance.

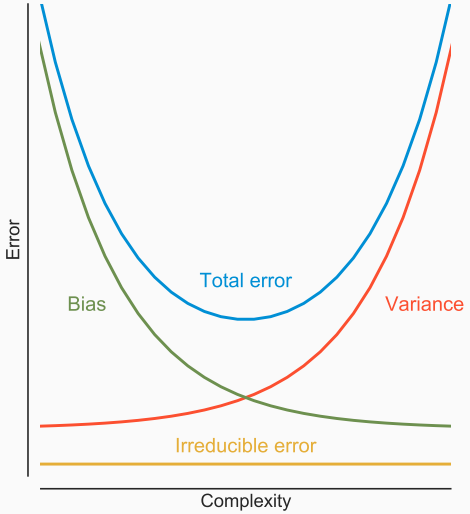


Credit: Scott Fortmann-Roe

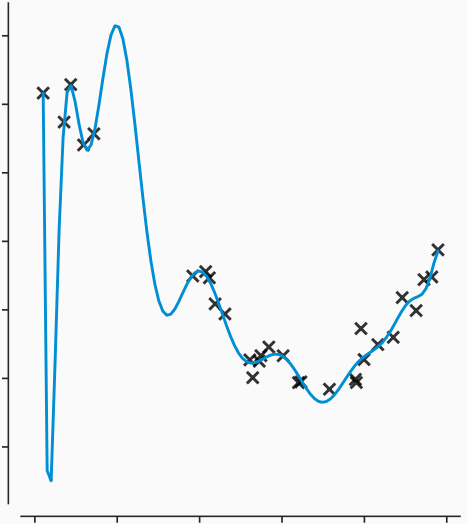
- As the complexity of our models increases,
  - Bias error tends to decrease.
  - Variance error tends to increase.
  - Irreducible error is unaffected.
- For instance, in our curve fitting example earlier:
  - The simplest model had high bias.
  - The most complex model had high variance.



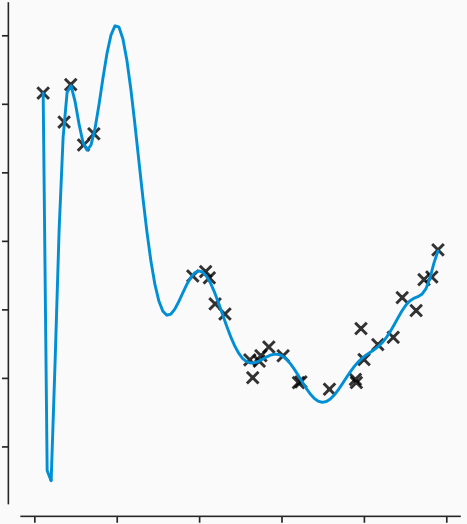
- This behaviour leads to the *bias-variance trade off*:
  - Models that are too simple have high total error due to high bias error.
  - Models that are too complicated have high total error due to high variance error.
  - Models that have a *sufficient* amount of complexity can minimise the total error by balancing the bias and variance errors.



- Ideally, we want to minimise the total error.
- However, measuring the total error can be problematic:
  - If a model is overfitted, its error on its training data is usually low.
  - For instance, the model on the right has low average error because the line fits most of the data.
  - Without new data, we cannot measure the error from variance, and so cannot measure the total error.



- One solution is to split our data into two sets, one for training, the other for testing.
- Because the model is built using the training data, the test data is “new” and can be used to estimate the total error.
- This technique is known as hold-out validation and is an example of a process known as cross validation.



Cross validation

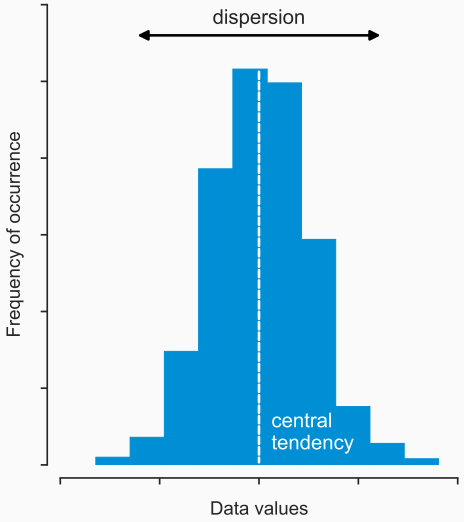
- Cross validation is a procedure for *estimating* the total error resulting from the application of a model to new data.
- It can be used to assess the quality of models or as an aid when optimising their *hyperparameters*.
- In general, cross validation relies on the splitting of data sets into two subsets — training and test:
  - The model is built using the training data.
  - The accuracy of the model is then evaluated using the test data.
- Consequently, when applying cross validation, we must consider two factors:
  1. The amount of data in each set, *i.e.* the split size.
  2. How exhaustive the validation will be.

- Choosing a split size is an important consideration:
  - Splitting the data means that there will be less data available for training, which *might* limit what we can learn[2].
  - However, if we don't have enough (or any!) test data, then we can't be confident that our model will generalise, regardless of how much data it was trained on.
- Choosing a split size is often subjective:
  - The ratios 50:50, 80:20 and 90:10 (training-to-test) are commonly used in practice.
  - However, it's also important to ensure that there is enough data in the test set for it to be meaningful.

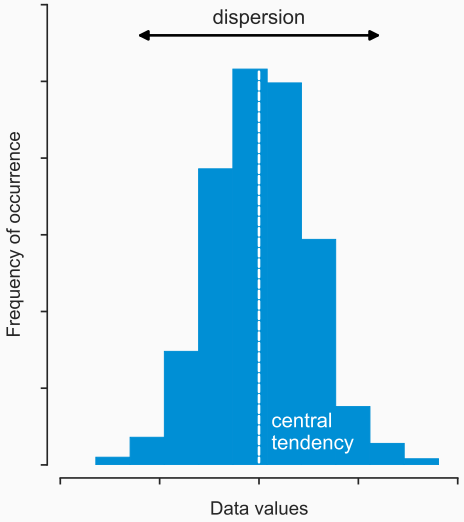
[2] The smaller the sample, the worse it tends to reflect the population it was drawn from.

- Another important choice is whether the validation should be exhaustive or non-exhaustive.
- Exhaustive validation involves the use of every possible combination of training and test sets that can be formed from the original data.
  - Essentially, this is a brute force approach.
  - Can be computationally demanding.
  - However, the resulting estimate of model error is typically better than a non-exhaustive validation.
- Non-exhaustive validation involves the validation of a limited, typically small, number of combinations of training and test sets.
  - Typically requires less computation than an exhaustive approach.
  - The estimate of model error may not be as reliable as the number of combinations used is smaller.

- Each iteration of cross validation produces a single error estimate.
- We can treat this collection of estimates as a data sample:
  - The central tendency of the error estimates should represent the typical error value.
  - The dispersion of the error estimates should represent the variation of the error from sample to sample.



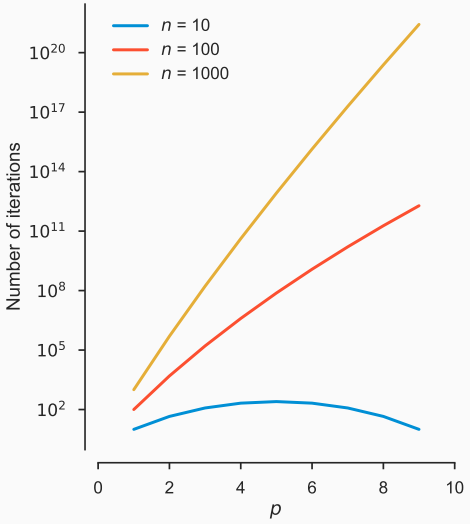
- However, if the number of iterations is small, then the number of points in our error estimate sample will also be small.
- The smaller the sample, the less likely it is to represent the population it was drawn from, *i.e.* the true error of the model.



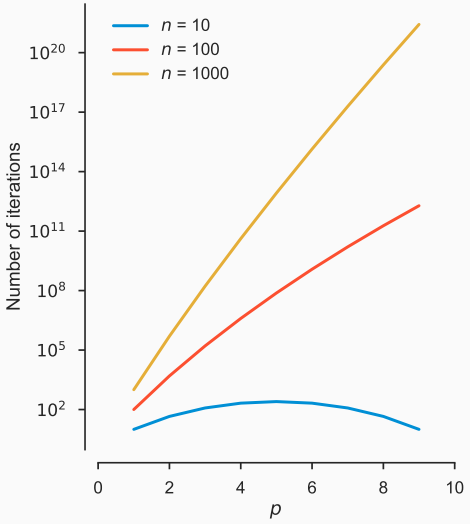
- Leave one out (LOO) cross validation is an exhaustive validation technique.
- For a data set with n samples, LOO cross validation works as follows:
  1. Select *one* sample to be the testing set and let the remainder (i.e. $n - 1$ samples) be the training set.
  2. Build a model using the training set and compute the model error using the single test sample.
  3. Repeat steps 1 and 2 for every possible single sample testing set, so that you have n individual model error measurements.
  4. Average these measurements to estimate the total error (*i.e.* the central tendency).
- LOO cross validation requires just n iterations in total.

- Leave p out (LPO) cross validation is a generalisation of LOO to p samples.
- For a data set with n samples, LPO cross validation works as follows:
  1. Select p samples to be the testing set and let the remainder (i.e. $n - p$ samples) be the training set.
  2. Build a model using the training set and compute the model error using the test set.
  3. Repeat steps 1 and 2 for every possible test set with p samples, and record the error measured in each case.
  4. Average the recorded errors to estimate the total error.
- LPO cross validation requires exactly $N = \binom{n}{p}$ iterations.

- As p becomes larger, a larger sample of error estimates is produced, and so the final estimate should better reflect the true error of the model.
- However, if p is very large, then $n - p$ (the training set) may become too small, and the quality of model produced may become worse.



- Depending on the values of n and p, LPO can be computationally demanding!
- Therefore, there is a trade off between gathering a smaller set of error estimates of higher quality model or a larger set of estimates of a lower quality model.



- Hold out cross validation is a non-exhaustive validation technique.
- For a data set with n samples, hold out cross validation works as follows:
  1. Randomly select p samples to be the testing set and let the remainder (i.e. $n - p$ samples) be the training set.
  2. Build a model using the training set and compute the model error using the test set.
- Unlike LPO/LOO, hold out validation requires just *one* iteration.
- However, because the error is only estimated once (*i.e.* we have just one sample), it can be a much less reliable measure of the true error.

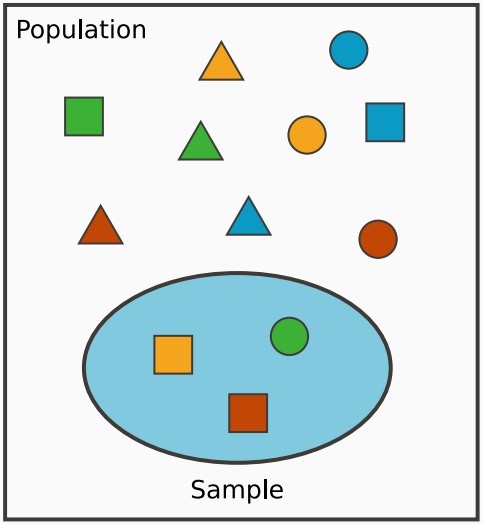
- K-fold cross validation is a further non-exhaustive validation technique.
- For a data set with n samples, K-fold cross validation works as follows:
  1. Randomly partition the data into K distinct sets of equal size.
  2. Choose one of the sets to be the test set and merge the remaining $K - 1$ sets to form the training set.
  3. Build a model using the training set and compute the model error using the test set.
  4. Repeat steps 2 and 3 for each of the K sets, recording the error in each case.
  5. Average the recorded errors to estimate the total error.
- K-fold cross validation requires just K iterations.

- As K-fold validation requires just K iterations, it is less computationally demanding than LPO and LOO validation[3].
- The larger the value of K, the more error estimates are produced.
- However, as K becomes larger, the size of the test sets becomes smaller, which can lead to less accurate individual error estimates.
- Therefore, there is a trade off between gathering a smaller set of higher quality estimates or a larger set of lower quality estimates.
  - There is no golden rule, but $K = 5$ and $K = 10$ are commonly used values.
  - Choosing K so that the size of the test set is reasonable is a good rule of thumb.

[3] K-fold is equivalent to LOO validation when $K = n$.

- Shuffle and split validation is a further non-exhaustive cross validation technique.
- For a data set with n samples, it works as follows:
  1. Randomly select p samples to be the test set and let the remainder of the samples be the training set.
  2. Build a model using the training set and compute the model error using the test set.
  3. Repeat these steps as many times as desired.
  4. Average the errors to estimate the total error.
- The number of iterations in shuffle and split validation is *arbitrary*.

- Unlike other techniques, shuffle and split allows precise control over the number of iterations *and* the train-test split ratio.
- However, there is no guarantee that the same samples won't get re-used in the training and test sets across iterations:
  - Some sets may over-represent some outcomes.
  - Some sets may under-represent some outcomes.



- When dealing with categoric data, it is important to ensure that outcomes are represented equally across splits.
- For instance, if we were building a spam detection model, and by chance most of our genuine spam emails ended up in the training set, then there would be very few genuine spam emails in our test set and our resulting estimate of model accuracy would be poor.
- However, if we could choose splits from the data so that the proportion of outcomes in each split was equal, then the test set would “look like” our training set and our resulting error estimate would be more representative.
- This technique is known as *stratification* and can give better error estimates in many cases.

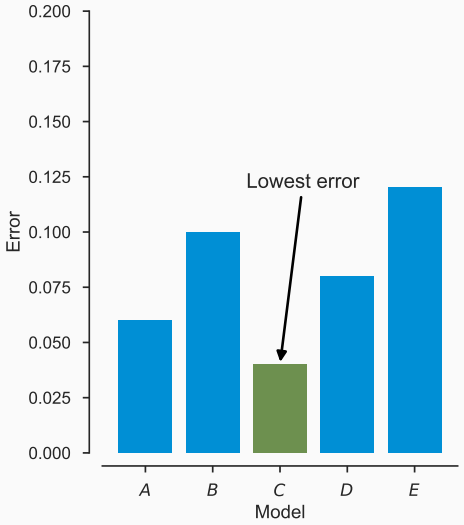
- Depending on the number of iterations involved in the cross validation technique, you may build more than one model when estimating error, *e.g.*
  - LOO: n models.
  - LPO: $\binom{n}{p}$ models.
  - Hold out: 1 model.
  - K-fold: K models.
  - Shuffle and split: arbitrary number of models.
- In cases where more than one model is built, it is typical to build a final model using *all* of the available data.
- While the error of this final model cannot be estimated, it is generally *assumed* to be close to the cross validation estimate[4].

[4] If your validation is rigorous enough, this *should* be true.

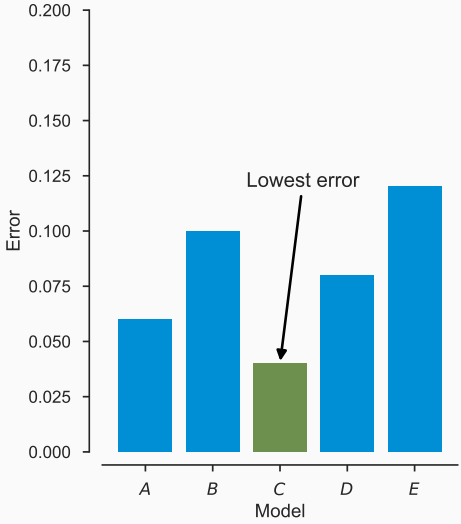
Model selection

- In many data analysis problems, we have a lot of choices to make, *e.g.*
  - Whether to include or exclude a given feature.
  - What value to assign to a hyperparameter.
  - Whether to use one machine learning algorithm over another.
- These choices often appear to be *subjective*, *i.e.* there is no clear answer as to which is best.
- Ideally, there would be some way to determine whether one combination of choices is better than another!

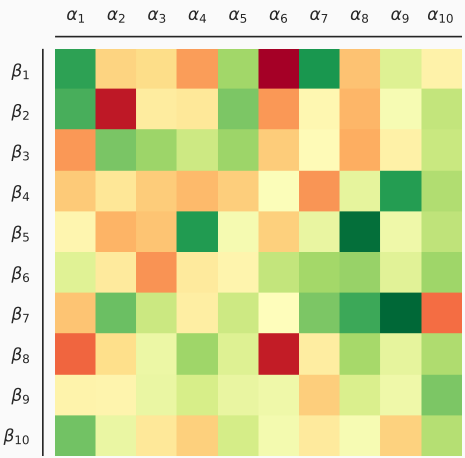
- *Model selection* is the process of choosing the best model from a set of given candidates.
- The basic procedure is quite simple:
  1. Build a set a candidate models, each representing a distinct set of parameter choices.
  2. Measure the error of each model using cross validation.
  3. Select the model with the lowest error.



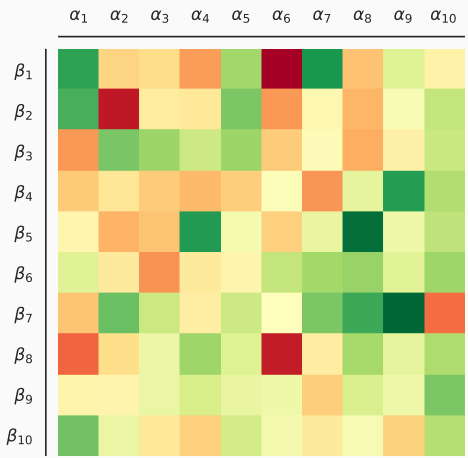
- For instance, in the chart to the right, the optimal parameter choice is the one that was used to build model *C*.



- One commonly used model selection technique is a *grid search*.
  - Sets of values are specified for each parameter.
  - Parameter sets are then formed based on combinations of these values.
- The chart to the right illustrates a grid search over every unique combination of α_i and β_j from the parameter ranges $\{\alpha_1, \alpha_2, \ldots, \alpha_{10}\}$ and $\{\beta_1, \beta_2, \ldots, \beta_{10}\}$.



- Grid search is an exhaustive algorithm — every possible combination of parameters is evaluated.
- As the number of individual parameter values increases, the time taken for model selection can increase significantly!
- Parameter range values should be chosen with care, where possible.



- In model selection, cross validation is used to select the best model from a given set of candidates — but we still need to determine the error of the selected model.
- The solution to this is to use a *nested* cross validation:
  1. Use model selection with an *inner* cross validation to select the best model from a set of candidates.
  2. Use an *outer* cross validation to measure the error of the selected model on the whole data set.
- Using nested cross validation ensures that we generate the most accurate error measure possible, given the amount of data that is available.

4.7 / EXAMPLE: NESTED CROSS VALIDATION IN SCIKIT-LEARN

```
from sklearn.model_selection import GridSearchCV, KFold, cross_val_predict

X = ...           # Explanatory variable(s) / feature(s)
y = ...           # Target variable
algorithm = ...   # The algorithm to train the model with

param_grid = {
    'a': [0, 1, 2, 3, 4, 5],  # First parameter
    'b': [True, False],       # Second parameter
    ...                       # Other parameters
}

# Select the optimal model
inner_cv = KFold(n_splits=5, shuffle=True)  # 5-fold CV, can be whatever you choose

model = GridSearchCV(algorithm, param_grid=param_grid, cv=inner_cv)
model.fit(X, y)

# Compute the model error
outer_cv = KFold(n_splits=5, shuffle=True)  # 5-fold CV, can be different to 'inner_cv'
y_pred = cross_val_predict(model, X, y, cv=outer_cv)

error = measure_error(y, y_pred)  # 'measure_error' depends on the model
```

Summary

- Data modelling:
  - Model types: rules, statistics and machine learning.
  - Sources of error: bias, variance and irreducible.
  - Cross validation: split size, exhaustive vs. non-exhaustive.
  - Model selection: grid search, nested cross validation.
- This week's lab:
  - Build a rule-based spam classification model.
  - Measure the model error.
  - Build many spam classification models and select the best one!
- Next week: linear regression!

1. Hastie et al. *The elements of statistical learning: data mining, inference and prediction.* 2nd edition, February 2009. (stanford.io/2i1T6fN)
2. Ullman et al. *Mining of massive data sets.* Cambridge University Press, 2014. (stanford.io/1qtgAYh)