Predictive models

DATA1002 Lecture 10A

Alan Fekete

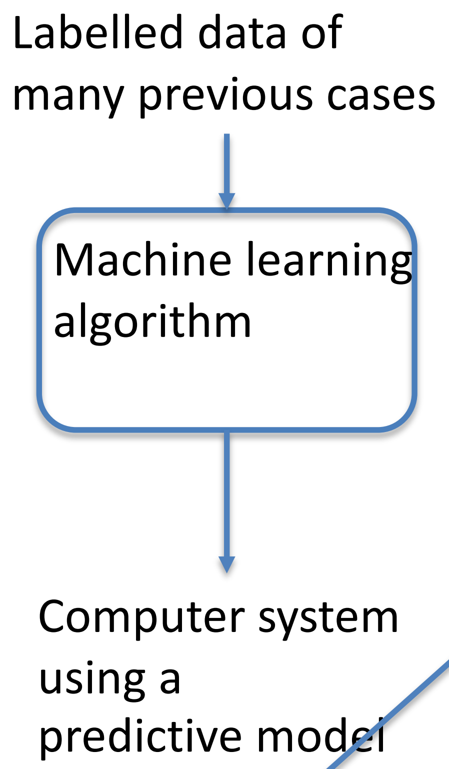
University of Sydney

Agenda

- *Overview*
- Features
- Evaluation
- Linear Regression
- Extensions of linear regression
- K-nearest-neighbour regression
- Decision-tree classification
- “Logistic regression” classification

Use ML to train a predictive model

- Typical training

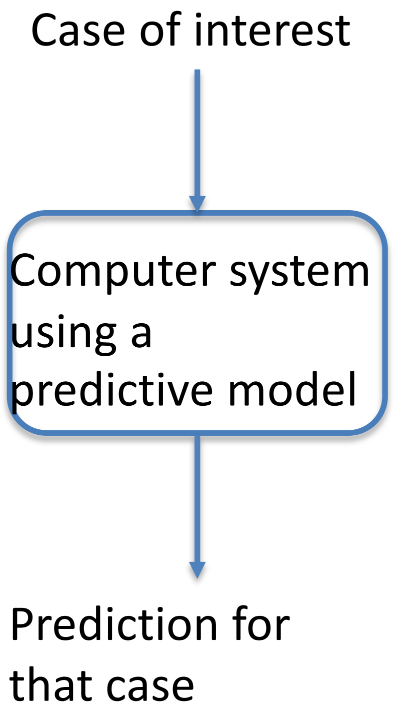

Labelled data of many previous cases
Machine learning algorithm
Computer system using a predictive model

This is *supervised* learning

- Input is a large dataset, with many previous cases
  - Eg attributes of each case
  - Also, labelled with actual value of predicted attribute
- Output of the training process is a computer system that includes a predictive model
  - Or maybe just the parameters that will be used to set up the predictive model

Deploy a predictive model

- Typical deployment



- As each case arrives, the system is run on the data about the case
  - Eg attributes of the case
- Output of the system is a prediction for some property of the case
- Eg, given input which is the contents, subject, sender etc of an email, output is either “spam” or “not spam”

More Examples

- Case is an agricultural plot growing season
  - Given data: seed-type, soil type, amount-of-rain, average-temperature, amount of nutrient1, amount of nutrient2, etc
  - Goal is to predict money-value-of-produced-crop [a *regression* task, as prediction is a quantitative attribute]
- Case is one voter
  - Given data: age, gender, job-type, salary, marital-status, number-of-dependents, etc
  - Goal is to predict party they vote for [a *classification* task, as prediction is a categorical attribute]

Agenda

- Overview
- *Features*
- Evaluation
- Linear Regression
- Extensions of linear regression
- K-nearest-neighbour regression
- Decision-tree classification
- “Logistic regression” classification

The data format

- Usually, training data is presented in “wide” logical format
  - In deployment, similar format for each case
- Each case has many attributes (also called “features”) as columns of a single row
  - In training data, the true value of the predicted attribute may be another column, or in a separate dataset that aligns with training rows
- Other terms used for the features: covariates, regressors, independent variables, input variables, control variables, explanatory variables, etc
- Other terms for the predicted feature: target, response variable, output variable, dependent variable, etc

Encoding complex data

- How do we handle data that doesn’t seem like columns of integer, float, string, etc?
  - Eg data includes an image (medical scan, etc) or audio recording (eg for voice recognition)
- We may treat this as many simple-value attributes, from the basic media format
  - Eg an attribute for each separate pixel in an image,
  - Eg an attribute for sound intensity at each separate sampled moment in an audio recording

Transformed attributes

- Motivation: Many ML algorithms work best when each attribute has a similar scale, or distribution of data values
  - Eg in Australia, most adult's heights are in range 1.4 m to 2.1 m, but most annual salaries range from 15,000 to 400,000, and there are some which are much much larger
- We often transform or rescale the data of an attribute, to get a scale that works well

Transformed attributes

- Some methods that are common:
  - Transform an attribute linearly so data is mostly in range between 0 and 1, or between -1 and 1
  - Transform an attribute so distribution of data values has mean=0, variance=1 (called “z-score”)
    - Calculating the score requires the whole distribution (usually approximated using the attribute values from the training data set); the calculation can be quite expensive to perform
  - Transform data which seems to vary multiplicatively, by taking logarithm
    - Eg noise volumes expressed using dB, or earthquake intensity on Richter scale

Derived attributes

- One can include attributes that are derived from one or more of the original ones
- Eg, if original data is image, with each pixel as an attribute, we can include average-brightness-over-whole-image as a new attribute, and/or we can include an attribute is-pixel-i-on-an-edge (using edge detection calculation, based on neighbouring pixels)

Encoding a categorical attribute

- Motivation: some ML methods only work when all attributes are quantitative
  - Original data may have categorical or ordinal attributes
- "Label encoding": each attribute value ("label") is assigned an integer
  - Often, we assign 1 to first label we see in data, 2 to next label we see, etc
  - If attribute is ordinal, we may assign 1 to smallest label, 2 to next smallest, etc
- Alternative encoding (usually preferred in practice) is "One-hot encoding": introduce a separate encoding attribute x_L for each original label L
  - This encoding attribute gets value 1 for item with L as its label, 0 for item with a different label
  - We say the attribute x_L is an "indicator" (or "characteristic feature") for label L

Agenda

- Overview
- Features
- *Evaluation*
- Linear Regression
- Extensions of linear regression
- K-nearest-neighbour regression
- Decision-tree classification
- “Logistic regression” classification

Importance of evaluating a model

- There are many different ways to produce a predictive model for a given dataset
  - They may give very different models
  - We want to know whether a model is working well, or not
  - Also, we may want to choose the best from several different possibilities
    - This is a kind of hyper-parameter tuning (deciding whether to use a linear prediction or a kNN one, deciding which k to use in kNN, or how much regularisation to do, etc)
- We want to have a score that reflects the success of a model in prediction
  - This is only one aspect of the choice! We also care about computation cost, understandability, etc

Scoring a model

- There are a wide variety of different scoring definitions available
- Scikit-learn offers many of these built-in, using sklearn.metrics package
- Some scores are better when high (eg accuracy) and others are better when low (eg rmse). Be aware which is which
- Some scores are meaningfully compared from different algorithms on different datasets, others only can be compared between different algorithms running with exactly the same dataset
- Always: look at score on test data that was not used during training or validation
    - Otherwise scores are deceptive, and model may work much worse in practice than you expect

Scoring a regression - RMSE

- Mean-square-error is defined as square of the difference between predicted and actual value, averaged on the various items in the test dataset
  - We square each difference, so that positive and negative mis-predictions do not cancel out
  - Statistical theory that this is a good thing to optimise, for cases where errors are independent and random
- MSE = [(predicted(test1) – actual(test1))^2 + (predicted(test2)-actual(test2))^2 + (predicted(test3)-actual(test3))^2 +etc]/(number of test cases)
- RMSE = square-root of MSE
  - This has same units as target attribute
- But hard to interpret, except by comparing two scores on the same test set
  - We desire a small value for RMSE

Scoring a regression - R^2

- Another score often used is called R-squared
- R^2 is at most 1
  - $R^2 = 1$ means excellent prediction
  - Despite its name, R^2 can be negative!
  - Predictions is social science rarely have R2 above 0.5, but predictions in physical sciences often can be much closer to 1 (eg 0.9)
- See https://en.wikipedia.org/wiki/Coefficient_of_determination

Scoring a classifier – accuracy

- Accuracy of a classifier is defined as the fraction of test items where the prediction is correct (ie how many i where predict(testi) = actual(testi)
- Warning: when the data is unbalanced (many more examples in one class compared to others, eg most patients do not have a disease), this is not very useful
  - high accuracy can come by just always predicting most common case (ignoring the data attributes)

Confusion matrix

- Count how many test cases there are, for each combination of predicted-label and actual-label
  - Record counts in a matrix
- Eg if we predict cat, and animal is really dog, this is counted in matrix entry at intersection of row predict=cat and column where actual=dog
- Accuracy = (sum of diagonal entries)/(sum of all entries)
- Full matrix helps us see what kinds of mistake our classifier is making

Scoring a binary classifier

- For a binary classifier (only two labels, eg item has-property-of-interest or item doesn't-have-property-of-interest)
- Name confusion matrix entries as

Predicted \ Actual	actually has property	Actually doesn't have property
Positive (predicted to have property)	TP	FP (number of False Positive cases)
Negative (predicted as not-with-property)	FN (number of False Negative cases)	TN

Scores for binary classifier

- Many different names used in different fields of scholarship, for various ratios
- Sensitivity, recall, hit-rate: TP/(TP+FN)
- Precision: TP/(TP+FP)
- Selectivity: TN/(TN+FP)
- F1 = 2*TP/(2*TP+FN+FP)
- In real applications, different situations may have very different impact
  - Eg predict-disease when no-actual-disease causes some inconvenience of extra tests, but predict-no-disease when actually has-disease may lead to patient death, and predict-disease when has-disease may save the life
  - So we want to choose predictive model that makes impact as good as possible
  - Choose to optimise the score that suits the domain characteristic

Agenda

- Overview
- Features
- Evaluation
- *Linear Regression*
- Extensions of linear regression
- K-nearest-neighbour regression
- Decision-tree classification
- “Logistic regression” classification

Linear predictive models

- Prediction = w_1*case.x_1+w_2*case.x_2+w_3*case.x_3+ etc
  - Here x_1, x_2 etc are attributes of a case
  - w_1, w_2 etc are “weights” that determine which predictive model is being used, within this family
  - Other notation: weights are also called coefficients, and are often denoted as $\boldsymbol{\beta}_1$, $\boldsymbol{\beta}_2$ etc
- Each attribute contributes linearly to the prediction
  - A given increase in the attribute, leads to same change in predicted value, no matter what value the attribute has, nor what values other attributes have

Limitations

- All attributes must be quantitative
  - Use encoding to convert any attributes that are originally categorical or ordered
- Choice of model is most influenced by the attributes with greatest variation
  - It is wise to transform and rescale each attribute, so they are comparable
- Before training a linear model, look at data to be convinced it seems linear
  - If not, maybe transform some attributes to adjust
- Linear models are quite fragile if data contains errors or outliers
  - It’s wise to remove these before training a model

Which model in the family?

- The simplest (and traditional) choice is to find the set of weights for which the mean-square-error (MSE) is smallest *on the training set*
  - That is, minimize [(predicted(train1) – actual(train1))^2 + (predicted(train2)-actual(train2))^2 + (predicted(train3)-actual(train3))^2 + etc]/(number of training items)
    - Here each training instance traini is itself a vector of attribute values
- There is a direct calculation for this using matrix operations
- But heuristic search is often used when there are large amounts of data to train on
  - Eg Stochastic gradient descent (“SGD”)

Scikit-learn training and deployment

This code example
(from Grok ML module)
uses data stored as pandas dataframe

```
import pandas as pd
from sklearn import linear_model
from sklearn import metrics
from sklearn.model_selection import train_test_split

df = pd.read_csv('CAvideos_stat.csv')
X = df.values[:, 1:3]       # slice dataFrame for input variables
y = df.values[:, 3]         # slice dataFrame for target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
random_state=42)

# train linear regression predictor to fit training data
regr = linear_model.LinearRegression().fit(X_train, y_train)

# The coefficients
print('Coefficients:')
print(regr.coef_)

# Use the model to predict y from X_test
y_pred = regr.predict(X_test)
```

Scikit-learn evaluation

```
import pandas as pd
from math import sqrt
from sklearn import metrics

# Somehow produce a model regr
# Use the model to predict y from X_test
y_pred = regr.predict(X_test)

# y_test is dataframe of actual values for X_test
# mean squared error
mse = metrics.mean_squared_error(y_test, y_pred)
print('Root mean squared error (RMSE):', sqrt(mse))
# R-squared score
print('R-squared score:', metrics.r2_score(y_test, y_pred))
```

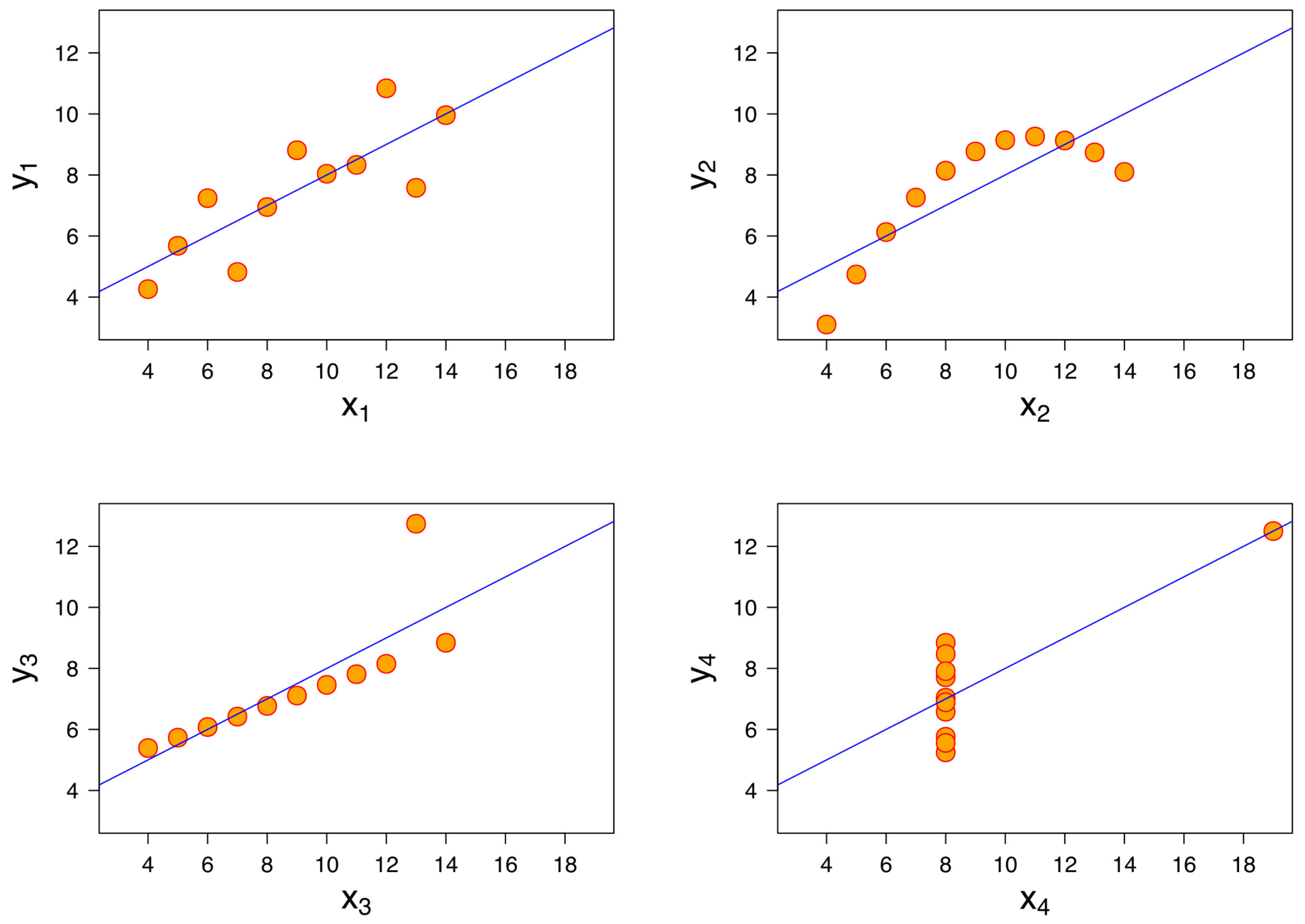
Interpretation of weights

- Coefficient of x_1 gives “amount two cases are predicted to differ in target-attribute, if they are the same on all other attributes of the dataset, and differ by 1 in value for x_1”
- Do not treat this as causal without much more evidence
  - Do not say “if we increase x_1 by 1, then target attribute will increase by w_1”
    - one often can’t simply change one attribute while keeping others the same
    - Eg suppose we predict salary from height!
  - Decision about which attributes are included in the model, can even alter the sign of the coefficient

Warnings

- Remember how misleading it is, to look at score on data that is used during training (or while validating to tune hyper-parameters)
- Don’t extrapolate with the model to cases that are quite unlike training data
  - Instead, only predict for data that is similar to training cases
  - Eg if training data is mostly in developed countries, don’t expect model to hold for developing world
  - Eg if training data is crop yields in tropics, don’t expect model to hold in temperate regions
- The software makes it very easy to produce a model, but the model may predict quite badly (eg if real world is not close to linear, if training data has outliers, etc)

Anscombe’s quartet (all have same best-fit linear predictor)



Agenda

- Overview
- Features
- Evaluation
- Linear Regression
- *Extensions of linear regression*
- K-nearest-neighbour regression
- Decision-tree classification
- “Logistic regression” classification

Polynomial regression

- If we expect target attribute to depend non-linearly on attribute x_i, a common technique is to include x_i, and then also x_i^2, x_i^3 etc as extra (derived) attributes in a linear regression
  - Up to some maximum power of x_i
- Warning: only do regression with powers of x_i, when you have good reason to expect a polynomial impact of the degree you are including
  - It is very easy to overfit training data with polynomial model

Interactions between features

- If we expect an interaction between x_1 and x_3, we might include x_1*x_3 as an extra derived attribute
  - This works best if one of the attributes is indicator variable (value is 0 or 1, or sometimes -1/1 is used)
- Warning: the number of interactions one might decide to include grows rapidly with number of attributes

Regularisation

- With many attributes in a dataset, overfitting becomes quite likely
- The search for a good model can be shaped to reduce this danger, by penalizing models where many weights are large
- Many ways to do this, often controlled by hyper-parameters
- Eg in scikit-learn

```
from sklearn import linear_model
regr = linear_model.Ridge(alpha=.5).fit(X_train, y_train)
```

Agenda

- Overview
- Features
- Evaluation
- Linear Regression
- Extensions of linear regression
- *K-nearest-neighbour regression*
- Decision-tree classification
- “Logistic regression” classification

Overview

- For a given case where we want a prediction, we find the k nearest members of the training set
  - Measure “distance” between two rows is based on the difference in each attribute, combined somehow
  - Eg “Euclidean distance” between case and traini is $\sqrt{}((\text{case}.x_1 - \text{traini}.x_1)^2+(\text{case}.x_2-\text{traini}.x_2)^2+\ldots)$
- Prediction for case is *weighted average* of the actual target value on each of the nearest training instances, combined with weight which is 1/(distance of that training instance from the case)
- k (the number of neighbours used) is a hyper-parameter of this regression technique which is often called “kNN” regression

Some strengths and limitations

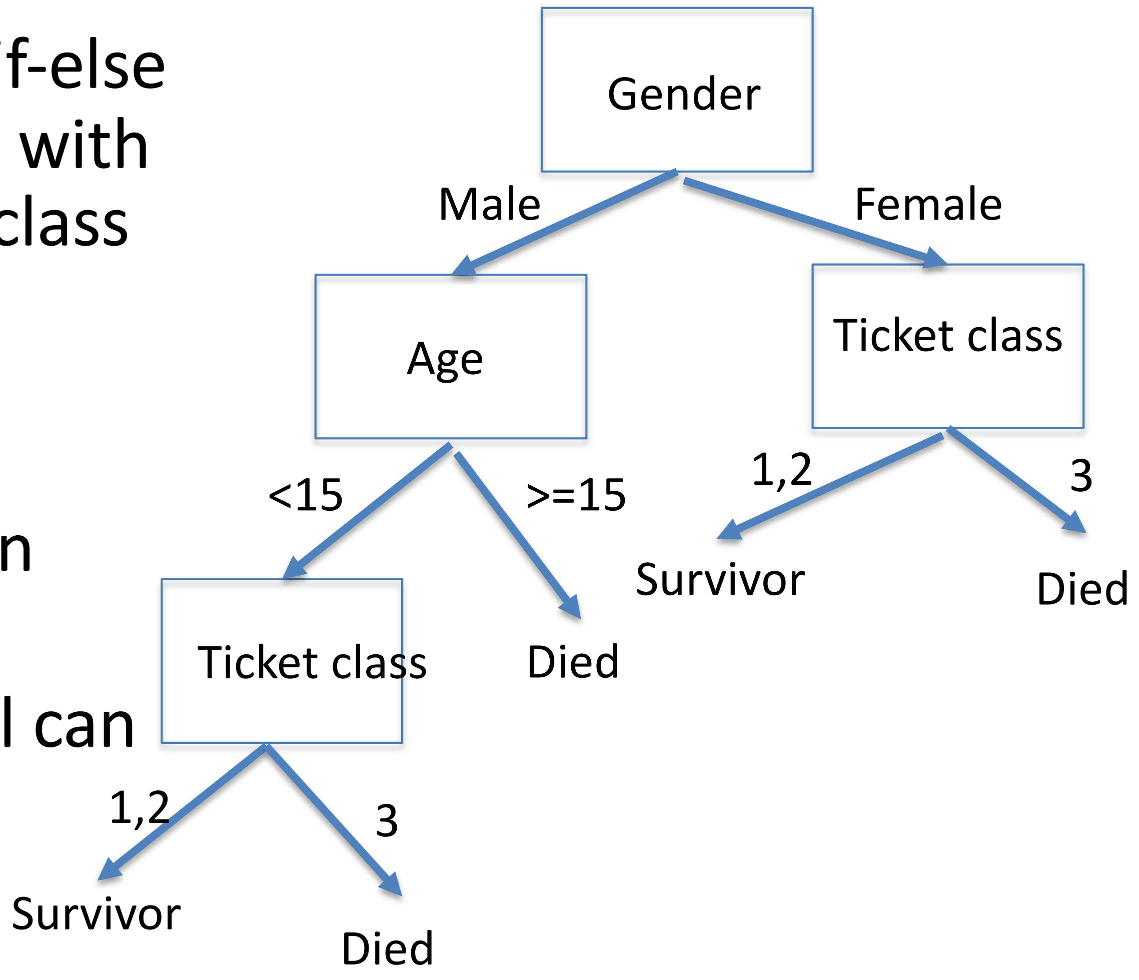
- kNN works well with data that has quite complicated patterns (not just nearly linear)
- kNN is not disturbed much by a few outliers/errors in data
- Deploying a kNN predictive model takes a lot of computation and storage

Agenda

- Overview
- Features
- Evaluation
- Linear Regression
- Extensions of linear regression
- K-nearest-neighbour regression
- *Decision-tree classification*
- “Logistic regression” classification

Overview of the family of models

- Essentially, a deeply nested collection of if-else logic choices, ending with the assignment of a class label
  - Like the game “20 questions”
- Each choice condition can be different
- The predictive model can be easy for users to understand



Possible classifier for Titanic passenger survival

Variety of models

- Each decision looks at the value of one attribute, and chooses next step based on comparing the value to model parameters
  - For split on a quantitative attribute, partition the range of possible values
  - For split on categorical attribute, each possible value can be a separate choice

Scikit-learn code

```
import pandas as pd
from sklearn import metrics
from sklearn import tree
from sklearn.model_selection import train_test_split

df = pd.read_csv('dataR2.csv')
[num_row, num_var] = df.shape
X = df.values[:, 0:num_var - 1] # slice dataFrame for input variables
y = df.values[:, num_var - 1]# slice dataFrame for target variable
X_train, X_test, y_train, y_test = \
         train_test_split(X, y, test_size=0.3, random_state=42)
clf = tree.DecisionTreeClassifier().fit(X_train, y_train)

y_pred = clf.predict(X_test)

print('Calculate the accuracy using the test data')
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
```

Choosing the best model

- CART algorithm:
- Find the choice of one splitting attribute and thresholds that produces the “lowest impurity” division
  - The subsets defined by this split are as close to uniform-label as possible (and different from one another)
- Then treat each subset as a new dataset and find its best split, etc

Warnings

- Decision tree classifiers are quite prone to overfitting
  - So we often restrict chosen tree so it is not too deep, etc
- When there are many attributes, the search for best decision tree can be very slow

Agenda

- Overview
- Features
- Evaluation
- Linear Regression
- Extensions of linear regression
- K-nearest-neighbour regression
- Decision-tree classification
- *“Logistic regression” classification*

Overview

- A logistic regression predicts a probability value “how likely is it that a case with these attributes, is in the class”?
- To use this for binary classification, we assign the case to the class, when the probability predicted is greater than 0.5
  - If there are multiple possible classes, predict a probability for each, and then assign case to whichever class has highest probability

Further learning

- Many wikipedia articles, start from:

https://en.wikipedia.org/wiki/Supervised_learning

- Data Science from Scratch (2nd ed), by J. Grus, O'Reilly 2019
- Hands-On Machine Learning with Scikit-Learn & TensorFlow, by A. Géron, O'Reilly 2017
- The Master Algorithm, by P. Domingos, Penguin 2015
- USyd unit COMP3308 Introduction to Artificial Intelligence
- For scikit-learn, see https://scikit-learn.org/