IMPERIAL

Machine Learning for Neuroscience

ML4NS

Regression Models and Linear Prediction

Payam Barnaghi
Department of Brain Sciences &
School of Convergence Science in Human and Artificial Intelligence
Imperial College London
January 2025

1

1

Linear models

IMPERIAL

- Linear models use a *linear* combination of features to make their assessments/predictions.

- During the training process, the model calculates weights for each of the features in the data to build a model that can predict or estimate a target value.
- For example, if your target is the risk that a person's cognitive ability will decline and your variables are age and sleep quality, and a cognitive test result, a simple linear model would be:

$$\alpha * Age + \beta * SleepQuality + \theta * CognitiveTestScore = Risk\_score$$

2

2

Linear models: example

IMPERIAL

3

3

Linear models - training

IMPERIAL

$\alpha * Age + \beta * SleepQuality + \theta * CognitiveTestScore = Risk\_score$

The question is how to learn the coefficients (weights in ML terms)?

4

4

Linear regression

IMPERIAL

– Linear regression is a simple approach to supervised learning. It assumes that the dependence of Y on $x_1, x_2, \ldots x_n$ is linear.

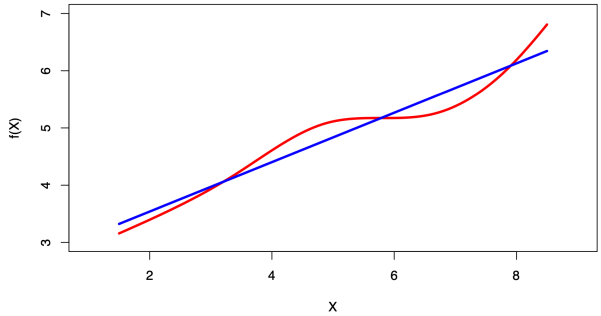
5

5

Linear regression and non-linear functions

IMPERIAL

– Learning the weights in linear regression is a type of supervised learning.
– It assumes that the target (often shown as Y) is dependent on the features.



– True regression functions are never linear!

Figure source: Tibshirani *et al.*, An introduction to statistical learning, Springer,: https://www.statlearning.com

6

6

The terminology in linear regression

IMPERIAL

- Let's assume a model:

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

Or

$$Y = w_0 + w_1 X + \varepsilon$$

- Where: β_0 and β_1 are two unknown parameters that represent the intercept and slope, also known as coefficients or weights, and ε is the error term.

7

7

Simple linear regression using a single predictor

IMPERIAL

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

- Given some estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ for the model coefficients, we want to predict future values using:

$$\hat{Y} = \widehat{\beta_0} + \widehat{\beta_1} X + \varepsilon$$

- where $\hat{Y}$ indicates a prediction of Y on the basis of $X = x$. The hat symbol denotes an estimated value.

8

8

Liner regression and parameters

IMPERIAL

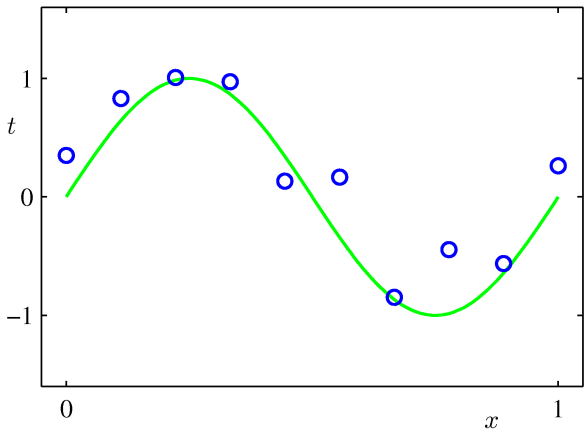


Figure source: Tibshirani *et al.*, An introduction to statistical learning, Springer,: https://www.statlearning.com

9

9

Estimating the parameters

IMPERIAL

– Let:

$$\widehat{y_i} = \widehat{\beta_0} + \widehat{\beta_1} x_i$$

And y_i to be the actual value of the target for x_i

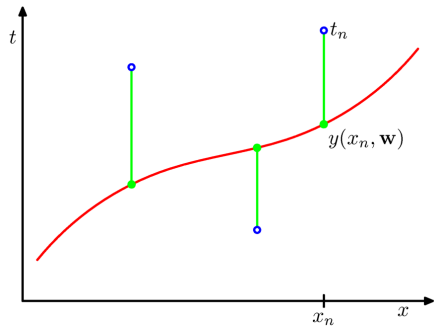
Then e_i = y_i - $\widehat{y_i}$ represents the error or the i[th] *residual.*

10

10

Residual

IMPERIAL



$$Error = Predicted\,value - Actual\,value$$

$$Error = \sum(Predicted\,value - Actual\,value)/n$$

Source: : Tibshirani *et al.*, An introduction to statistical learning, Springer,: https://www.statlearning.com

11

11

Estimating the parameters

IMPERIAL

$$\hat{y}_i = \widehat{\beta_0} + \widehat{\beta_1} x_i$$

$$e_i = y_i - \hat{y}_i$$

– The residual sum of squares (RSS) is defined as:

$$RSS = e_1^2 + e_2^2 + \ldots + e_n^2$$

12

12

Estimating the parameters*

IMPERIAL

- The least squares approach chooses $\hat{\beta}_0$ and $\hat{\beta}_1$ to minimise the RSS.
- The minimising values can be shown to be

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2},$$
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x},$$

where $\bar{y} \equiv \frac{1}{n}\sum_{i=1}^{n} y_i$ and $\bar{x} \equiv \frac{1}{n}\sum_{i=1}^{n} x_i$ are the sample means.

*Note: When you see this star sign (*) on the slides, you can skip it if you are not interested in the detailed mathematical background. This information is provided as an additional background and will not be part of the module assessment, and you are not expected to know them.

Source: Tibshirani *et al.*, An introduction to statistical learning, Springer,: https://www.statlearning.com

13

13

Estimating the parameters in a simple regression

IMPERIAL

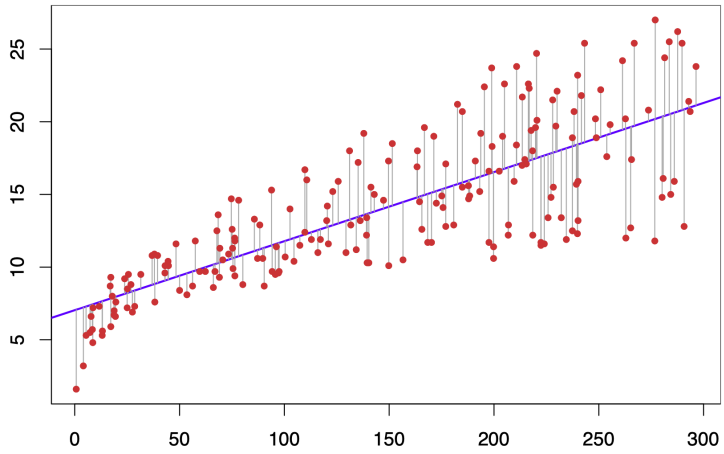
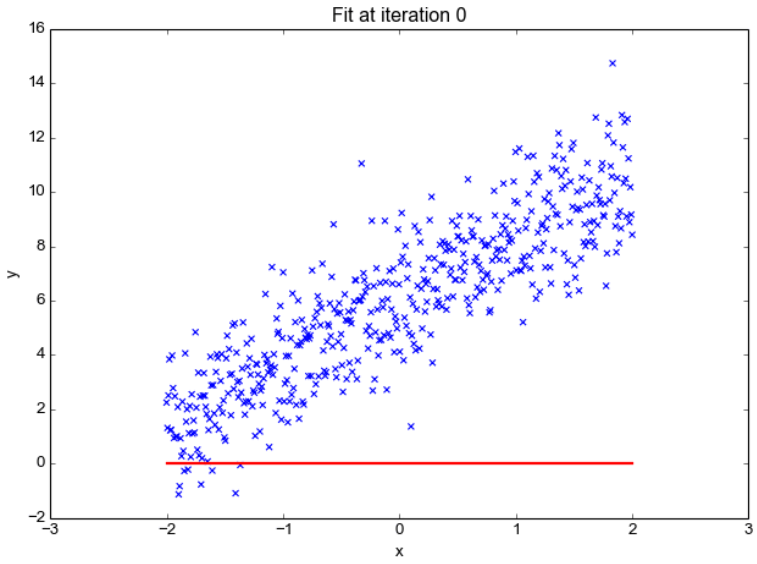


Figure source: Tibshirani *et al.*, An introduction to statistical learning, Springer,: https://www.statlearning.com

14

14

IMPERIAL



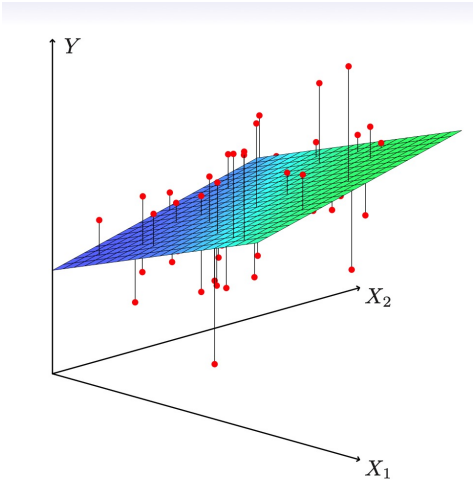
Source: Nagesh Singh Chauhan, KDnuggets *https://www.kdnuggets.com/2019/03/beginners-guide-linear-regression-python-scikit-learn.html*

15

15

IMPERIAL

Estimation and prediction for multiple regression



16

16

Defining linear regression - again

IMPERIAL

- A Linear Regression model uses a linear model with coefficients $w = (w_1, \ldots, w_n)$ to minimise the residual sum of squares between the observed targets in the dataset and the targets that are predicted by the linear approximation used in the model.

17

17

Sequential learning – Stochastic Gradient Descent*

IMPERIAL

- The input data is considered one at a time.
- We can use stochastic (sequential) gradient descent to learn the parameters of a multiparameter regression:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n$$

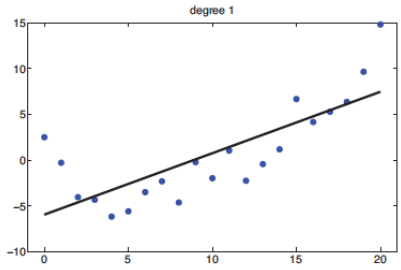
- This is also known as the *least-mean-squares (LMS) algorithm*.
- More on this topic later – in the neural networks section.

18

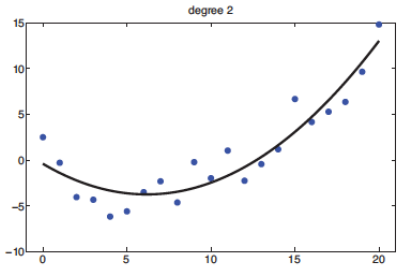
18

Obviously, the solution is not always a line

IMPERIAL



(a) Linear regression on some 1d data.



(b) Data with polynomial regression (degree 2).

Figure source: Tibshirani *et al.*, An introduction to statistical learning, Springer,: https://www.statlearning.com

19

19

Type of problems that we can apply to

IMPERIAL

- Predict the change in Neuropsychiatric Inventory (NPI) scores in people living with dementia as a function of a number of different clinical measurements and/or in-home observations.
  - e.g., https://pubmed.ncbi.nlm.nih.gov/22531424/
- Predicting the functional consequences after TBI as a function of TBI severity, more prominent CT abnormality, past psychiatric history and alcohol intoxication.
  - e.g., https://pubmed.ncbi.nlm.nih.gov/34711118/

20

20

How to evaluate your LR model

IMPERIAL

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)

21

21

Mean Absolute Error (MAE)

IMPERIAL

- MAE is the mean of the absolute error values

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$$

22

22

Mean Squared Error (MSE)

IMPERIAL

- MSE is the mean of squared error

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

- What do you think is the effect of squaring the errors here?

23

23

Root Mean Squared Error (RMSE)

IMPERIAL

- The square root of mean square errors

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

24

24

Mean Squared Error (MSE) and Root Mean Squared Error (RMSE)

IMPERIAL

- RMSE is the square root of MSE.
- MSE is measured in units that are the square of the target variable, while RMSE is measured in the same units as the target variable.
- Due to its formulation, MSE, just like the squared loss function that it derives from, effectively penalises larger errors more severely.

Source: Machine Learning with Spark - Second Edition by Rajdeep Dua, Manpreet Singh Ghotra, Nick Pentreath, O'Reilly

25

25

Linear regression in Python

IMPERIAL

```
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression
>>> X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
>>> # y = 1 * x_0 + 2 * x_1 + 3
>>> y = np.dot(X, np.array([1, 2])) + 3
>>> reg = LinearRegression().fit(X, y)
>>> reg.score(X, y)
1.0
>>> reg.coef_
array([1., 2.])
>>> reg.intercept_
3.0...
>>> reg.predict(np.array([[3, 5]]))
array([16.])
```

Source: scikit-learn

26

26

Logistic regression

IMPERIAL

- We can generalise linear regression to the (binary) classification setting by making some changes.
- We compute a linear combination of the inputs, as before, but then we pass this through a function that ensures $0 \leq \mu(x) \leq 1$ by defining:

$$\mu(\mathbf{x}) = \text{sigm}(\mathbf{w}^T\mathbf{x})$$

27

27

The Sigmoid function

IMPERIAL

- where sigm(η) refers to the sigmoid function, also known as the *logistic* or *logit* function.
- This is defined as

$$\text{sigm}(\eta) \triangleq \frac{1}{1+\exp(-\eta)} = \frac{e^\eta}{e^\eta + 1}$$

- The term "*sigmoid*" means S-shaped for a plot.
- It is also known as a squashing function since it maps the whole real line to [0, 1].

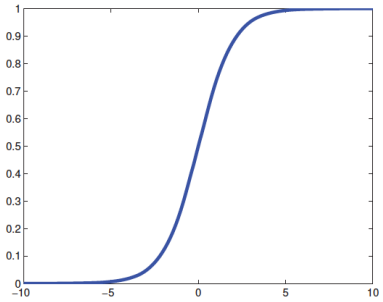
28

28

Logistic regression

IMPERIAL

– The process of applying a linear combination of the inputs, as before, but then we pass this through a logistics function, is called logistic regression due to its similarity to linear regression (although it is a form of classification, not regression!).



29

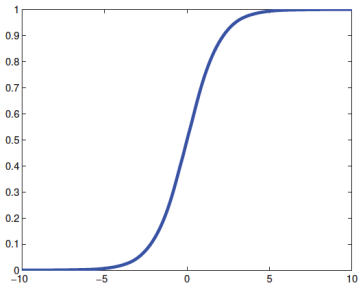
29

Decision boundary

IMPERIAL

– We can imagine drawing a vertical line at $x = x^*$; this is known as a decision boundary.

– Everything to the left of this line is classified as a 0, and everything to the right of the line is classified as a 1.



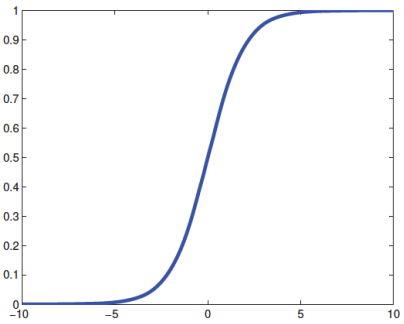
30

30

Decision boundary

IMPERIAL

– This decision rule could have a non-zero error rate even on the training set. This is because the data is not linearly separable, i.e., we can draw no straight line to separate the 0s from the 1s.



31

31

Practical methodology

IMPERIAL

– Parametric vs. non-parametric models
– Non-parametric models make no assumption on the data distribution or dataset size to generate a model.
– The parametric models have a fixed set of parameters (e.g., k-means)
– In non-parametric models , the number of parameters grows with the amount of training data.

32

32

Parametric vs. non-parametric models

IMPERIAL

- Parametric models have the advantage of often being faster to use, but the disadvantage of making stronger assumptions about the nature of the data distributions.
- Nonparametric models are more flexible, but often computationally intractable for large datasets.

33

33

K-nearest neighbours

IMPERIAL

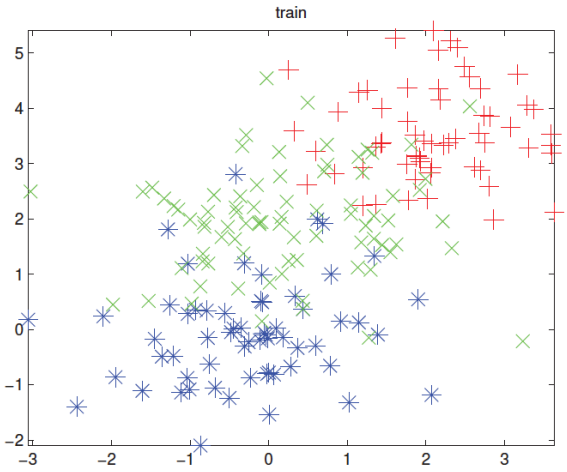
- A simple example of a non-parametric classifier is the K nearest neighbour (KNN) classifier.
- KNN simply "looks at" the K points in the training set that are nearest to the test input x.
- It then counts the number of members of each class in this set and returns the empirical fraction as the estimate.

34

34

K-nearest neighbours - example

IMPERIAL



35

35

KNN

IMPERIAL

- This method is an example of memory-based learning or instance-based learning.
- The most common distance metric to use is Euclidean distance , although other metrics can also be used.

36

36

Euclidean distance

IMPERIAL

- Euclidean distance is a distance measure that determines the length of a segment that connects points between two points.

$$Dist(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- Euclidean distance can be skewed depending on the units/scale of the features.
- Remember you may need to normalise your data/features before applying the Euclidean distance.

37

37

Euclidean distance - example

IMPERIAL

$$Dist(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

38

38

The curse of dimensionality

IMPERIAL

- The KNN classifier is simple and can work quite well, provided it is given a good distance metric and has enough labelled training data.
- However, the main problem with KNN classifiers is that they do not work well with high-dimensional inputs. The poor performance in high-dimensional settings is due to the *curse of dimensionality*.

39

39

KNN in Python

IMPERIAL

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsClassifier
>>> neigh = KNeighborsClassifier(n_neighbors=3)
>>> neigh.fit(X, y)
KNeighborsClassifier(...)
>>> print(neigh.predict([[1.1]]))
[0]
>>> print(neigh.predict_proba([[0.9]]))
[[0.666... 0.333...]]
```

Methods

`fit`(X, y)	Fit the k-nearest neighbors classifier from the training dataset.
`get_params`([deep])	Get parameters for this estimator.
`kneighbors`([X, n_neighbors, return_distance])	Find the K-neighbors of a point.
`kneighbors_graph`([X, n_neighbors, mode])	Compute the (weighted) graph of k-Neighbors for points in X.
`predict`(X)	Predict the class labels for the provided data.
`predict_proba`(X)	Return probability estimates for the test data X.
`score`(X, y[, sample_weight])	Return the mean accuracy on the given test data and labels.
`set_params`(**params)	Set the parameters of this estimator.

Source: scikit-learn

40

40

Model selection

IMPERIAL

– When we have a variety of models of different complexity (e.g., linear or logistic regression models with different degree polynomials or KNN classifiers with different values of K), how should we pick the right one?

41

41

Misclassification rate

IMPERIAL

– A natural approach is to compute the misclassification rate on the training set for each method.

– However, what we really care about is generalisation error, which is the expected value of the misclassification rate when averaged over future data.

– This can be approximated by computing the misclassification rate on a large independent test set, not used during model training.

42

42

Model selection - revisited

IMPERIAL

- When training the model, we don't have access to the test set (by assumption), so we cannot use the test set to pick the model of the right complexity.
- However, we can create a test set by partitioning the training set into two parts: the part used for training the model and the second part, called the validation set, used for selecting the model complexity.
- We then fit all the models on the training set, and evaluate their performance on the validation set, and pick the best model/paramters.
- Once we have picked the best model, we can refit it to all the available data. If we have a separate test set, we can evaluate performance on this, to estimate the performance of our method.

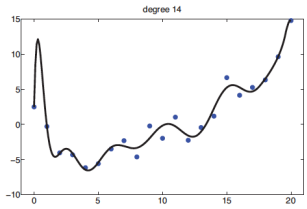
43

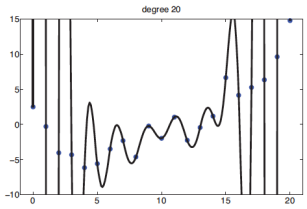
43

Overfitting

IMPERIAL

- When we fit highly flexible models, we need to be careful not to overfit the data; that is, we should avoid trying to model every minor variation in the input since this is more likely to be noise than true signal.





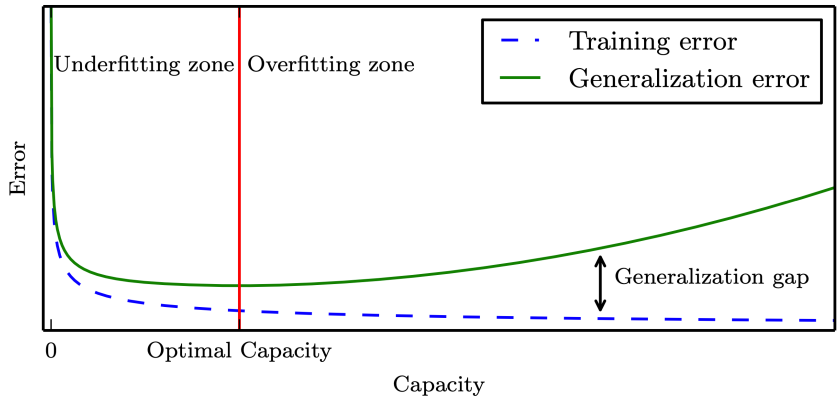
- This is illustrated in the figure above, where we see that using a high-degree polynomial results in a very "wiggly" curve. It is unlikely that the true function has such extreme oscillations.
- Using such a model may result in in accurate predictions of future outputs.

44

44

Training and test errors

IMPERIAL



Source: Deep Learning, Ian Goodfellow *et al.*, MIT press.

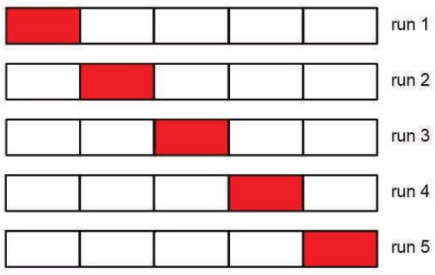
45

45

Cross-validation

IMPERIAL

– The idea is simple: we split the training data into K folds; then, for each fold $k \in \{1, \ldots, K\}$, we train on all the folds but the k'th, and test on the k'th, in a round-robin fashion.



46

46

N-Fold cross validation

IMPERIAL

- We then compute the error averaged over all the folds and use this as a proxy for the test error.
- Note that each point gets predicted only once, although it will be used for training K−1 times.
- It is common to use K = 5 or 10; this is called 5 or 10-fold CV.
- If we set K = N, then we get a method called leave-one-out cross-validation, or LOOCV, since in fold i, we train on all the data cases except for i and then test on i.

47

47

Cross validation and model selection

IMPERIAL

- Choosing K for a KNN classifier is a special case of a more general problem known as model selection, where we have to choose between models with different degrees of flexibility.
- Cross-validation is widely used for solving such problems, although there are other methods/approaches (more on this topic later).

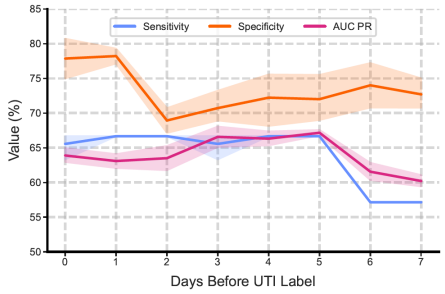
48

48

Reporting the results

IMPERIAL

		Sensitivity	Specificity	Precision
Date	Validation	86.6 (80.9 - 92.3)	94.5 (91.7 - 97.3)	87.3 (82.2 - 92.4)
Date	Test	69.0 (64.4 - 73.5)	94.1 (92.0 - 96.2)	81.9 (75.5 - 88.2)
Date-ID	Validation	98.3 (95.5 - 101.1)	90.0 (85.5 - 94.5)	81.7 (74.4 - 89.1)
Date-ID	Test	74.7 (67.9 - 81.5)	87.9 (85.0 - 90.9)	77.0 (71.9 - 82.1)



		Accuracy	No. of Participants	Positive : Negative	$\Pr(\hat{Y} = 1 \mid \text{Sex})$
Date	Female	52.6 (31.8 - 73.4)	16	1 : 1.7	35.5 (32.4 - 38.5)
	Male	86.5 (76.2 - 96.9)	25	1 : 5.5	32.3 (30.9 - 33.8)
Date-ID	Female	54.6 (26.8 - 82.4)	11	1 : 2.1	37.4 (33.6 - 41.1)
	Male	85.3 (72.9 - 97.7)	20	1 : 4.1	38.0 (36.4 - 39.7)

A. Capstick *et al.*, *NPJ Digital Medicine*, 2023.

49

49

No free lunch theorem

IMPERIAL

- All models are wrong, but some models are useful. — George Box (Box and Draper 1987).
- Much of machine learning is concerned with devising different models, and different algorithms to fit them.
- We can use methods such as cross-validation to choose the best method for our problem empirically.
- However, there is no universally best model — this is sometimes called the no free lunch theorem (Wolpert 1996).
- The reason for this is that a set of assumptions that work well in one domain may work poorly in another.

50

50

Different models for different problems

IMPERIAL

- We need to develop many different types of models to cover the wide variety of data that occurs in the real world.
- And for each model, there may be many different methods that we can use to train the model, which makes different speed-accuracy-complexity trade-offs.
- More on this next week.

51

51

IMPERIAL

Review questions

Source: The questions are adapted from "Deep Learning Interviews", Shlomo Kashani.

52

Q1

IMPERIAL

– True or False: A non-parametric model has no parameter to train?

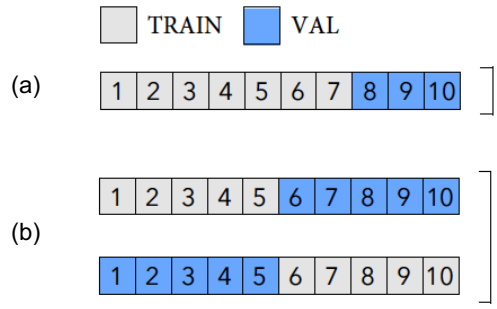
53

53

Q2

IMPERIAL

– The figure depicts two different cross-validation approaches. Which one is a k-fold cross validation?



54

54

Q3

IMPERIAL

– Sigmoid function is a nonlinear function, Ture or False?

55

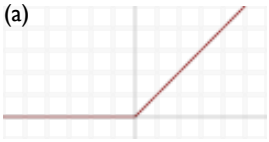
55

Q4

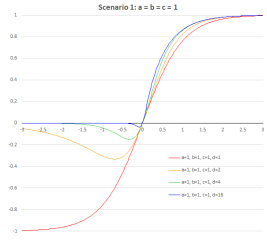
IMPERIAL

– Which of these show the output of a sigmoid function?

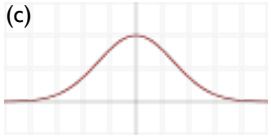
(a)



(b)



(c)



(d)

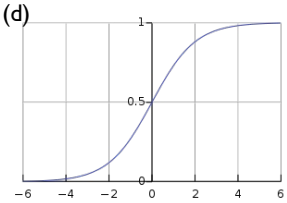


Image sources: Wikipedia

56

56

Acknowledgement

IMPERIAL

– Some of the slides in this lecture are adapted from Kevin Murphy's and Tibshirani et al.'s book:
  – Machine Learning: A Probabilistic Perspective Kevin P. Murphy, MIT Press.
  – Tibshirani et al.: An introduction to statistical learning: https://www.statlearning.com

57

57

IMPERIAL

Additional slides (optional further reading)

In the linear regression section, we discussed different metrics to minimise the error. There are also methods that control the weights (coefficients) and try to find optimised values/sets of coefficients/weights that can be used in a regression model. Lasso and Ridge are two of these techniques. The next few slides discuss them briefly.

58

L1 Regularisation (Lasso)*

IMPERIAL

- Lasso is an acronym for: Least Absolute Shrinkage and Selection Operator.
- Lasso regression is a regression model that uses $\ell 1$ regularisation.
- In LASSO we modify the optimisation function and add a coefficient which is calculated based on the square of weights (parameters).

$$\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 + \lambda \sum_{i=1}^{n}|\beta_i|$$

59

59

Lasso – setting λ*

IMPERIAL

$$\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 + \lambda \sum_{i=1}^{n}|\beta_i|$$

- If we set lambda to zero, then the function becomes like an ordinary least squares
- If we set lambda to a very large value, it will make coefficients to be come zero, which will make the model underfit .

Source: Anuja Nagpal, https://builtin.com/data-science/l2-regularization

60

60

$\ell 2$ regularisation (Ridge)*

IMPERIAL

– Ridge regression sum of squared weight values as the penalty term in the optimisation function.

$$\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 + \lambda \sum_{i=1}^{n} {\beta_i}^2$$

61

61

Lasso and Ridge

IMPERIAL

– The key difference between these two techniques is that lasso shrinks the less important feature's coefficient to zero, thus removing some features altogether.

– In other words, $\ell 1$ regularisation works well for feature selection in case we have a large number of features.

– Ridge reduces the complexity of the model by shrinking the coefficient (penalising higher weights).

62

62

If you have any questions

IMPERIAL

- Please feel free to arrange a meeting or email (p.barnaghi@imperial.ac.uk).
- My office: 928, Sir Michael Uren Research Hub, White City Campus.

63

63