Class 1: Revision on some basics of machine learning

Andrew Parnell
andrew.parnell@mu.ie

PRESS RECORD

https://andrewcparnell.github.io/intermediate_ML

Let's get started

- Introduction from Oliver Hooker, PR Statistics
- Tell me:
  - who you are,
  - where you are from,
  - your previous experience in working with R and machine learning,
  - what you are working on,
  - what you want to get out of the course,
- Timetable for the week
- Pre-requisites

How this course works

- This course lives on GitHub, which means anyone can see the slides, code, etc, and make comments on it
- The timetable document (`index.html`) provides links to all the pdf slides and practicals
- The slides and the practicals are all written in `Rmarkdown` format, which means you can load them up in Rstudio and see how everything was created
- Let me know if you spot mistakes, as these can be easily updated on the GitHub page
- There is an `intermediate_ML.Rproj` R project file from which you should be able to run all the code

Copyright statement

All the non-GitHub materials provided in the Intermediate to advanced machine learning in R course are copyright of Andrew Parnell.

This means:

- As a user (the student) you have permission (licence) to access the materials to aid and support your individual studies.
- You are not permitted to copy or distribute any materials without the relevant permission
- We may reserve the right to remove a user in the event of any possible infringement

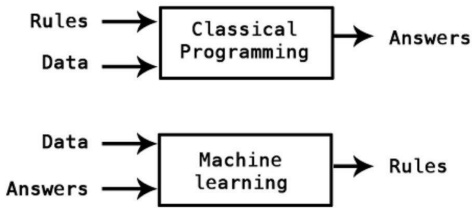
Course format and other details

- Lectures and guided coding classes will take place in the morning via Zoom, practical classes in the afternoon
- In the practical classes I will go round the room asking people how they are getting on
- If you want to send me a private message use Slack
- Please ask lots of questions, but **MUTE YOUR MICROPHONE** when not asking them

Reminder: machine learning

Machine Learning (ML) / Artificial Intelligence (AI) involves the use of algorithms and statistical models to allow computers to perform a task without explicit instructions.

The main goal is not to teach the computer the rules of what to do but to give it many examples of what you want and let it figure out the rules.



From https://friendsofyates.org/kcyo.php?cname=explanation+of+machine+learning&cid=7

Common nomenclature in ML

- Features: Input variables used to make predictions.
- Target/Label: Desired output for supervised learning.
- Training data: Data used to train the model.
- Test data: Data used to evaluate the model's performance.
- Model: Algorithm applied to the data.
- Loss function: Measures how well the model's predictions match the true target values or labels.
- Training: Process of adjusting model parameters to minimize loss.

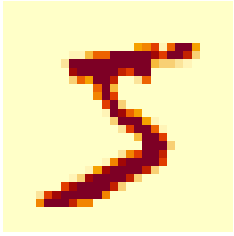
Types of machine learning

- **Supervised Learning**: Learn from labeled data to predict outcomes.
- **Unsupervised Learning**: Discover hidden patterns in unlabeled data.
- **Reinforcement Learning**: Learn by interacting with an environment and receiving feedback.

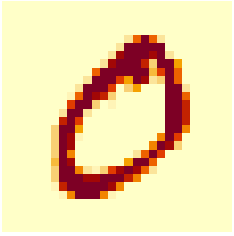
In supervised learning (the majority of the focus of this course) we minimise a loss function $L(y, x)$ to estimate $\hat{f}$ using target variable y and features x. The structure of f can be more or less anything we like

A popular example

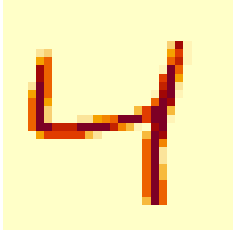
Label: 5



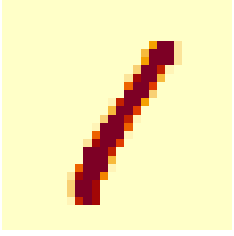
Label: 0



Label: 4



Label: 1



Common ML techniques

- Decision Trees / Random Forests: Flowchart-like structures for decision-making.
- Support Vector Machines: Find the hyperplane that best divides datasets.
- Gradient boosting: Adding together weaker learners (usually decision trees)
- **Neural Networks**: Creates a network of inter-dependent weights that are learnt to produce predictions

Supervised learning divided up into:

- Regression: Target variable is continuous.
- Classification: Target variable is discrete

Focussing on neural networks

- The majority of this course will be focussed on how to use neural networks to perform supervised and unsupervised learning

We will cover:

- Deep Learning: Neural networks with many layers.
- Convolutional Neural Networks (CNNs): Specialised for image data.
- Recurrent Neural Networks (RNNs): Specialised for time series
- Transformer models: specialised for text analysis (but also for time series in general)

Some special NN terminology

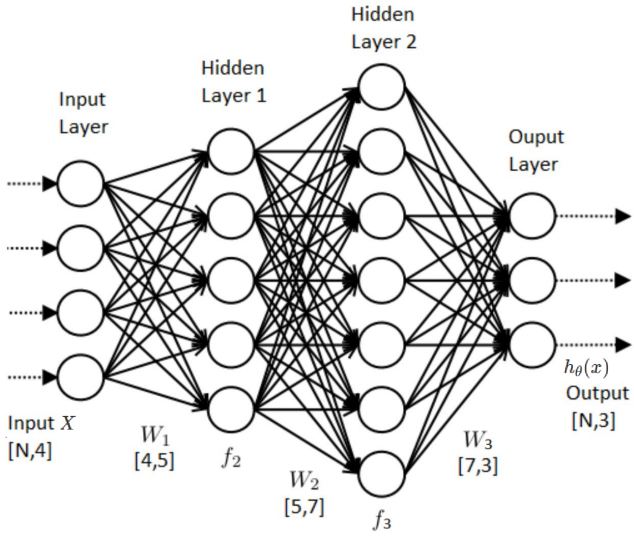
Neural networks have:

- Input Layer: Receives raw data.
- Hidden Layers: Layers between input and output. Process inputs from previous layers.
- Output Layer: Provides the final prediction/result.
- Weights and biases: Key parameters that tell you how you did

Train a neural network by:

- Backpropagation: adjusts weights based on the error in prediction.
- Setting the learning rate: determines the step size during weight updates.
- Setting the number of epochs: the number of times the training data is shown to the network.
- Setting the batch size: the number of training samples used per iteration

NNs in pictures



From here

How to fit a neural network

1. Data Prep: training/test split, feature extraction
2. Model Initialization: guess initial parameter values
3. Forward Propagation: pass input data through the network to get a prediction
4. Loss Computation: work out how badly you did
5. Backpropagation: calculate how you need to change the weights to reduce the loss
6. Weight Update: use an optimisation algorithm to estimate new weight values
7. Repeat!

Steps 2 - 7 are all done by the package but we will go through how they all work (in principle)

Let's get to a simple regression example

```
data(mtcars)
data <- mtcars %>% select(hp, wt, mpg) %>%
  scale() %>%
  as.data.frame()
str(data) # Target is mpg
```

```
## 'data.frame':    32 obs. of  3 variables:
##  $ hp : num  -0.535 -0.535 -0.783 -0.535 0.413 ...
##  $ wt : num  -0.6104 -0.3498 -0.917 -0.0023 0.2277 ...
##  $ mpg: num  0.151 0.151 0.45 0.217 -0.231 ...
```

Data prep

```r
set.seed(123) # For reproducibility
train_indices <- sample(1:nrow(data), 0.8 * nrow(data))
train_data <- data[train_indices, ]
test_data <- data[-train_indices, ]
x_train <- train_data %>% select(hp, wt) %>% as.matrix()
y_train <- train_data$mpg
x_test <- test_data %>% select(hp, wt) %>% as.matrix()
y_test <- test_data$mpg
```

Compile the model

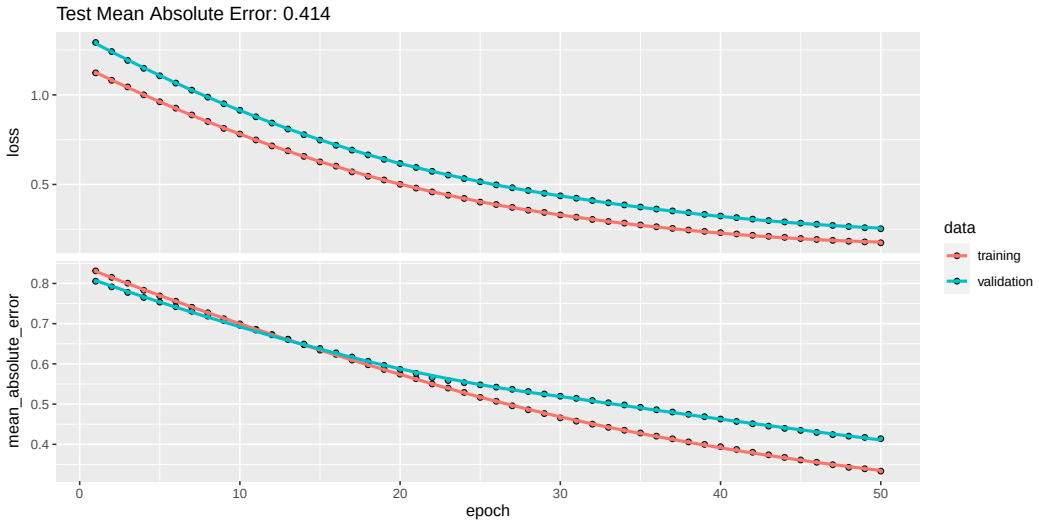
```
library(keras)
model <- keras_model_sequential() %>%
  layer_dense(units = 32,
              activation = 'relu',
              input_shape = 2) %>%
  layer_dense(units = 1)

model %>% compile(
  optimizer = "adam",
  loss = "mse",
  metrics = c("mean_absolute_error")
)
```

Train the model

```
history <- model %>% fit(
  x_train, y_train,
  epochs = 50, # Number of passes through the training data
  batch_size = 16, # Number of samples to train on
  validation_data = list(x_test, y_test) # Just for plotting
)
```

e.g. if you have a training dataset of 1000 samples and you set the batch size to 100, the model will take 10 iterations to complete one epoch

Evaluate performance



Now a simple classification example

```
data(iris)
encoded_species <- to_categorical(as.numeric(iris$Species) - 1)
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1
```

Set up training and test data

```r
# Split data into training and test sets
set.seed(123) # For reproducibility
train_indices <- sample(1:nrow(iris), 0.8 * nrow(iris))
train_data <- iris[train_indices, ]
test_data <- iris[-train_indices, ]

x_train <- train_data %>%
  select(Sepal.Length, Sepal.Width,
         Petal.Length, Petal.Width) %>%
  scale() %>% as.matrix()
y_train <- encoded_species[train_indices, ]

x_test <- test_data %>%
  select(Sepal.Length, Sepal.Width,
         Petal.Length, Petal.Width) %>%
  scale %>% as.matrix()
y_test <- encoded_species[-train_indices, ]
```

Set up and compile the model

```
model <- keras_model_sequential() %>%
  layer_dense(units = 32, activation = 'relu',
              input_shape = 4) %>%
  layer_dense(units = 3, activation = 'softmax')

model %>% compile(
  optimizer = "adam",
  loss = "categorical_crossentropy",
  metrics = "accuracy"
)
```

Train the model

```
history <- model %>% fit(
  x_train, y_train,
  epochs = 50,
  batch_size = 16,
  validation_data = list(x_test, y_test)
)
```

Assess the results

```
predictions <- model %>% predict(x_test)
colnames(predictions) <- colnames(y_test) <- levels(iris$Species)
predictions_final <- colnames(predictions)[max.col(predictions)]
y_test_final <- colnames(y_test)[max.col(y_test)]
confusion_matrix <- table(Predicted = predictions_final,
                          Actual = y_test_final)
```

```
print(confusion_matrix)
```

```
##             Actual
## Predicted    setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0          7         0
##   virginica       0          8         5
```

Some topics we don't cover

There are some advanced methods that are very new or not well implemented in R:

- Transfer learning,
- Building your own transformer
- Bayesian optimisation,
- Running models in GPUs
- Generative adversarial networks
- ...

You can do all of these things in R, but they can get quite fiddly

Summary

- We will focus on fitting neural networks to regression models (including image data), time series models, and classification models
- We will use the `keras` package to do most of the work for us
- Most of the clever part is in the model definition
- In the next class we will go through what is happening underneath the hood