Perceptrons and stacking

John Paul Gosling

2024-11-21

In this practical, we will be looking at the mechanics behind perceptrons and stacking. We will start by building a simple perceptron model and then move on to stacking multiple models together to improve performance.

Perceptrons

Let's begin by training a perceptron model on the `weather_classification_data` that we met in Practical 2.

```
# Load the data
weather_full <- read.csv("https://www.maths.dur.ac.uk/users/john.p.gosling/MATH3431_practicals/weather_c

# Display the first few rows
head(weather_full)
```

```
##   Temperature Humidity Wind.Speed Precipitation....   Cloud.Cover
## 1          14       73        9.5                82 partly cloudy
## 2          39       96        8.5                71 partly cloudy
## 3          30       64        7.0                16         clear
## 4          38       83        1.5                82         clear
## 5          27       74       17.0                66      overcast
## 6          32       55        3.5                26      overcast
##   Atmospheric.Pressure UV.Index Season Visibility..km. Location Weather.Type
## 1              1010.82        2 Winter             3.5   inland        Rainy
## 2              1011.43        7 Spring            10.0   inland       Cloudy
## 3              1018.72        5 Spring             5.5 mountain        Sunny
## 4              1026.25        7 Spring             1.0  coastal        Sunny
## 5               990.67        1 Winter             2.5 mountain        Rainy
## 6              1010.03        2 Summer             5.0   inland       Cloudy
```

```
# Select the features of interest
weather <- weather_full[,c(1:6)]

# Pick 1000 random rows
set.seed(1312)
weather <- weather[sample(1:nrow(weather), 1000),]

# Convert Cloud.Cover to a binary variable (clear vs not)
weather$Cloud.Cover <- ifelse(weather$Cloud.Cover == "clear", 1, -1)

# Add in a variable for a constant term
weather <- cbind(weather,1)

# Summarise the data
```

```r
summary(weather)
```

```
##   Temperature        Humidity        Wind.Speed     Precipitation....
##  Min.   :-22.00   Min.   : 20.00   Min.   : 0.000   Min.   :  0.00
##  1st Qu.:  4.00   1st Qu.: 58.00   1st Qu.: 5.000   1st Qu.: 20.00
##  Median : 21.00   Median : 70.00   Median : 9.000   Median : 59.00
##  Mean   : 18.78   Mean   : 68.97   Mean   : 9.881   Mean   : 53.57
##  3rd Qu.: 30.00   3rd Qu.: 83.00   3rd Qu.:13.500   3rd Qu.: 80.25
##  Max.   : 91.00   Max.   :109.00   Max.   :44.000   Max.   :109.00
##   Cloud.Cover     Atmospheric.Pressure       1
##  Min.   :-1.000   Min.   : 803.3       Min.   :1
##  1st Qu.:-1.000   1st Qu.: 994.3       1st Qu.:1
##  Median :-1.000   Median :1007.3       Median :1
##  Mean   :-0.674   Mean   :1004.1       Mean   :1
##  3rd Qu.:-1.000   3rd Qu.:1016.1       3rd Qu.:1
##  Max.   : 1.000   Max.   :1198.4       Max.   :1
```

We will also split the data into a training and testing set (70/30).

```r
# Set the seed
set.seed(141)

# Split the data
train_indices <- sample(1:nrow(weather), 0.7 * nrow(weather))
train_data <- weather[train_indices, ]
test_data <- weather[-train_indices, ]
```

Task 1.1 - Build your own perceptron

Build a perceptron model that predicts the `class` variable using the other variables as predictors. You should use the base R strategy given in the notes.

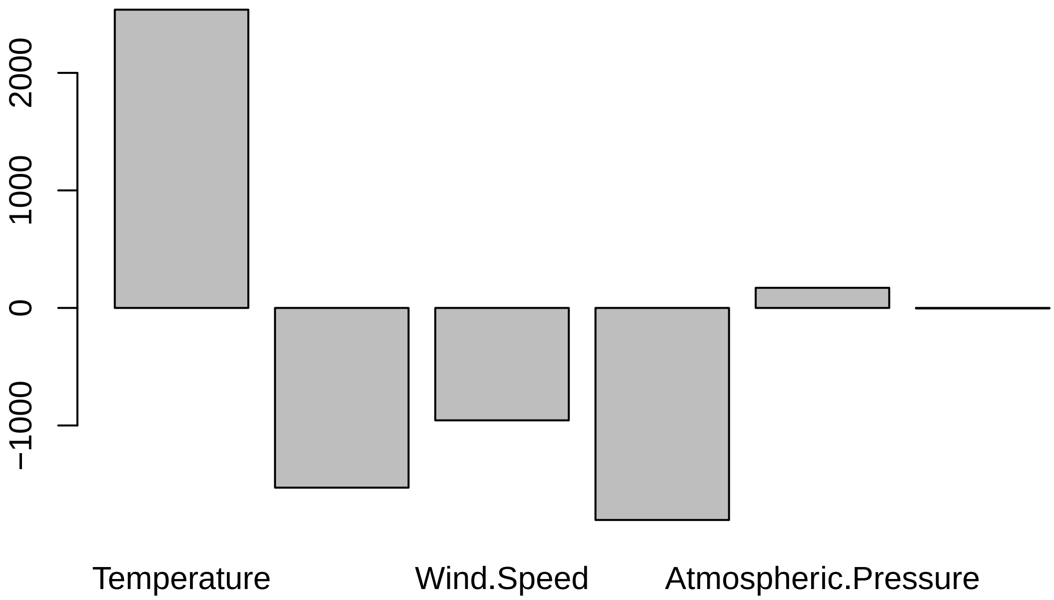
```r
# Initialise the weights to zero
weights <- rep(0, ncol(weather) - 1)

# Set the learning rate
alpha <- 0.1

# Set the maximum number of iterations
max_iter <- 30

# Repeat the following steps until the maximum number of
# iterations is reached
for (i in 1:max_iter) {
  # For each input in the training data
  for (j in 1:nrow(train_data)) {
    # Compute the predicted class label
    predicted <- ifelse(sum(weights * train_data[j, -5]) > 0,
                        1, -1)
    # Update the weights based on the classification error
    weights <- weights + alpha * (train_data[j, 5] - predicted) *
      train_data[j, -5]
  }
}
```

Visualise the weights.

```r
weights
```

```
##      Temperature Humidity Wind.Speed Precipitation.... Atmospheric.Pressure  1
## 1471      2537.8    -1529     -956.5             -1804              171.268 -4
```

```r
barplot(as.numeric(weights), names.arg = colnames(train_data)[-5])
```



Make predictions on the test data and evaluate the model's performance using accuracy.

```r
# Make predictions
predictions <- NULL
for (j in 1:nrow(test_data)) {
  predictions[j] <- ifelse(sum(weights * test_data[j, -5]) > 0,
                           1, -1)
}

# Calculate the accuracy
accuracy <- sum(predictions == test_data[,5]) / nrow(test_data)
accuracy
```

```
## [1] 0.63
```

Task 1.2 - Perceptron using standardised data

Create a standardised version of the five explanatory variables and repeat the above steps.

```r
# Standardise the data by subtracting the mean
# and dividing by the standard deviation
standardised_weather <- scale(weather[, -c(5,7)])

# Combine the standardised data with the class variable
# and the constant term
standardised_weather <- cbind(standardised_weather, weather[, c(5,7)])

# Split the data
set.seed(141)
train_indices <- sample(1:nrow(standardised_weather), 0.7 * nrow(standardised_weather))
```

```r
train_data <- standardised_weather[train_indices, ]
test_data <- standardised_weather[-train_indices, ]

# Initialise the weights to zero
weights <- rep(0, ncol(standardised_weather) - 1)

# Set the learning rate
alpha <- 0.1

# Set the maximum number of iterations
max_iter <- 30

# Repeat the following steps until the maximum number of
# iterations is reached
for (i in 1:max_iter) {
  # For each input in the training data
  for (j in 1:nrow(train_data)) {
    # Compute the predicted class label
    predicted <- ifelse(sum(weights * train_data[j, -6]) > 0,
                        1, -1)
    # Update the weights based on the classification error
    weights <- weights + alpha * (train_data[j, 6] - predicted) *
      train_data[j, -6]
  }
}

# Make predictions
predictions <- NULL
for (j in 1:nrow(test_data)) {
  predictions[j] <- ifelse(sum(weights * test_data[j, -6]) > 0,
                           1, -1)
}

# Calculate the accuracy
std_accuracy <- sum(predictions == test_data[,6]) / nrow(test_data)
std_accuracy
```

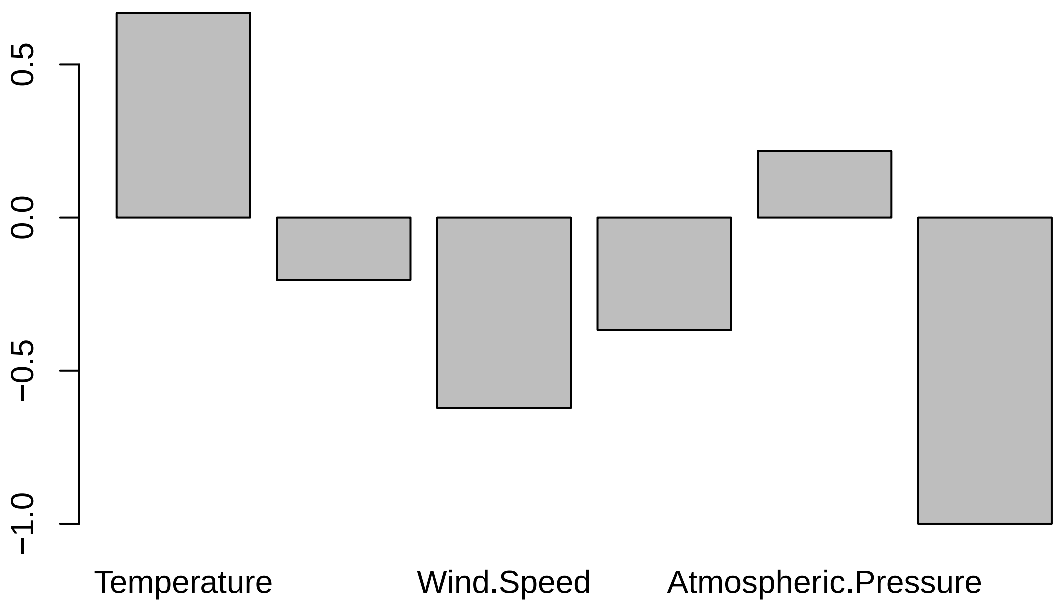
```
## [1] 0.8133333
```

Is this transformation necessary? Is it beneficial?

```r
weights
```

```
##      Temperature   Humidity Wind.Speed Precipitation.... Atmospheric.Pressure
## 1471   0.6679704 -0.2037378 -0.6222185        -0.3669384            0.2171479
##       1
## 1471 -1
```

```r
barplot(as.numeric(weights), names.arg = colnames(train_data)[-6])
```



It is probably worthwhile to standardise the data as it makes the weights more interpretable. In this case, the accuracy has also improved.

Task 2 - Changing the parameters

Repeat the above steps for the original data but try different learning rates and maximum iterations.

```
# Set the seed
set.seed(141)

# Split the data
train_indices <- sample(1:nrow(weather), 0.7 * nrow(weather))
train_data <- weather[train_indices, ]
test_data <- weather[-train_indices, ]

# Initialise the weights to zero
weights <- rep(0, ncol(weather) - 1)

# Set the learning rate
alpha <- 0.01

# Set the maximum number of iterations
max_iter <- 100

# Repeat the following steps until the maximum number of
# iterations is reached
for (i in 1:max_iter) {
  # For each input in the training data
  for (j in 1:nrow(train_data)) {
    # Compute the predicted class label
    predicted <- ifelse(sum(weights * train_data[j, -5]) > 0,
                        1, -1)
    # Update the weights based on the classification error
    weights <- weights + alpha * (train_data[j, 5] - predicted) *
      train_data[j, -5]
```

```r
  }
}

# Make predictions
predictions <- NULL
for (j in 1:nrow(test_data)) {
  predictions[j] <- ifelse(sum(weights * test_data[j, -5]) > 0,
                           1, -1)
}

# Calculate the accuracy
accuracy_try <- sum(predictions == test_data[,5]) / nrow(test_data)
```

Have things improved?

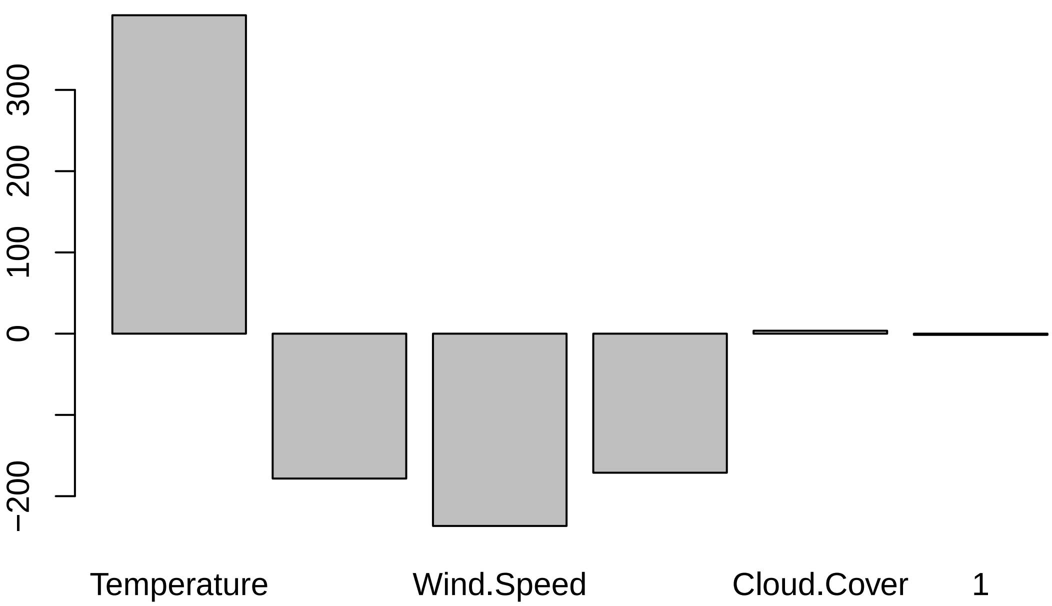
```r
accuracy_try
```

```
## [1] 0.88
```

```r
weights
```

```
##      Temperature Humidity Wind.Speed Precipitation.... Atmospheric.Pressure
## 1471       391.9   -178.3    -236.67           -171.16                 3.62
##          1
## 1471 -1.48
```

```r
barplot(as.numeric(weights), names.arg = colnames(train_data)[-6])
```



To get a handle on what is going on, consider the interplay between `alpha` and the standardisation being performed on the data.

Stacking

Let's go back to the `Glass` dataset that we first met in Practical 1. We will use this dataset to build a stacking model for the `RI` response variable.

```r
# Load in the data
Glass <- read.csv("https://www.maths.dur.ac.uk/users/john.p.gosling/MATH3431_practicals/Glass.csv")

# Look at the first few rows
head(Glass)
```

```
##        RI    Na   Mg   Al    Si    K   Ca Ba   Fe Type
## 1 1.52101 13.64 4.49 1.10 71.78 0.06 8.75  0 0.00    1
## 2 1.51761 13.89 3.60 1.36 72.73 0.48 7.83  0 0.00    1
## 3 1.51618 13.53 3.55 1.54 72.99 0.39 7.78  0 0.00    1
## 4 1.51766 13.21 3.69 1.29 72.61 0.57 8.22  0 0.00    1
## 5 1.51742 13.27 3.62 1.24 73.08 0.55 8.07  0 0.00    1
## 6 1.51596 12.79 3.61 1.62 72.97 0.64 8.07  0 0.26    1
```

```r
# Let's split the data into a training and testing set (70/30)
set.seed(123)
train_indices <- sample(1:nrow(Glass), 0.7 * nrow(Glass))
train_data <- Glass[train_indices, ]
test_data <- Glass[-train_indices, ]
```

Task 3 - Building poor models

Let's start by building four weak learners.

- **Model 1** A linear regression utilising just `Na` and `Mg` as predictors.
- **Model 2** A linear regression utilising just `Al` as a predictor with no intercept term.
- **Model 3** A 1-NN model utilising just `Ca`, `Ba` and `Fe`.
- **Model 4** A decision tree model utilising all variables but with a maximum depth of 2.

Task 3.1 - Model 1 Build the model and evaluate its performance on the test data (MSE and MAE).

```r
model_1 <- lm(RI ~ Na + Mg,
              data = train_data)

# Make predictions
predictions_1 <- predict(model_1,
                         newdata = test_data)

# Calculate the MSE and MAE
mse_1 <- mean((test_data$RI - predictions_1)^2)
mae_1 <- mean(abs(test_data$RI - predictions_1))
```

Task 3.2 - Model 2 Build the model and evaluate its performance on the test data (MSE and MAE).

```r
model_2 <- lm(RI ~ Al - 1,
              data = train_data)

# Make predictions
predictions_2 <- predict(model_2,
                         newdata = test_data)

# Calculate the MSE and MAE
mse_2 <- mean((test_data$RI - predictions_2)^2)
mae_2 <- mean(abs(test_data$RI - predictions_2))
```

Task 3.3 - Model 3 Build the model and evaluate its performance on the test data (MSE and MAE).

```
library(caret)

model_3 <- train(RI ~ Ca + Ba + Fe,
                 method = "knn",
                 data = train_data)

# Make predictions
predictions_3 <- predict(model_3,
                         newdata = test_data)

# Calculate the MSE and MAE
mse_3 <- mean((test_data$RI - predictions_3)^2)
mae_3 <- mean(abs(test_data$RI - predictions_3))
```

Task 3.4 - Model 4 Build the model and evaluate its performance on the test data (MSE and MAE).

```
library(rpart)

model_4 <- rpart(RI ~ .,
                 data = train_data,
                 method = "anova",
                 control = rpart.control(maxdepth = 2))

# Make predictions
predictions_4 <- predict(model_4,
                         newdata = test_data)

# Calculate the MSE and MAE
mse_4 <- mean((test_data$RI - predictions_4)^2)
mae_4 <- mean(abs(test_data$RI - predictions_4))
```

Which model is best so far?

Model	MSE	MAE
1	9.4×10^{-6}	0.00225
2	0.1947849	0.34292
3	3.2×10^{-6}	0.00121
4	4.9×10^{-6}	0.00152

Task 4 - Stacking models

Now we will stack the models together to see if we can improve performance.

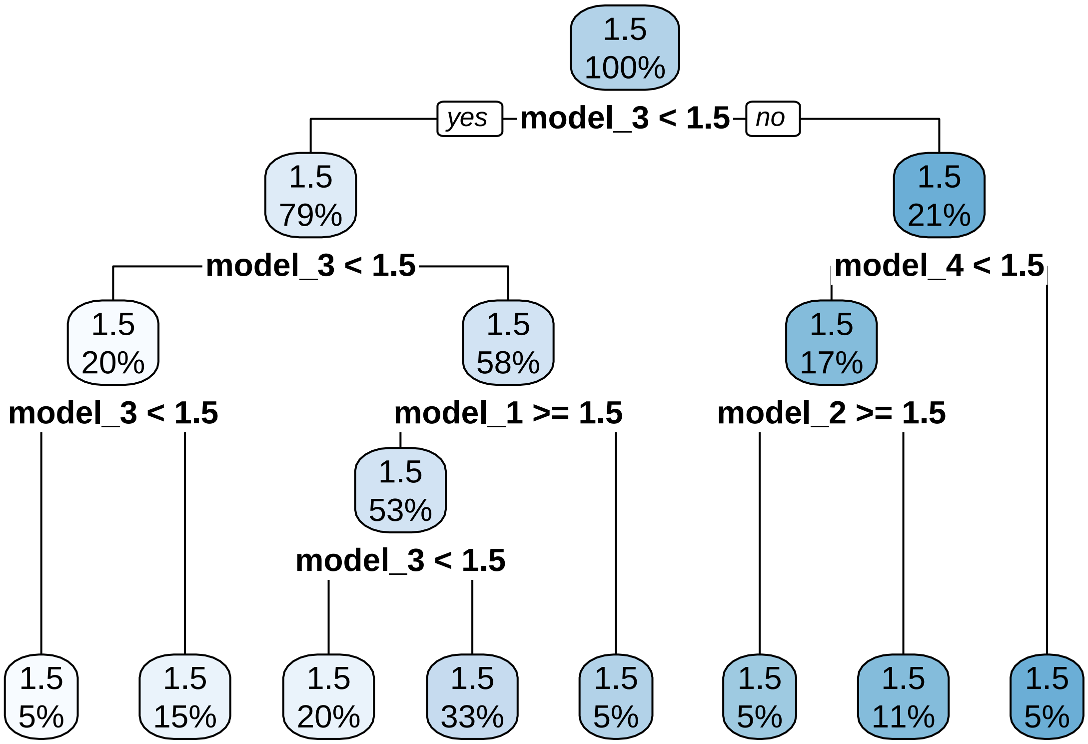
Task 4.1 - Build the meta-model Build a decision tree model that takes the predictions from the four weak learners as input. We want the possibility of a more detailed model so set the maximum depth to 5.

```
# Create a new data frame with the predictions from the training data alongside
# the actual response variable
stacking_data <- data.frame(model_1 = predict(model_1),
                            model_2 = predict(model_2),
                            model_3 = predict(model_3),
                            model_4 = predict(model_4),
```

```
                            RI = train_data$RI)

# Build the meta-model
stacking_model <- rpart(RI ~ .,
                        data = stacking_data,
                        method = "anova",
                        control = rpart.control(maxdepth = 5))
```

Plot the tree.

```
library(rpart.plot)

rpart.plot(stacking_model)
```



What is notable about the tree?

It ignores the predictions from model 1 and model 4.

Task 4.2 - Evaluate the meta-model Make predictions using the meta-model and evaluate its performance on the test data (MSE and MAE).

```
# Create the test set by combining the predictions from the weak learners
# alongside the actual response variable
stacking_test_data <- data.frame(model_1 = predictions_1,
                                 model_2 = predictions_2,
                                 model_3 = predictions_3,
                                 model_4 = predictions_4,
                                 RI = test_data$RI)

# Make predictions
stacking_predictions <- predict(stacking_model,
```

```r
                                        newdata = stacking_test_data)

# Calculate the MSE and MAE
stacking_mse <- mean((test_data$RI - stacking_predictions)^2)
stacking_mae <- mean(abs(test_data$RI - stacking_predictions))
```

How does the meta-model perform compared to the individual models?

Model	MSE	MAE
1	9.4×10^{-6}	0.00225
2	0.1947849	0.34292
3	3.2×10^{-6}	0.00121
4	4.9×10^{-6}	0.00152
Stacking	4.3×10^{-6}	0.00144

Given these results, what model would you recommend using for this dataset?

It would seem that the stacked model is very close to the performance of models 3 and 4. Given this, I would recommend using model 3 as it is the simplest model of the three.