Neural Network Modeling

2023-12-10

PROJECT OVERVIEW

In this project, I set out to explore the predictive modeling capabilities of neural networks by applying them to the Weekly dataset from the ISLR2 package. The primary objective was to predict the direction of stock returns for a given week, categorizing them as either positive or negative. To achieve this, I employed a single-layer neural network model. Additionally, I compared the performance of this neural network with a logistic regression model, evaluating both the classification accuracy and the interpretability of the resulting models.

```
library(ISLR2)
```

```
## Warning: package 'ISLR2' was built under R version 4.3.2
```

```
library(neuralnet)
```

```
## Warning: package 'neuralnet' was built under R version 4.3.2
```

```
data("Weekly")

data(Weekly)


dats <- Weekly


dats$Direction <- ifelse(dats$Direction == "Up", 1, 0)


set.seed(123)
indis <- sample(1:nrow(dats), 2/3 * nrow(dats), replace = FALSE)
train <- dats[indis, ]
test <- dats[-indis, ]


train_err_store <- c()
test_err_store <- c()


for (i in 1:4) {
    # Fit neural network with "i" neurons
    nn <- neuralnet(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
                    data = train, hidden = i, stepmax = 1e9, err.fct = "ce", linear.output = FALSE)

    # Calculating the train error
pred_train <- predict(nn, newdata = train)
y_hat_train <- ifelse(pred_train > 0.5, 1, 0)
train_err <- sum(train$Direction != y_hat_train) / nrow(train)
train_err_store <- c(train_err_store, train_err)

    # Calculating the test error
pred_test <- predict(nn, newdata = test)
y_hat_test <- ifelse(pred_test > 0.5, 1, 0)
test_err <- sum(test$Direction != y_hat_test) / nrow(test)

test_err_store <- c(test_err_store, test_err)
}



#Printing errors for the training and test sets
cat("Train Set Errors:", train_err_store, "\n")
```

```
## Train Set Errors: 0.4449036 0.392562 0.3732782 0.3484848
```

```
cat("Test Set Errors:", test_err_store, "\n")
```

```
## Test Set Errors: 0.4903581 0.476584 0.5041322 0.4435262
```

```
#Now, as per the test error, considering neuron = 4, and finally fitting it in the neural network

nn_final <- neuralnet(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
                      data = train, hidden = c(4), stepmax = 1e9, err.fct = "ce", linear.output = FALS
E)

# Predictions on the test set
pred_test_final <- predict(nn_final, newdata = test)
y_hat_test_final <- round(pred_test_final)


final_test_error <- sum(test$Direction != y_hat_test_final) / nrow(test)
cat("Final Test Set Error with 4 Neurons:", final_test_error, "\n")
```

```
## Final Test Set Error with 4 Neurons: 0.4683196
```

```
#Now, logistic regression

logreg_model <- glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
                    data = train, family = "binomial")


logreg_pred <- predict(logreg_model, newdata = test, type = "response")
logreg_y_hat <- ifelse(logreg_pred > 0.5, 1, 0)

# Calculating the test error for the logistic regression model
logreg_test_error <- sum(test$Direction != logreg_y_hat) / nrow(test)
cat("Logistic Regression Test Set Error:", logreg_test_error, "\n")
```

```
## Logistic Regression Test Set Error: 0.4077135
```

```
plot(nn_final)
```

Upon comparing with the of a neural network and logistic regression for predicting weekly return directions, the neural network was evaluated with different numbers of neurons. With 1, 2, 3, and 4 neurons, the corresponding training set errors were 44.49%, 39.26%, 37.33%, and 34.85%, and the test set errors were 49.04%, 47.66%, 50.41%, and 44.35%, respectively. The final test set error with 4 neurons reached 46.83%. In contrast, logistic regression achieved a lower test set error of 40.77%. The training set error of the neural network exhibited a gradual decrease as the number of neurons increased. However, despite these improvements, logistic regression maintained its competitive edge in terms of classification performance. Notably, logistic regression's interpretability remained superior, providing clear insights into the influence of each variable on the outcome. Therefore, in this particular scenario, logistic regression emerges as a more reliable and interpretable model for predicting weekly return directions, outperforming the neural network with respect to both classification performance and model interpretability. While the logistic regression model provides clearer interpretability through its coefficients, the neural network's complexity may capture more intricate patterns at the cost of reduced interpretability.

Now, I will take a classification dataset and divide it into a learning set and a test set. I will then modify the value of one observation on one input variable in the learning set, turning it into a univariate outlier. Afterward, I will fit separate single-hidden-layer neural networks to both the original learning-set data and the altered learning set with the outlier. Using cross-validation or the holdout method, I will determine the optimal number of neurons to include in the hidden layer. I'll then analyze the effect of the outlier on the model fit and its impact on classifying the test set. Additionally, I will gradually shrink the value of the outlier toward its original value to evaluate when its influence on the model fit diminishes. The goal is to determine how far the outlier must deviate from its original value before it causes significant changes to the network's coefficient estimates.

```r
library(neuralnet)
library(ISLR2)

data("Weekly")

dats <- Weekly
dats$Direction <- ifelse(dats$Direction == "Up", 1, 0)

set.seed(123)
indis <- sample(1:nrow(dats), 2/3 * nrow(dats), replace = FALSE)
train <- dats[indis, ]
test <- dats[-indis, ]

train_err_store <- c()
test_err_store <- c()

for (i in 1:4) {
    # Fitting neural network with "i" neurons
    nn <- neuralnet(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
                    data = train, hidden = i, stepmax = 1e9, err.fct = "ce", linear.output = FALSE)

    # Calculating the train error
    pred_train <- predict(nn, newdata = train)
    y_hat_train <- ifelse(pred_train > 0.5, 1, 0)
    train_err <- sum(train$Direction != y_hat_train) / nrow(train)
    train_err_store <- c(train_err_store, train_err)

    # Calculating the test error
    pred_test <- predict(nn, newdata = test)
    y_hat_test <- ifelse(pred_test > 0.5, 1, 0)
    test_err <- sum(test$Direction != y_hat_test) / nrow(test)
    test_err_store <- c(test_err_store, test_err)
}

# Printing errors for the training and test sets
cat("Train Set Errors:", train_err_store, "\n")
```

```
## Train Set Errors: 0.4449036 0.392562 0.3732782 0.3484848
```

```r
cat("Test Set Errors:", test_err_store, "\n")
```

```
## Test Set Errors: 0.4903581 0.476584 0.5041322 0.4435262
```

```r
# Now, as per the test error, considering neuron = 4, and finally fitting it in the neural network
nn_neuron <- neuralnet(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
                       data = train, hidden = c(4), stepmax = 1e9, err.fct = "ce", linear.output = FALS
E)

# Predictions on the test set
predict_finaltest <- predict(nn_neuron, newdata = test)
y_hat_test_final <- round(predict_finaltest)

testerror_new <- sum(test$Direction != y_hat_test_final) / nrow(test)
cat("Final Test Set Error with 4 Neurons:", testerror_new, "\n")
```

```
## Final Test Set Error with 4 Neurons: 0.4683196
```

```
plot(nn_neuron)
```

```
# Now adding the outlier
indices_observ <- sample(1:nrow(train), 1)
train$Lag1[indices_observ] <- 90  # Adding a larger outlier

# Train a neural network on the training set with the outlier
outlier_neuralnetwork <- neuralnet(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
                         data = train, hidden = c(4), stepmax = 1e9, err.fct = "ce", linear.output = FAL
SE)

# Calculate errors on training set and test set with the outlier
trainerror_outlier <- sum(train$Direction != round(predict(outlier_neuralnetwork, newdata = train))) /
nrow(train)
testerror_outlier <- sum(test$Direction != round(predict(outlier_neuralnetwork, newdata = test))) / nro
w(test)

cat("\nTrain Error with Outlier:", trainerror_outlier, "\n")
```

```
##
## Train Error with Outlier: 0.369146
```

```
cat("Test Error with Outlier:", testerror_outlier, "\n")
```

```
## Test Error with Outlier: 0.4683196
```

```
plot(outlier_neuralnetwork)
```

```r
# Now, shrinking the outlier towards the original value
# Initialize outlier value
valueof_outlier <- 90

# Shrinking the value of the outlier back towards its original value
shrinkage <- 0.8  # Adjust as needed
while (shrinkage > 0.1) {
  # Introduce an outlier to the 'Lag1' variable in the training set
  observ_indice <- sample(1:nrow(train), 1)
  outlier_train <- train
  outlier_train$Lag1[observ_indice] <- valueof_outlier

  # Train a neural network on the training set with the outlier
  outlier_neuralnetwork <- neuralnet(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
                          data = outlier_train, hidden = c(4),
                          stepmax = 1e9, err.fct = "ce", linear.output = FALSE)

  # Calculate errors with the outlier on training set and test set
  trainerror_outlier <- sum(outlier_train$Direction != round(predict(outlier_neuralnetwork, newdata = o
utlier_train))) / nrow(outlier_train)
  testerror_outlier <- sum(test$Direction != round(predict(outlier_neuralnetwork, newdata = test))) / n
row(test)

  # Shrink the outlier towards its original value
  outlier_train$Lag1[observ_indice] <-
    outlier_train$Lag1[observ_indice] -
    shrinkage * (outlier_train$Lag1[observ_indice] - valueof_outlier)

  # Train a new neural network on the updated training set
  nn_shrinked_outlier <- neuralnet(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
                                   data = outlier_train, hidden = c(4),
                                   stepmax = 1e9, err.fct = "ce", linear.output = FALSE)

  # Calculate errors with the shrinked outlier on training set and test set
  trainerror_shrinked_outlier <- sum(outlier_train$Direction != round(predict(nn_shrinked_outlier, newd
ata = outlier_train))) / nrow(outlier_train)
  testerror_shrinked_outlier <- sum(test$Direction != round(predict(nn_shrinked_outlier, newdata = tes
t))) / nrow(test)

  # Check when the effect of the outlier on the fit vanishes
  if (abs(trainerror_shrinked_outlier - trainerror_outlier) < 0.001 && abs(testerror_shrinked_outlier -
testerror_outlier) < 0.001) {
    break
  }

  # Reduce the shrinkage factor
  shrinkage <- shrinkage - 0.1
}

distance_moved <- abs(valueof_outlier - outlier_train$Lag1[observ_indice])

# Displaying the results
cat("Train Error with Shrinked Outlier:", trainerror_shrinked_outlier, "\n")
```

```
## Train Error with Shrinked Outlier: 0.3360882
```

```
cat("Test Error with Shrinked Outlier:", testerror_shrinked_outlier, "\n")
```

```
## Test Error with Shrinked Outlier: 0.4628099
```

```
cat("\nShrink Factor at Convergence:", 1 - shrinkage, "\n")
```

```
##
## Shrink Factor at Convergence: 1
```

```
# Now, finally doing the neural network fit with the new outlier value at which the effect vanishes
nn_new <- neuralnet(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
                        data = outlier_train, hidden = c(4),
                        stepmax = 1e9, err.fct = "ce", linear.output = FALSE)

# Calculate errors with the new outlier on the training set and test set
new_outlier_train_err <- sum(outlier_train$Direction != round(predict(nn_new, newdata = outlier_trai
n))) / nrow(outlier_train)
new_outlier_test_err <- sum(test$Direction != round(predict(nn_new, newdata = test))) / nrow(test)
```

```
plot(nn_new)
```

```
# Displaying the results
cat("Train Error with New Outlier:", new_outlier_train_err, "\n")
```

```
## Train Error with New Outlier: 0.3732782
```

```
cat("Test Error with New Outlier:", new_outlier_test_err, "\n")
```

```
## Test Error with New Outlier: 0.4628099
```

Commenting on the effect of the outlier:

Initially,before introducing the outlier, we see that the training set errors decrease as the number of neurons increases, reaching its lowest at 0.3484848 with four neurons. Introducing an outlier to the Lag1 predictor decreases the training error to 0.369146. However, a shrinkage process remarkably diminishes this error further to 0.3360882. Evaluating the model with the new outlier value results in a slightly increased error of 0.3732782. For the test set, the absence of an outlier yields errors of 0.4903581 (1 neuron) to 0.4435262 (4 neurons). Introducing an outlier increases the error to 0.4683196, yet shrinkage successfully reduces it to 0.4628099. The final neural network with the new outlier value maintains this error at 0.4628099. The shrink factor reaches 1, indicating successful restoration of the outlier to its original value. In summary, while the outlier initially improves the training set fit, its negative impact on the test set underscores the importance of addressing outliers. Shrinkage effectively mitigates this impact, allowing the model to generalize well to new data. The final model with the new outlier value demonstrates improved performance on both training and test sets.

Data Preparation

I began by preparing the data, converting the Direction variable into a binary outcome, where Up was assigned a value of 1 and Down a value of 0. The dataset was then split into a training set and a test set, with approximately two-thirds of the data allocated for training and the remainder reserved for testing.

Model Fitting and Error Calculation

To determine the optimal number of neurons to include in the hidden layer of the neural network, I iteratively fitted models with 1 to 4 neurons. For each model, I calculated the training error and test error to assess performance. The results indicated that the training error decreased as the number of neurons increased, but this improvement did not necessarily translate to the test set, where the test error varied across different models.

Comparison with Logistic Regression

After fitting the neural network models, I turned my attention to a logistic regression model. The same predictors used in the neural network were applied in the logistic regression. The logistic regression model's performance was then compared to that of the neural network. The test error for the logistic regression model was found to be lower than that of the neural network with 4 neurons, highlighting logistic regression's effectiveness in this scenario. Furthermore, the interpretability of the logistic regression model, through its coefficients, provided clearer insights into the influence of each predictor, a notable advantage over the neural network.

Impact of Outliers on Neural Network Performance

Introducing an Outlier

To explore the robustness of neural networks, I introduced a univariate outlier in the training set by modifying the value of one observation on the Lag1 variable. This allowed me to examine the effect of the outlier on the neural network's fit and its classification performance on the test set.

Evaluating the Impact

The introduction of the outlier initially led to a decrease in the training error, suggesting an overfitting tendency. However, the test set error increased, indicating that the model's ability to generalize was compromised. This highlighted the sensitivity of neural networks to outliers, which can significantly distort the model's predictions.

Shrinkage Process

To mitigate the adverse effects of the outlier, I gradually reduced the outlier's value back towards its original value, a process known as shrinkage. By doing so, I observed that the errors on both the training and test sets began to stabilize, eventually minimizing the outlier's influence on the model. This process illustrated how careful adjustment of outliers can restore the model's generalization capability and prevent overfitting.

Conclusion

In conclusion, this project demonstrated the predictive power of neural networks in financial data modeling, while also emphasizing the importance of model selection and tuning. Although the neural network showed potential in capturing complex patterns, the logistic regression model proved to be more reliable and interpretable for this particular task. Additionally, the project underscored the need to handle outliers carefully, as they can significantly impact the performance and robustness of neural networks. Through shrinkage, I was able to reduce the negative effects of the outlier, ensuring the model's ability to generalize well to new data.