

Getting Started with densemlp

Contents

0.1	Classification	1
0.2	Regression	2
0.3	Training Diagnostics	3
0.4	Tuning	4
0.5	Permutation Importance	5

```
library(densemlp)
```

densemlp trains dense feed-forward multilayer perceptrons for tabular data. It accepts a formula and data frame, preprocesses numeric and categorical predictors, and returns a **densemlp_fit** object with methods for prediction, plotting, metrics, tuning, and variable importance.

The models implemented in **densemlp** are dense feed-forward multilayer perceptrons. Optional components such as dropout, batch normalization, residual connections, gated blocks, and input projection extend the basic dense multilayer perceptron architecture but do not change the model class into a convolutional, recurrent, transformer, or tree-based model.

task is optional in the main API. When it is set to "auto" or omitted, the task is inferred from the outcome. Use it only when you need to override that inference.

This vignette documents version 0.5.0.

This vignette assumes the package is installed or loaded with `pkgload::load_all(".")`. Do not source individual files such as `R/densemlp.R` because exported functions rely on helpers loaded through the package namespace.

densemlp implements dense feed-forward multilayer perceptrons for tabular data. Each hidden block is centered on a fully connected transformation, optionally combined with activation functions, dropout, batch normalization, residual connections, gated mechanisms, and input projection.

0.1 Classification

For classification, use a factor outcome. The task is inferred from the outcome type, so **task** can usually be omitted.

```
fit <- densemlp(
  Species ~ .,
  data = iris,
  epochs = 10,
  patience = 3,
  verbose = FALSE,
  seed = 1
)

fit
#> <densemlp_fit>
#> Task: classification
#> Outcome levels: setosa, versicolor, virginica
```

```
#> Encoded features: 4
#> Best epoch: 10
```

Class predictions are returned as factors with the same levels as the training outcome.

```
predict(fit, iris[1:5, ], type = "class")
#> [1] setosa setosa setosa setosa setosa
#> Levels: setosa versicolor virginica
round(predict(fit, iris[1:5, ], type = "prob"), 3)
#>      setosa versicolor virginica
#> [1,]  0.982      0.009      0.008
#> [2,]  0.927      0.051      0.022
#> [3,]  0.966      0.022      0.012
#> [4,]  0.934      0.045      0.021
#> [5,]  0.984      0.010      0.007
```

Use task-aware metrics to summarize predictions.

```
pred <- predict(fit, iris, type = "class")
densemlp_metrics(iris$Species, pred)
#> $accuracy
#> [1] 0.9466667
```

0.2 Regression

For regression, use a numeric outcome. The task is inferred automatically for numeric outcomes.

```
fit_reg <- densemlp(
  mpg ~ disp + hp + wt,
  data = mtcars,
  epochs = 10,
  patience = 3,
  verbose = FALSE,
  seed = 2
)

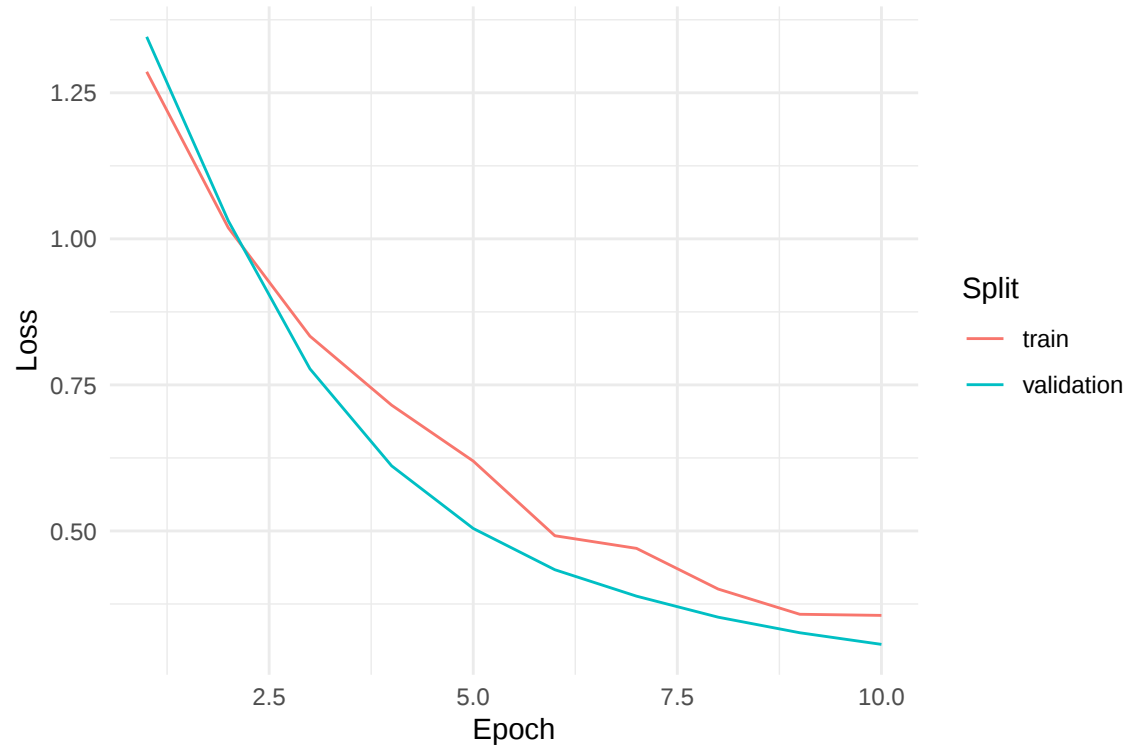
fit_reg
#> <densemlp_fit>
#> Task: regression
#> Encoded features: 3
#> Best epoch: 10

pred_reg <- predict(fit_reg, mtcars, type = "response")
head(round(pred_reg, 2))
#> [1] 26.61 24.15 31.63 22.54 20.10 23.74
densemlp_metrics(mtcars$mpg, pred_reg)
#> $rmse
#> [1] 5.818005
#>
#> $mae
#> [1] 5.026398
#>
#> $rsq
#> [1] 0.03807414
```

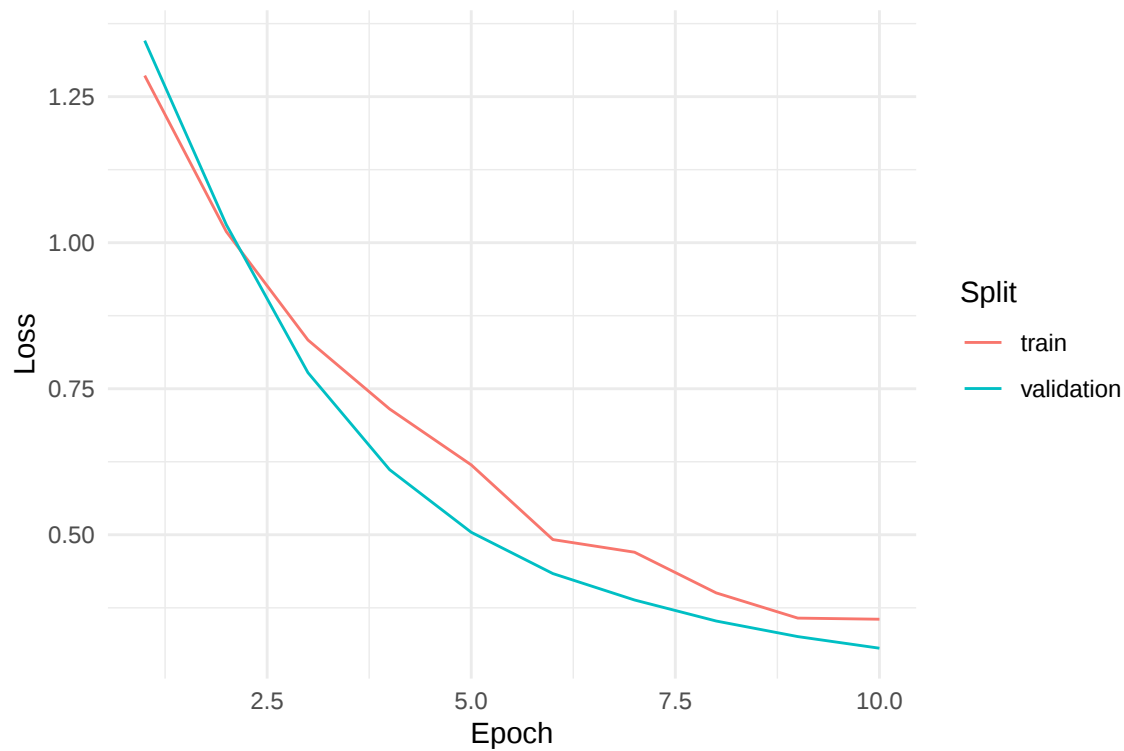
0.3 Training Diagnostics

The fitted object stores training history, which can be displayed directly or through the `ggplot2::autoplot()` method.

```
plot_history(fit)
```



```
ggplot2::autoplot(fit)
```



0.4 Tuning

`tune_densemip()` evaluates a grid of hyperparameters and, by default, refits the best configuration on the supplied data.

```
tuned <- tune_densemip(
  Species ~ .,
  data = iris,
  grid = list(
    hidden_units = list(c(8), c(16, 8)),
    activation = c("relu"),
    dropout = c(0),
    batch_size = c(8),
    lr = c(1e-3),
    epochs = c(10)
  ),
  patience = 3,
  seed = 3,
  verbose = FALSE
)
```

```
tuned$results
#>   hidden_units dropout activation batch_norm residual gated input_projection
#> 1      16-8      0      relu      TRUE    FALSE FALSE             NA
#> 2       8      0      relu      TRUE    FALSE FALSE             NA
#>   epochs batch_size  lr optimizer lr_schedule weight_decay      loss
#> 1     10         8 0.001    adam    cosine          0 bce_with_logits
#> 2     10         8 0.001    adam    cosine          0 bce_with_logits
#>   label_smoothing focal_gamma  metric      score score_sd repeats
#> 1              0          2 accuracy 0.5000000 0.1154701      3
```

```
#> 2 0 2 accuracy 0.488889 0.195315 3
```

0.5 Permutation Importance

Permutation importance estimates how much a metric changes after shuffling each predictor.

```
importance <- perm_importance(fit, iris[, -5], iris$Species)
importance$data
#>      feature importance
#> 1 Sepal.Length 0.2066667
#> 2 Petal.Length 0.1933333
#> 3 Petal.Width 0.1800000
#> 4 Sepal.Width 0.1333333
plot(importance)
```

