Electricity prices forecasting

Project work ASLII: Time Series Forecasting

Dario Falchetta

04-06-2024

Contents

Descrizione del dataset

Il dataset, reperibile da Kaggle, contiene dati riguardanti la Spagna tra gli anni 2015 e 2018 ed è suddiviso in due file:

- energy_dataset.csv: contiene i dati relativi a consumo, produzione e prezzo dell'energia elettrica (in €/MWh)
- weather_features.csv: contiene i dati metereologici.

L'obiettivo di questo progetto sarà quello di costruire una rete neurale che approssimi al meglio, e cerchi di prevedere, l'andamento del prezzo dell'energia elettrica. Per questo motivo, verrà considerata eslusivamente la variabile "price_actual" del file energy_dataset.csv ed eventualmente alcune altre variabili del file weather_features.csv durante la fase di analisi esplorativa.

```
dat <- read_csv(file = "./Dati/energy_dataset.csv")
weather <- read_csv(file = "Dati/weather_features.csv")
```

Analisi del dataset

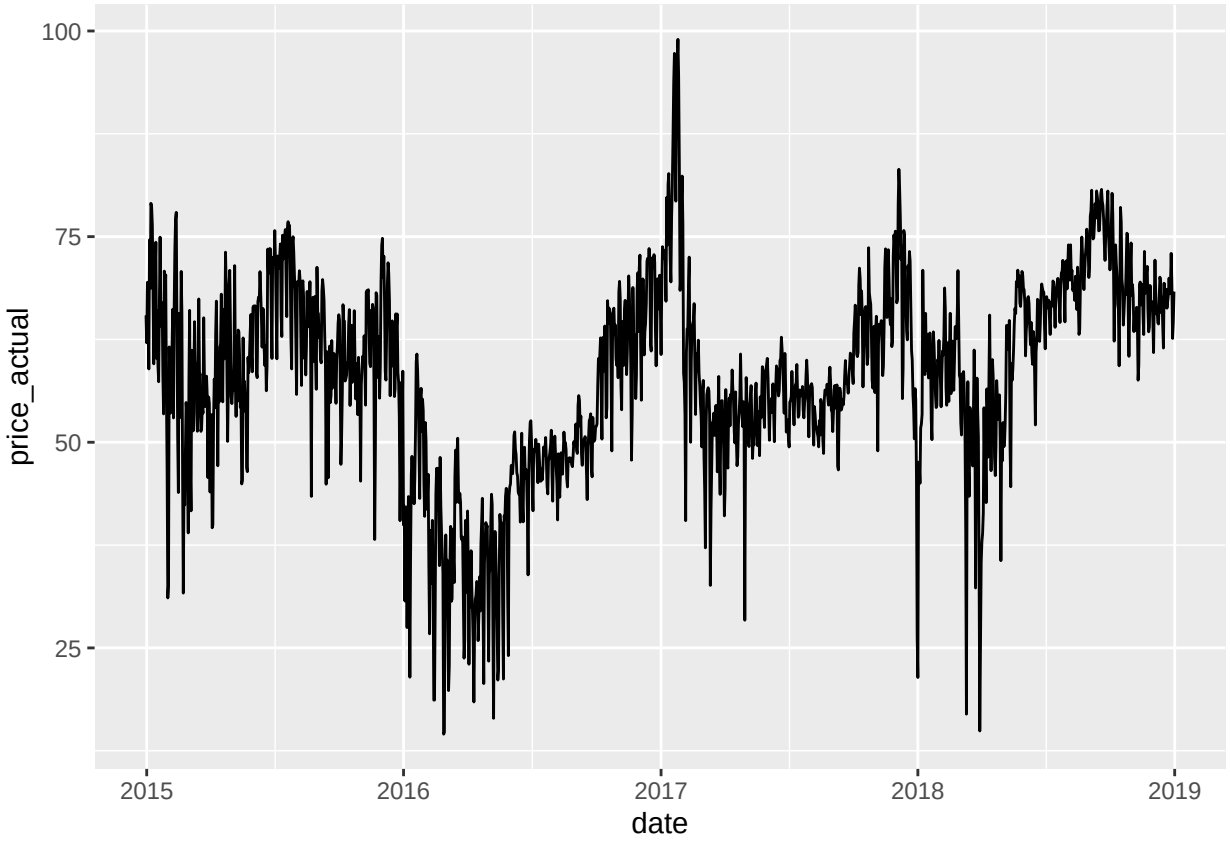
Data wrangling

Come prima cosa, vado a scartare tutte le variabili ad eccezione della variabile da modellare, ovvero *price_actual*. Inoltre, siccome la serie storica originale presenta rilevazioni orarie del prezzo, vado ad aggregare i valori della serie attraverso una media in modo da ottenere una serie storica giornaliera. Infine, converto la tibble in una tsibble per le analisi successive.

```
data <- dat %>%
  mutate(date = date(time)) %>%
  select(date, price_actual = `price actual`) %>%
  group_by(date) %>%
  summarise(price_actual = mean(price_actual)) %>%
  as_tsibble(index = date)
```

Analisi esplorativa

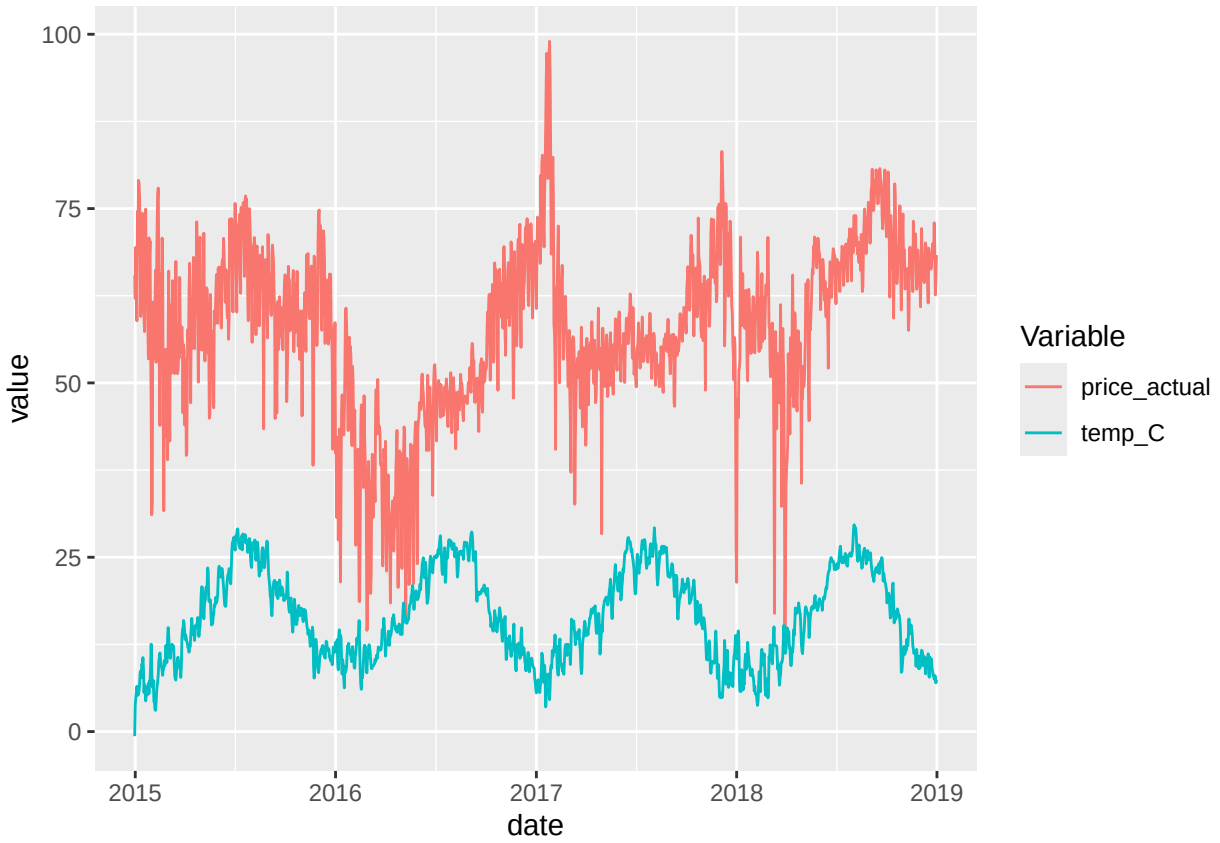
```
# Serie dei prezzi
data %>%
  ggplot(aes(date, price_actual)) +
  geom_line()
```



A giudicare dal plot della serie, i dati non sembrano avere una componente di trend, cioè non c'è un aumento o una diminuzione sistematica dei valori della serie, ma piuttosto questa oscilla sempre attorno ad un valore medio di 57.89€/MWh, con un massimo rilevato di 98.98€/MWh ed un minimo di 14.51€/MWh. Per avere un punto di riferimento per questi prezzi, una famiglia di 4 persone raggiunge consumi giornalieri tra gli 8 e i 9.5 kWh che, considerando il prezzo medio di questa serie storica, gli costerebbero 0.058€/MWh.

Una cosa evidente però è la presenza di una componente stagionale nei dati. Come è possibile vedere, il prezzo dell'elettricità sembra assumere valori più alti nel periodo finale dell'anno, in corrispondenza dell'inverno, per poi decrescere ed aumentare nuovamente col passare delle stagioni calde. Per verificare ciò, sovrappongo alla precedente serie storica, quella delle temperature (convertendole da gradi Kelvin a gradi Celsius per preferenza personale).

```
weather_temp <- weather %>%
  mutate(date = date(dt_iso),
         temp_C = temp - 273.15) %>%
  select(date, temp_C) %>%
  group_by(date) %>%
  summarise(temp_C = mean(temp_C)) %>%
  as_tsibble(index = date)


# Plot serie dei prezzi e temperature
merge(data, weather_temp, by = "date") %>%
  melt(id.var = "date") %>%
  dplyr::rename(Variable = variable) %>%
  ggplot(aes(x = date, y = value, colour = Variable)) +
  geom_line()
```



Quanto detto sembra effettivamente vero: le oscillazioni del prezzo dell'elettricità seguono quelle della temperatura, raggiungendo i punti più alti durante le stagioni fredde ed i più bassi durante le stagioni calde. In particolare, il prezzo massimo dell'elettricità 98.98€/MWh è stato rilevato in corrispondenza di una temperatura di 4.6°C mentre, in corrispondenza del prezzo minimo 14.51€/MWh, la temperatura rilevata è stata 8.4°C. Queste temperature non corrispondono necessariamente con le temperature minime e massime rilevate per la serie, rispettivamente pari a -0.66°C e 29.67°C, quindi è ragionevole pensare che la temperatura atmosferica non sia l'unico fattore che influenza il prezzo dell'elettricità.

Costruzione dei modelli

Test di non linearità

Prima di andare a stimare qualsiasi modello non lineare, occorre verificare se fare ciò è effettivamente opportuno: se i dati fossero lineari, non avrebbe senso usare modelli non lineari perché questo significherebbe rinunciare a tutti i vantaggi dei primi tra cui le garanzie teoriche presenti in letteratura, la loro semplicità computazionale e, soprattutto nel caso delle reti neurali, la loro interpretabilità, che verrebbe completamente persa.

Il test di White e quello di Terasvirta sono due esempi di test di linearità basati sulle reti neurali ed entrambi formalizzano il sistema di ipotesi sostanzialmente come

$$H_0 : il\ DGP \text{ è } lineare, \qquad H_1 : il\ DGP\ non \text{ è } lineare$$

Questi test sono effettivamente implementati aggiungendo una parte lineare al modello, uno skip layer, e andando a verificare che i parametri associati alla parte non lineare siano tutti nulli. Qualora questo non

fosse vero, ovvero qualora ci fosse anche solo un parametro diverso da zero, l'ipotesi nulla di linearità sarebbe rifiutata.

Sia il test di White che quello di Terasvirta sono dei test chi-quadro la cui statistica test è basata sull'indice R^2 calcolato su una regressione ausiliaria che usa come predittori le componenti principali degli output dei neuroni dell'hidden layer:

- il test di White è basato su un'inizializzazione casuale dei pesi quindi presenta un certo livello di imprevedibilità, il che potrebbe portarlo a generare conclusioni contraddittorie se eseguito poche volte. Per questo motivo, vado ad eseguirlo un gran numero di volte e a considerare le conclusioni complessivamente;
- il test di Terasvirta è il più potente tra i due ed è deterministico quindi se eseguito diverse volte, produce sempre lo stesso risultato.

Per risolvere il problema del test di White vado a scrivere una funzione che prende in input il dataset da testare, il numero di volte in cui eseguire il test, il livello di confidenza desiderato ed un seed per controllare la procedura randomica (se il seed non viene specificato, viene generato internamente alla funzione e poi restituito per avere un punto di riferimento). La funzione poi, tramite un ciclo for, va ad eseguire il test il numero specificato di volte e salva i risultati (rifiuto/non rifiuto) in un vettore che verrà poi restituito alla fine, insieme a tutti gli altri parametri di controllo utilizzati.

```
# Test di White e Terasvirta
repeated_white_test <- function(data, n=100, alpha=0.05, seed){
  # Se il seed non è stato specificato, viene generato casualmente
  if(missing(seed)){
    seed <- sample(1:9999, size = 1, replace = TRUE)
  }
  set.seed(seed)

  white <- rep(0, 100)

  # Test
  for(i in 1:n){
    w <- ifelse(white.test(as.ts(data$price_actual))$p.value < alpha,
                "Rifiuto H0", "Non rifiuto H0")
    white[i] <- w

  }

  to_return <- list(res = white,
                    n = n,
                    alpha = alpha,
                    seed = seed)


  return(to_return)
}
```

```
daily_res <- repeated_white_test(data = data, seed = 7820)

# Test di White su dati giornalieri
table(daily_res$res)
```

```
##
## Non rifiuto H0     Rifiuto H0
##             33             67
```

```
# Test di Terasvirta su dati giornalieri
ifelse(terasvirta.test(as.ts(data$price_actual))$p.value < 0.05,
            "Rifiuto H0", "Non rifiuto H0")
```

```
## [1] "Rifiuto H0"
```

Come anticipato, il test di White ha prodotto risultati più incerti ma nella maggior parte dei casi ha comunque portato al rifiuto dell'ipotesi di linearità. Invece, il test di Terasvirta ha portato al rifiuto dell'ipotesi di linearità. Di conseguenza, sulla base dei risultati ottenuti, è possibile concludere che il DGP per la serie dei prezzi dell'elettricità è effettivamente non lineare e quindi risulta opportuno l'utilizzo delle reti neurali per la modellazione.

Modello su dati giornalieri senza ritardo stagionale

Come primo modello, andrò a stimare una rete neurale sui dati utilizzati finora, ovvero quelli che sono stati portati ad avere frequenza giornaliera. Siccome la componente giornaliera è molto grande e può variare a seconda dell'anno bisestile ($m = 365$ o $m = 366$), non utilizzerò i ritardi stagionali come predittori di questo primo modello.

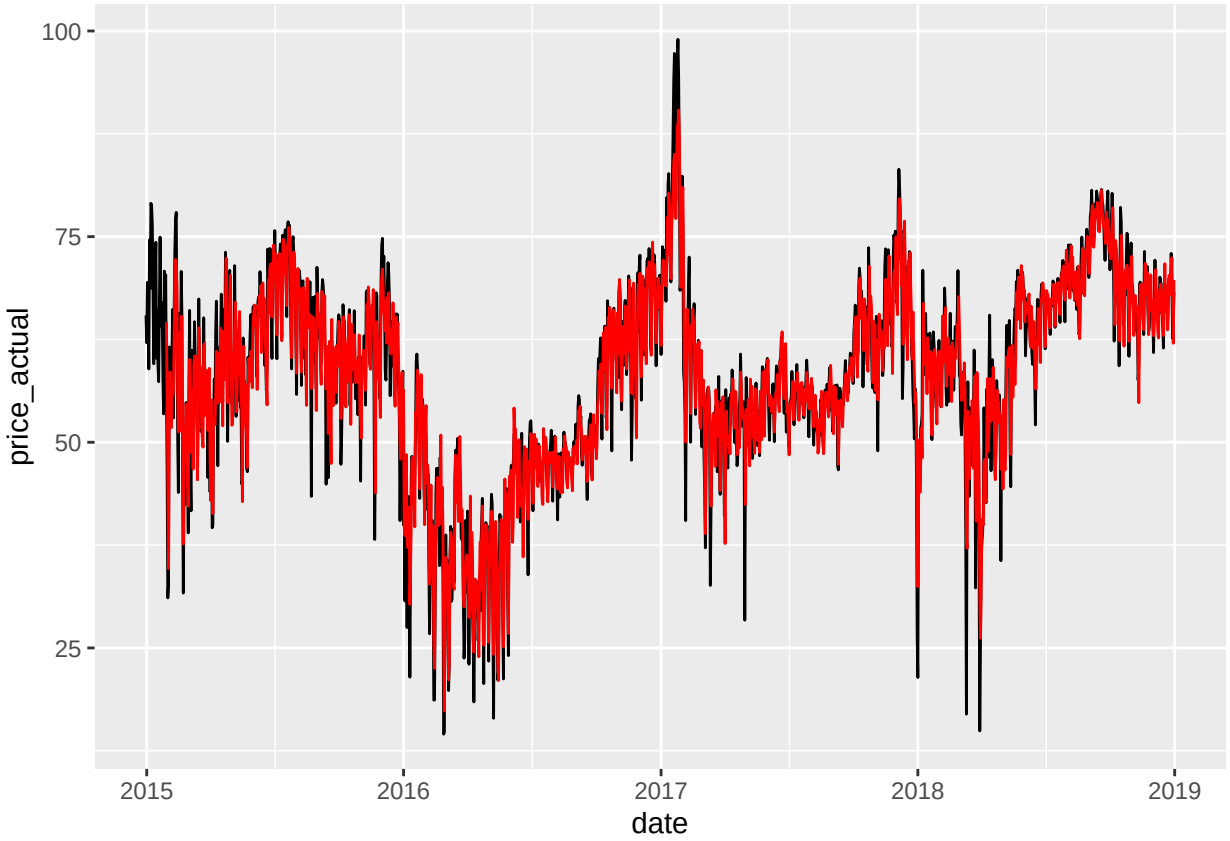
Inizio stimando una rete con 5 neuroni intermedi e senza penalizzazione. Non specificando la quantità di lag da considerare, il software andrà a costruire diversi modelli e sceglierà il valore di lag associato a quello con l'AIC più basso.

```
fit <- data %>%
  model(nn5 = NNETAR(price_actual ~ AR(),       ## Modello nn5
                     n_nodes = 5,               # - n_nodes = 5
                     n_networks = 10,           # - penalty = 0
                     scale_inputs = TRUE))

fit %>% report()
```

```
## Series: price_actual
## Model: NNAR(29,1,5)[7]
##
## Average of 10 networks, each of which is
## a 29-5-1 network with 156 weights
## options were - linear output units
##
## sigma^2 estimated as 15.28
```

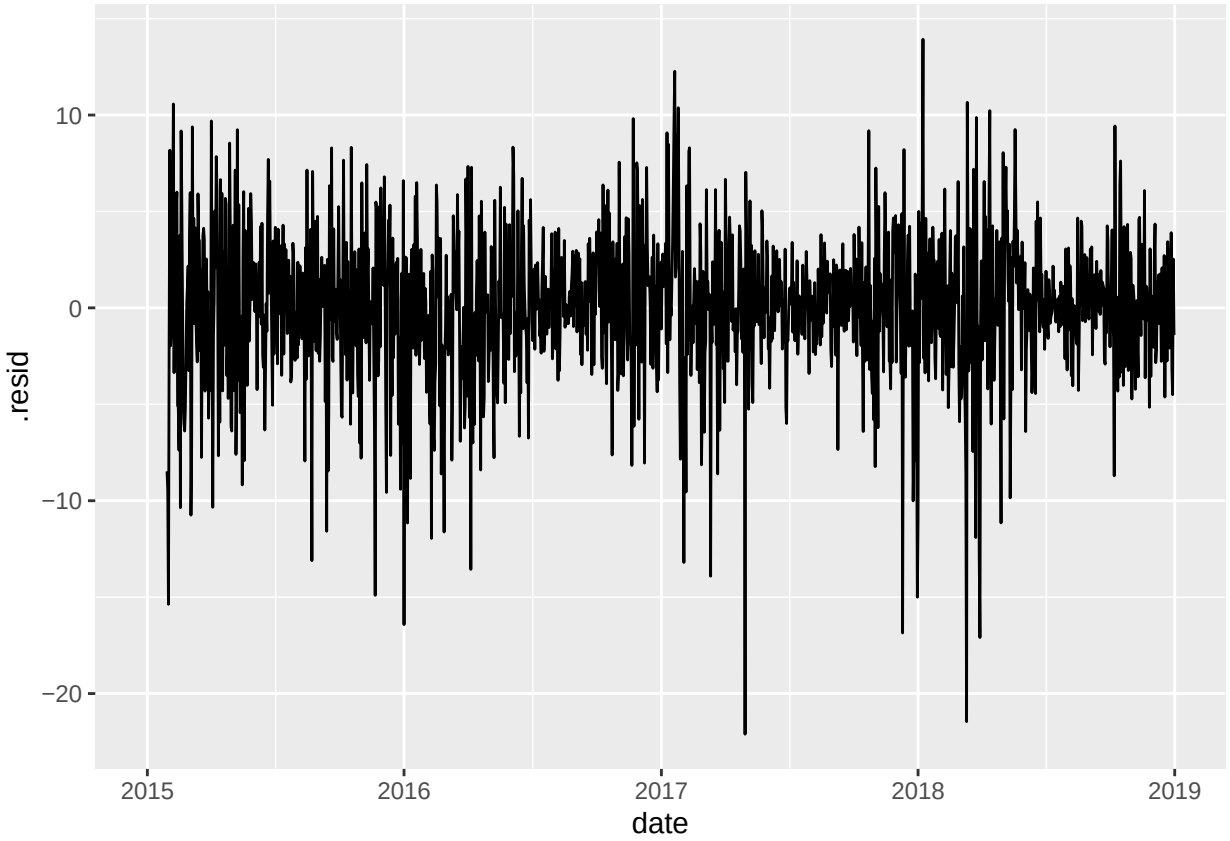
Il valore di ritardo che è stato scelto è 29. Questo significa che per la previsione di ogni valore, la rete userà i valori osservati della serie nei 29 giorni precedenti.

```
# Plot dei fitted values
fit %>%
 augment() %>%
 ggplot(aes(x = date, y = price_actual)) +
  geom_line() +
  geom_line(aes(x = date, y = .fitted), col="red")
```



La rete sembra adattarsi piuttosto bene alla serie osservata, riuscendo a catturare il suo andamento ed anche gli occasionali picchi, sia positivi che negativi. Tuttavia, la rete potrebbe aver colto troppo bene le oscillazioni della serie ed essere così finita in overfitting.

```
# Plot dei residui
fit %>%
  augment() %>%
  ggplot(aes(x = date, y = .resid)) +
  geom_line()
```



Quanto detto è confermato dai residui, che non sembrano presentare alcuna struttura e sembrano oscillare attorno ad un valore medio di 0 anche se, in realtà, queste oscillazioni sono piuttosto ampie, soprattutto in corrispondenza dei picchi della serie.

Confronto tra modelli

Stimato un semplice modello iniziale e analizzate rapidamente le sue performance, vado a ripetere la stima, andando a considerare diverse configurazioni per la rete. In particolare, vado a considerare le performance di modelli costruiti a partire da un numero crescente di neuroni ed a verificare se una leggera penalizzazione permette di ottenere risultati migliori.

```
## Confronto modelli
fit_cmp <- data %>%
  model(nn5 = NNETAR(price_actual ~ AR(),                    ## Modello nn5
                     n_nodes = 5,                            # - n_nodes = 5
                     n_networks = 10,                        # - penalty = 0
                     scale_inputs = TRUE),
        nn5_wp = NNETAR(price_actual ~ AR(),                 ## Modello nn5_wp
                        n_nodes = 5,                         # - n_nodes = 5
                        penalty = 0.0005,                    # - penalty = 0.0005
                        n_networks = 10,
                        scale_inputs = TRUE),

        nn10 = NNETAR(price_actual ~ AR(),                   ## Modello nn10
                      n_nodes = 10,                          # - n_nodes = 10
```

```
                      n_networks = 10,                    # - penalty = 0
                      scale_inputs = TRUE),
        nn10_wp = NNETAR(price_actual ~ AR(),             ## Modello nn10_wp
                         n_nodes = 10,                    # - n_nodes = 10
                         penalty = 0.0005,                # - penalty = 0.0005
                         n_networks = 10,
                         scale_inputs = TRUE),

        nn20 = NNETAR(price_actual ~ AR(),                ## Modello nn20
                      n_nodes = 20,                       # - n_nodes = 20
                      n_networks = 10,                    # - penalty = 0
                      scale_inputs = TRUE),
        nn20_wp = NNETAR(price_actual ~ AR(),             ## Modello nn20_wp
                         n_nodes = 20,                    # - n_nodes = 20
                         penalty = 0.0005,                # - penalty = 0.0005
                         n_networks = 10,
                         scale_inputs = TRUE))

fit_cmp %>% accuracy() %>%
  arrange(RMSE)
```

```
## # A tibble: 6 x 10
##   .model  .type          ME  RMSE   MAE    MPE  MAPE  MASE RMSSE    ACF1
##   <chr>   <chr>       <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>   <dbl>
## 1 nn20_wp Training -0.00862  1.92  1.46 -0.330  2.75 0.265 0.246 0.00735
## 2 nn20    Training -0.00282  1.98  1.50 -0.337  2.81 0.272 0.254 0.0130
## 3 nn10    Training  0.0273   2.87  2.18 -0.512  4.21 0.396 0.368 0.0249
## 4 nn10_wp Training  0.0194   2.89  2.19 -0.544  4.24 0.399 0.370 0.0222
## 5 nn5_wp  Training  0.0263   3.79  2.80 -0.826  5.54 0.510 0.486 0.00911
## 6 nn5     Training  0.00575  3.82  2.81 -0.897  5.60 0.512 0.490 0.0311
```

Come è possibile vedere da risultati ottenuti per misure come il RMSE ed il MAE, è evidente che il numero di neuroni permetta di ottenere un'approssimazione sempre migliore del DGP. Meno scontato è quello che succede con il parametro di penalizzazione: per il modello semplice ($n\_nodes = 5$) e quello più complesso ($n\_nodes = 20$), l'aggiunta di un termine di penalizzazione ($penalty = 0.0005$) sembra portare i modelli a produrre risultati leggermente migliori, più significativi sul modello semplice. Invece, per il modello con r=10, l'aggiunta del termine di penalizzazione lo porta a produrre risultati leggermente peggiori. Il magnitudo di queste variazioni tuttavia è molto piccolo quindi potrebbe essere considerato trascurabile.

Confronto previsioni ex-ante vs ex-post

Per determinare quale di questi modelli è quello che si comporta meglio, vado a considerare le loro previsioni ex-post, ovvero le previsioni che producono per periodi già effettivamente osservati. Dopodiché, i migliori tra questi modelli verranno confrontati sulle loro previsioni ex-ante, in modo da analizzare il loro comportamento.

Previsioni ex-post

Vado a stimare i modelli menzionati sui dati osservati fino a Novembre 2018 e prevedo sui dati di Dicembre 2018, confrontando poi le previsioni con i valori osservati della serie.

```r
# Addestramento su dati fino al 2018-12-01
fit_cmp_expost <- data %>%
  filter(date < "2018-12-01") %>%
  model(nn5 = NNETAR(price_actual ~ AR(),
                     n_nodes = 5,
                     n_networks = 10,
                     scale_inputs = TRUE),
        nn5_wp = NNETAR(price_actual ~ AR(),
                        n_nodes = 5, penalty = 0.0005,
                        n_networks = 10,
                        scale_inputs = TRUE),

        nn10 = NNETAR(price_actual ~ AR(),
                      n_nodes = 10,
                      n_networks = 10,
                      scale_inputs = TRUE),
        nn10_wp = NNETAR(price_actual ~ AR(),
                         n_nodes = 10, penalty = 0.0005,
                         n_networks = 10,
                         scale_inputs = TRUE),

        nn20 = NNETAR(price_actual ~ AR(),
                      n_nodes = 20,
                      n_networks = 10,
                      scale_inputs = TRUE),
        nn20_wp = NNETAR(price_actual ~ AR(),
                         n_nodes = 20, penalty = 0.0005,
                         n_networks = 10,
                         scale_inputs = TRUE))
```

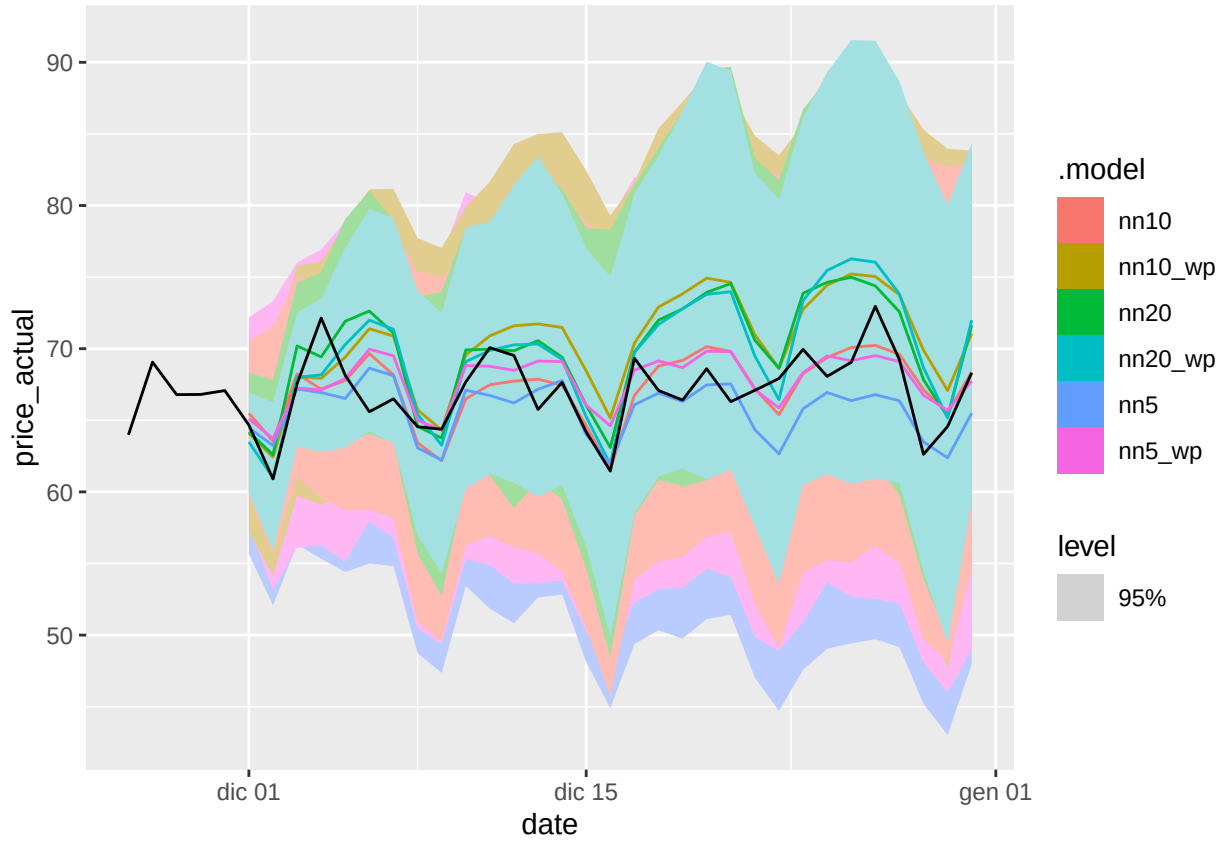
```r
# Previsioni dal 2018-12-01 al 2018-12-31
preds_expost <- fit_cmp_expost %>%
 forecast(h=31,
          simulate = F, bootstrap = T, times = 999,
          point_forecast = list(.mean = mean))

preds_expost %>% accuracy(data) %>%
  arrange(RMSE)
```

```
## # A tibble: 6 x 10
##   .model  .type     ME  RMSE   MAE    MPE  MAPE  MASE RMSSE  ACF1
##   <chr>   <chr>  <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 nn10    Test  -0.210  2.18  1.75 -0.391  2.61 0.315 0.277 0.195
## 2 nn5_wp  Test  -0.689  2.24  1.76 -1.13   2.64 0.316 0.285 0.164
## 3 nn5     Test   1.35   2.62  2.07  1.92   3.04 0.372 0.333 0.251
## 4 nn20    Test  -2.85   3.85  3.10 -4.27   4.64 0.558 0.489 0.266
## 5 nn20_wp Test  -2.56   3.90  3.07 -3.82   4.57 0.552 0.495 0.403
## 6 nn10_wp Test  -3.24   4.25  3.55 -4.87   5.32 0.638 0.540 0.424
```

Come volevasi dimostrare, le performance apparentemente migliori dei modelli complessi sulla porzione di training del dataset, si sono tradotte in una scarsa capacità di generalizzazione a causa dell'overfitting ai dati osservati, portando i modelli più complessi ad ottenere errori grandi sulle previsioni ex-post, facendo risaltare invece la capacità di generalizzare dei modelli semplici.

Per questo motivo, per le previsioni ex-ante, considererò solo i 3 modelli migliori in termini di RMSE e MAE, ovvero i modelli denominati *nn10*, *nn5_wp* ed *nn5*.

```
# Plot delle previsioni dal 2018-12-01 al 2018-12-31
preds_expost %>%
  autoplot(data %>%
             filter(date > "2018-11-25"),
           level = 95)
```



Le previsioni dei vari modelli sembrano seguire praticamente lo stesso andamento, anche se sembra che i modelli più complessi tendono a sovrastimare il valore della quantità di elettricità domandata. Anche gli intervalli di previsione sembrano essere pressocché identici, se non per lo "scarto grafico" dovuto alle diverse altezze delle curve.

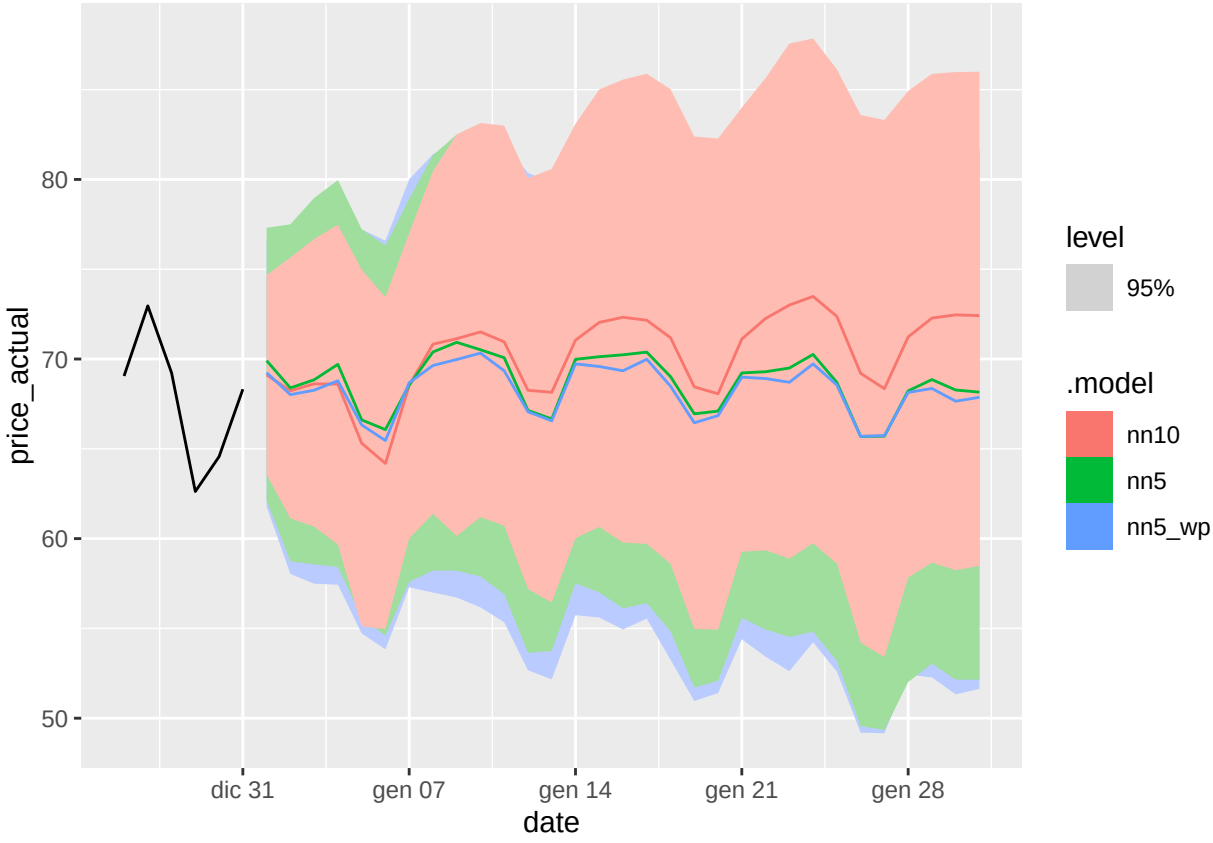
La serie delle previsioni per il mese di Dicembre 2018 sembra presentare un leggero trend crescente, fenomeno che si era già osservato inizialmente durante la fase di analisi esplorativa: col trascorrere della stagione fredda, la domanda dell'elettricità, e di conseguenza il suo prezzo, risulta essere sempre più alta.

Previsioni ex-ante

A questo punto, presi i modelli *nn10*, *nn5_wp* ed *nn5*, quelli che si erano comportati meglio sulle previsioni ex-post, vado a confrontarne le previsioni ex-ante per l'intero mese di Gennaio 2019.

```
gen19_cmp <- fit_cmp %>%
  select(nn5_wp, nn5, nn10) %>%
  forecast(h = 31)
```

```
gen19_cmp %>%
  autoplot(data %>%
             filter(date > "2018-12-25"),
           level = 95)
```



Il grafico delle previsioni di Gennaio 2019 mostra la presenza di un leggero trend crescente come appena osservato dalle previsioni per Dicembre. Questo trend quindi è probabilmente spiegato dall'arrivo della parte più fredda della stagione invernale e la diminuzione delle temperature che porta gli individui ad usare una maggiore quantità di energia elettrica per il riscaldamento. Altra componente evidente di queste previsioni è l'aumento dell'incertezza, cioè della varianza del previsore, ed il conseguente aumento dell'ampiezza degli intervalli di previsione. Questo comportamento è ragionevole: più è distante l'orizzonte previsivo, più è difficile prevedere perché non sappiamo cosa potrebbe accadere dall'istante temporale attuale a quello in cui si prevede, soprattutto nel caso in cui quest'ultimo è molto lontano.

```
# Prendo gli intervalli di previsione al 95%
forecast_intervals <- gen19_cmp %>%
  hilo(level = 95)

# Separo per modello
intervals_nn5 <- forecast_intervals %>%
  filter(.model == "nn5")

intervals_nn5_wp <- forecast_intervals %>%
  filter(.model == "nn5_wp")
```

```r
intervals_nn10 <- forecast_intervals %>%
  filter(.model == "nn10")


# Calcolo l'ampiezza media degli intervalli di previsione dei vari modelli
avg_interval_widths <- tibble(nn5_avg_interval_width =
                                mean(intervals_nn5$`95%`$upper -
                                       intervals_nn5$`95%`$lower),
                              nn5_wp_avg_interval_width =
                                mean(intervals_nn5_wp$`95%`$upper -
                                       intervals_nn5_wp$`95%`$lower),
                              nn10_avg_interval_width =
                                mean(intervals_nn10$`95%`$upper -
                                       intervals_nn10$`95%`$lower)
                              )

avg_interval_widths
```

```
## # A tibble: 1 x 3
##   nn5_avg_interval_width nn5_wp_avg_interval_width nn10_avg_interval_width
##                    <dbl>                     <dbl>                    <dbl>
## 1                   25.6                      26.7                     23.6
```

Tra i tre modelli considerati, quello che produce le previsioni mediamente più sicure è il modello con 10 neuroni nell'hidden layer e nessuna penalizzazione.

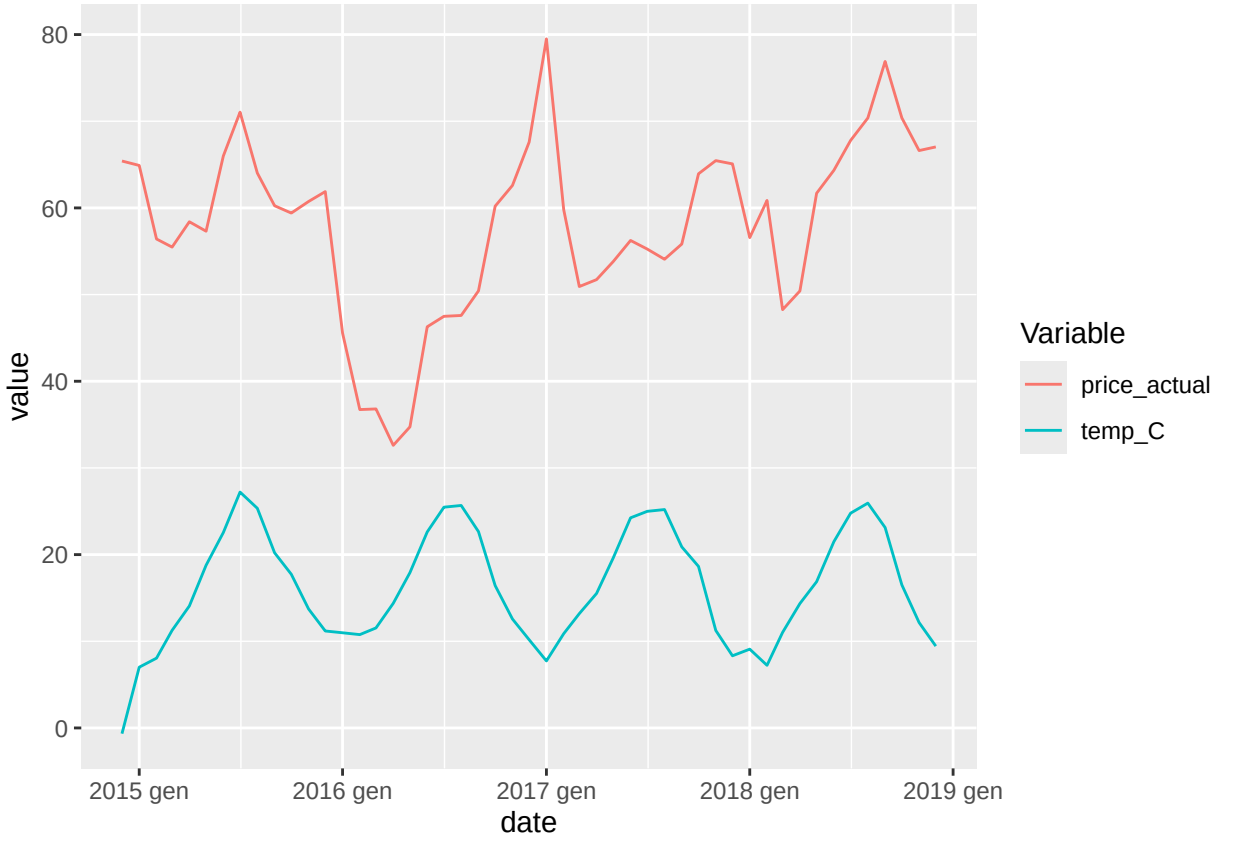
Aggregazione in dati mensili

Per semplificare la struttura stagionale dei dati e per provare ad introdurla nelle reti stimate, vado ad aggregare le osservazioni fino ad averne una per ogni mese del periodo di interesse.

```r
# Aggregazione in dati mensili
data_monthly <- dat %>%
  mutate(date = yearmonth(time)) %>%
  select(date, price_actual = `price actual`) %>%
  group_by(date) %>%
  summarise(price_actual = mean(price_actual)) %>%
  as_tsibble(index = date)

weather_temp_monthly <- weather %>%
  mutate(date = yearmonth(dt_iso),
         temp_C = temp - 273.15) %>%
  select(date, temp_C) %>%
  group_by(date) %>%
  summarise(temp_C = mean(temp_C)) %>%
  as_tsibble(index = date)
```

```r
# Plot serie dei prezzi e temperature
merge(data_monthly, weather_temp_monthly, by = "date") %>%
  melt(id.var = "date") %>%
  dplyr::rename(Variable = variable) %>%
```

```
  ggplot(aes(x = date, y = value, colour = Variable)) +
  geom_line()
```



Ovviamente la serie è sempre la stessa. Tuttavia, le forti oscillazioni dovute alla frequenza delle rilevazioni vengono smussate dalla media dei loro valori condizionatamente al giorno in cui queste sono state registrate.

Test di linearità

A questo punto, vado a studiare la non linearità dei nuovi dati aggregati, in modo da studiare la loro predisposizione ad essere modellati da modelli non lineari.

```
monthly_res <- repeated_white_test(data = data_monthly, seed = 7820)

# Test di White su dati mensili
table(monthly_res$res)
```

```
##
## Non rifiuto H0     Rifiuto H0
##             97              3
```

```
# Test di Terasvirta su dati mensili
ifelse(terasvirta.test(as.ts(data_monthly$price_actual))$p.value < 0.05,
            "Rifiuto H0", "Non rifiuto H0")
```

```
## [1] "Non rifiuto H0"
```

In seguito all'aggregazione in mensili, i dati portano al non rifiuto dell'ipotesi di linearità, suggerendo che un modello non lineare come una rete neurale potrebbe essere inutilmente complesso per il set di dati in questione. Come ulteriore conferma di questa affermazione, vado a confrontare le performance di un modello ARIMA con quelle di una rete neurale, in modo da determinare se conviene o meno ricorrere ad un modello più complesso.

Confronto tra modelli: ARIMA vs NN

Per quanto riguarda la configurazione del modello da utilizzare per il confronto con un modello ARIMA, siccome il test di linearità ha dato modo di pensare che i dati siano lineari, vado a stimare solo le reti più semplici tra le precedenti. In particolare considero tutte le configurazioni considerate precedentemente, ad eccezione di quelle con 20 neuroni nell'hidden layer. Inoltre, siccome ho convertito i dati in mensili, la stagionalità risulta più semplice da modellare quindi come confronto vado anche ad introdurre il lag stagionale per ogni modello e vedere come cambiano i risultati.

```
fit_arima_nn <- data_monthly %>%
  model(arima = ARIMA(price_actual),
        nn5 = NNETAR(price_actual ~ AR(),
                     n_nodes = 5,
                     n_networks = 10,
                     scale_inputs = TRUE),
        nn5_wp = NNETAR(price_actual ~ AR(),
                        n_nodes = 5,
                        penalty = 0.0005,
                        n_networks = 10,
                        scale_inputs = TRUE),

        nn5_seasonal = NNETAR(price_actual ~ AR(p=12),
                              n_nodes = 5,
                              n_networks = 10,
                              scale_inputs = TRUE),
        nn5_wp_seasonal = NNETAR(price_actual ~ AR(p=12),
                                 n_nodes = 5,
                                 penalty = 0.0005,
                                 n_networks = 10,
                                 scale_inputs = TRUE),

        nn10 = NNETAR(price_actual ~ AR(),
                      n_nodes = 10,
                      n_networks = 10,
                      scale_inputs = TRUE),
        nn10_wp = NNETAR(price_actual ~ AR(),
                         n_nodes = 10,
                         penalty = 0.0005,
                         n_networks = 10,
                         scale_inputs = TRUE),

        nn10_seasonal = NNETAR(price_actual ~ AR(p=12),
                               n_nodes = 10,
                               n_networks = 10,
                               scale_inputs = TRUE),
```

```
        nn10_wp_seasonal = NNETAR(price_actual ~ AR(p=12),
                                  n_nodes = 10,
                                  penalty = 0.0005,
                                  n_networks = 10,
                                  scale_inputs = TRUE)
        )

fit_arima_nn %>% accuracy() %>%
  arrange(RMSE)
```

```
## # A tibble: 9 x 10
##   .model          .type       ME    RMSE     MAE      MPE    MAPE    MASE   RMSSE
##   <chr>           <chr>    <dbl>   <dbl>   <dbl>    <dbl>   <dbl>   <dbl>   <dbl>
## 1 nn10            Trai~ -3.19e-5 0.00371 0.00298 -1.69e-4 0.00531 2.51e-4 2.51e-4
## 2 nn10_wp_seaso~  Trai~  9.79e-6 0.00373 0.00306  1.48e-4 0.00547 2.58e-4 2.52e-4
## 3 nn10_seasonal   Trai~  6.93e-4 0.00415 0.00311  1.31e-3 0.00613 2.62e-4 2.81e-4
## 4 nn10_wp         Trai~ -2.25e-3 0.0287  0.0166  -4.05e-3 0.0303  1.40e-3 1.94e-3
## 5 nn5_wp_season~  Trai~ -1.78e-4 0.0859  0.0328  -2.76e-3 0.0632  2.76e-3 5.81e-3
## 6 nn5_seasonal    Trai~ -4.23e-4 0.164   0.0812  -7.76e-3 0.143   6.85e-3 1.11e-2
## 7 nn5_wp          Trai~  1.10e-2 0.438   0.294   -1.01e-2 0.539   2.48e-2 2.96e-2
## 8 nn5             Trai~ -1.66e-3 0.585   0.389   -4.06e-2 0.689   3.28e-2 3.95e-2
## 9 arima           Trai~ -1.02e-1 6.01    4.40    -1.47e+0 8.08    3.71e-1 4.07e-1
## # i 1 more variable: ACF1 <dbl>
```

Sui dati di training, il modello ARIMA sembra ottenere un errore di un ordine di grandezza superiore rispetto alle reti più semplici, e diversi ordini di grandezza rispetto alle migliori, mostrando come questo in realtà non si comporti bene quanto le reti considerate. Invece, per quanto riguarda le reti, le migliori sono quelle con 10 neuroni nell'hidden layer, che riescono a raggiungere tutte livelli di performance molto simili in termini di RMSE e MAE. Le reti con n_nodes = 5 invece producono performance diverse ma, tra queste, quelle che considerano anche la stagionalità dei dati sono le migliori.

Come ulteriore controllo, considero anche le previsioni ex-post, stimando nuovamente tutti i modelli per capire quali sono quelli che preservano la miglior capacità di generalizzazione.

Confronto ARIMA vs NN su previsioni ex-post

```
# Addestramento su dati fino al 2018-06
fit_arima_nn_expost <- data_monthly %>%
  filter(date < yearmonth("2018-06")) %>%
  model(arima = ARIMA(price_actual),
        nn5 = NNETAR(price_actual ~ AR(),
                     n_nodes = 5,
                     n_networks = 10,
                     scale_inputs = TRUE),
        nn5_wp = NNETAR(price_actual ~ AR(),
                        n_nodes = 5,
                        penalty = 0.0005,
                        n_networks = 10,
                        scale_inputs = TRUE),

        nn5_seasonal = NNETAR(price_actual ~ AR(p=12),
```

```
                            n_nodes = 5,
                            n_networks = 10,
                            scale_inputs = TRUE),
        nn5_wp_seasonal = NNETAR(price_actual ~ AR(p=12),
                               n_nodes = 5,
                               penalty = 0.0005,
                               n_networks = 10,
                               scale_inputs = TRUE),

        nn10 = NNETAR(price_actual ~ AR(),
                      n_nodes = 10,
                      n_networks = 10,
                      scale_inputs = TRUE),
        nn10_wp = NNETAR(price_actual ~ AR(),
                         n_nodes = 10,
                         penalty = 0.0005,
                         n_networks = 10,
                         scale_inputs = TRUE),

        nn10_seasonal = NNETAR(price_actual ~ AR(p=12),
                               n_nodes = 10,
                               n_networks = 10,
                               scale_inputs = TRUE),
        nn10_wp_seasonal = NNETAR(price_actual ~ AR(p=12),
                                  n_nodes = 10,
                                  penalty = 0.0005,
                                  n_networks = 10,
                                  scale_inputs = TRUE)
        )
```

```
# Previsioni dal 2018-06 al 2018-12
preds_arima_nn_expost <- fit_arima_nn_expost %>%
 forecast(h=31,
          simulate = F, bootstrap = T, times = 999,
          point_forecast = list(.mean = mean))

preds_arima_nn_expost %>% accuracy(data) %>%
  arrange(RMSE)
```

```
## # A tibble: 9 x 10
##   .model           .type    ME  RMSE   MAE   MPE  MAPE  MASE RMSSE   ACF1
##   <chr>            <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
## 1 nn5              Test   5.25  7.74  6.07  7.66  8.89 0.751 0.695  0.160
## 2 arima            Test   9.04  9.89  9.04 13.2  13.2  1.12  0.888  0.390
## 3 nn5_wp           Test  10.3  10.7  10.3  15.2  15.2  1.27  0.965 -0.218
## 4 nn10_wp          Test  10.5  11.2  10.5  15.5  15.5  1.30  1.01  -0.289
## 5 nn5_wp_seasonal  Test  11.3  12.1  11.3  16.9  16.9  1.40  1.09   0.602
## 6 nn10_seasonal    Test  12.7  13.2  12.7  18.9  18.9  1.58  1.19   0.513
## 7 nn5_seasonal     Test  13.8  14.0  13.8  20.5  20.5  1.71  1.25   0.344
## 8 nn10_wp_seasonal Test  14.5  15.2  14.5  21.5  21.5  1.80  1.37   0.537
## 9 nn10             Test  17.7  18.1  17.7  26.1  26.1  2.19  1.63  -0.255
```

I risultati ottenuti sono coerenti con i risultati ottenuti finora. In termini di previsioni ex-post, i modelli più complessi si erano presentati come quelli dalle migliori performance, data la capacità delle reti di ottenere

errori arbitrariamente piccoli semplicemente aumentando il numero di neuroni. Tuttavia, in questo esempio è osservabile anche il lato negativo di questa affermazione, ovvero la propensione delle reti complesse ad andare in overfitting. Infatti, i modelli migliori sono quelli più semplice.

In aggiunta a questo, i risultati vanno anche a confermare le decisioni consigliate dai test di White di Terasvirta sul nuovo dataset aggregato: i test suggerivano la linearità dei dati e infatti il modello ARIMA stimato si comporta meglio di tutte le reti neurali considerate, comparabile solo con la più semplice, il che può essere vista come ulteriore evidenza del fatto che un modello semplice potrebbe essere più opportuno per questa serie storica.

```
#save.image(file = "workspace_image.RData")
```

Conclusioni

L'analisi condotta ha permesso di osservare una serie di aspetti che possono presentarsi nel lavorare con reti neurali applicate alle serie storiche.

Per prima cosa, si ha avuto un assaggio di cosa significa lavorare con serie storiche dall'alta frequenza, ovvero con rilevazioni poco distanti l'una dall'altra nel tempo, e della potenza computazionale necessaria.

Dopo aver aggregato i dati in giornalieri e dopo aver condotto i test di linearità per verificare se fosse opportuno modellare la relazione tra i dati con un modello non lineare come una rete neurale, è stata osservata la differenza di potenza tra i test di White e di Terasvirta e di come il primo producesse risultati instabili perché influenzato dall'inizializzazione casuale dei pesi della rete: nonostante ciò, dopo averlo eseguito un gran numero di volte, entrambi i test hanno presentato i dati come non lineari, e quindi modellabili da una rete neurale.

Durante l'addestramento su questi dati giornalieri, i modelli più complessi tendevano ad andare in overfitting, come è stato possibile osservare a partire dalle previsioni ex-post.

In un tentativo di semplificare ulteriormente la struttura dei dati in modo da poter inserire le osservazioni stagionali nel modello con maggiore facilità, sono andato ad aggregarli ulteriormente fino ad avere un'osservazione per ogni mese del periodo di rilevazione. Questo ha, infatti, semplificato la struttura dei dati, al punto da renderli lineari secondo i test di White e Terasvirta. Come ulteriore verifica, sono andato a confrontare le performance ottenute da reti neurali, configurate come allo step precedente, con quelle ottenute da un modello ARIMA. Sui dati di training, le reti più complesse ottenevano performance di ordini di magnitudo migliori di quelle del modello ARIMA. Tuttavia, sui dati di testing, il modello ARIMA era quello che riusciva a generalizzare meglio proprio perché i dati erano abbastanza semplice da poter essere modellati da una relazione lineare.