2. Problema 2: Machine Learning

En esta sección de la tarea, mostraremos primero los algoritmos utilizados en cada inciso y las pruebas de *cross-validation* ("CV") realizadas para seleccionar a los mejores. Una vez hecho, enlistaremos los mejores modelos de cada inciso, las variables utilizadas, y el resto de información solicitada.

2.1. Predicción de la inflación subyacente para el mes de octubre 2024 (pública en noviembre). Usar como máximo desde el periodo 2015.

En primer lugar utilizamos un modelo de *Support Vector Machines* ("SVM") con 22 variables. Utilizamos la función `train()` de la librería `caret` para entrenar el modelo y elegir los valores óptimos del regularizador mediante un proceso de CV. El siguiente código realiza lo anterior:

```
set.seed(1)
modelo_svm <- train(formula,
              data = as.ts(set_entrenamiento),
              method = "svmLinear",
              preProcess = c("center","scale"),
              tuneGrid = expand.grid(C = seq(0.1, 2, length = 20)))
modelo_svm
```

```
## Support Vector Machines with Linear Kernel
##
## 94 samples
## 22 predictors
##
## Pre-processing: centered (22), scaled (22)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 94, 94, 94, 94, 94, 94, ...
## Resampling results across tuning parameters:
##
##   C    RMSE       Rsquared   MAE
##   0.1  0.1818623  0.9870896  0.1446059
##   0.2  0.1752370  0.9881192  0.1382555
##   0.3  0.1703474  0.9887899  0.1338662
##   0.4  0.1679538  0.9889999  0.1312154
##   0.5  0.1669441  0.9890946  0.1300137
##   0.6  0.1660651  0.9892343  0.1292957
##   0.7  0.1655394  0.9893302  0.1288176
##   0.8  0.1653266  0.9893664  0.1285483
##   0.9  0.1654840  0.9893730  0.1284290
##   1.0  0.1652578  0.9894284  0.1280198
##   1.1  0.1651666  0.9894695  0.1277261
##   1.2  0.1652487  0.9894941  0.1278167
##   1.3  0.1657760  0.9894590  0.1281979
##   1.4  0.1664714  0.9894029  0.1289254
##   1.5  0.1669595  0.9893553  0.1295371
##   1.6  0.1675509  0.9892882  0.1301303
##   1.7  0.1684767  0.9891884  0.1310544
##   1.8  0.1694955  0.9890754  0.1321239
##   1.9  0.1700682  0.9890223  0.1328191
##   2.0  0.1705987  0.9889677  0.1333600
##
```

```
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was C = 1.1.
```

```
c <- modelo_svm$bestTune$C
```

Los resultados muestran que el valor del regularizador C elegido mediante CV es 1.1.

En segundo lugar, se entrenó un modelo de *Random Forests* ("RF") siguiendo una metodología similar al caso anterior, con la excepción de que probamos dos librerías distintas para entrenar el modelo y seleccionar el mejor mediante CV, las cuales fueron `randomForest` y `caret`; sin embargo, las pruebas realizadas indicaron que la primera de éstas dos resultaba normalmente en un mejor modelo, por lo que en este documento solamente mostramos el proceso de entrenamiento y CV usando dicha librería:

```
set.seed(1)
cv_randomF <- rfcv(trainx = as.ts(set_entrenamiento)[,-2],
                   trainy = as.ts(set_entrenamiento)[,2],
                   step = 0.9,
                   cv.fold = 5)

# nos dice cuál es el número óptimo
mtry_CV <- cv_randomF$error.cv[which.min(cv_randomF$error.cv)]
mtry_CV <- as.numeric(names(mtry_CV))
mtry_CV
```

```
## [1] 8
```

Lo anterior indica que el número de variables a utilizar sería 8 según los resultados de CV. Con esta información, podemos entrenar el modelo de RF; por simplicidad, fijaremos en 500 el número de árboles del algoritmo, aunque podríamos buscar el número óptimo mediante CV. El siguiente código entrena el algoritmo:

```
set.seed(1)
modelo_rf = randomForest(formula = formula,
                         data = set_entrenamiento,
                         ntree = 500,
                         mtry = mtry_CV)
modelo_rf
```

```
##
## Call:
##  randomForest(formula = formula, data = set_entrenamiento, ntree = 500,      mtry = mtry_CV)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 8
##
##           Mean of squared residuals: 0.02672882
##                     % Var explained: 99.02
```

En tercer lugar, se eligió una red neuronal de una capa oculta; la metodología de entrenamiento y CV fue la proporcionada por `caret`, con la diferencia de que especificamos el número de *folds* en el proceso de CV igual a 10. Asimismo, definimos una malla de parámetros de los cuales CV seleccionará a los mejores:

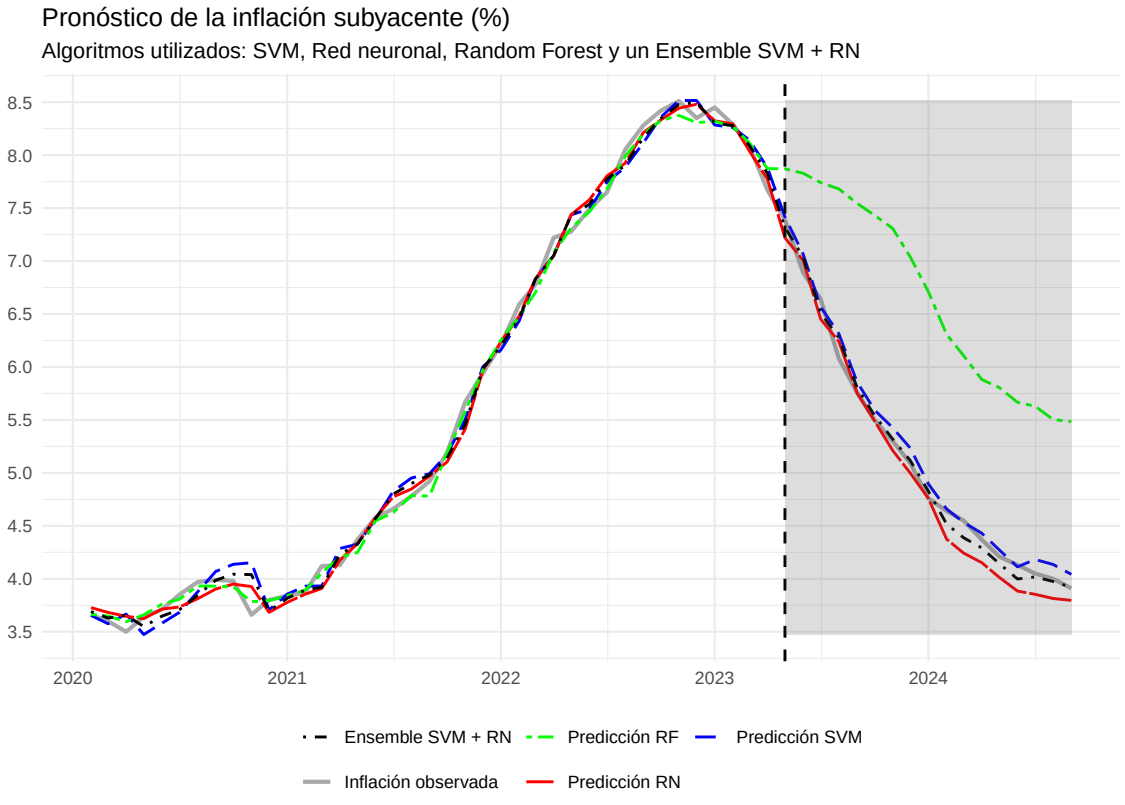
```r
control <- trainControl(method = "boot", number = 10)

set.seed(1)
modelo_nn <- train(
  infl_sub ~ infl_sub_lag1 + shock_sal_lag + infl_sub_lag6,
  data = as.ts(set_entrenamiento),      # Datos
  method = "nnet",                # Método para redes neuronales
  trControl = control,            # Configuración de cross-validation
  tuneGrid = expand.grid(size = seq(1, 15), decay = c(0.01, 0.05, 0.1)),
  preProcess = c("center", "scale"),  # Normalización de los datos
  linout = TRUE,                  # Para regresión
  trace = FALSE
)
# Muestra el mejor modelo y los resultados de cross-validation
print(modelo_nn)
```

```
## Neural Network
##
## 94 samples
##  3 predictor
##
## Pre-processing: centered (3), scaled (3)
## Resampling: Bootstrapped (10 reps)
## Summary of sample sizes: 94, 94, 94, 94, 94, 94, ...
## Resampling results across tuning parameters:
##
##   size  decay  RMSE       Rsquared   MAE
##    1    0.01   0.1564715  0.9901004  0.1250644
##    1    0.05   0.1785328  0.9872393  0.1419023
##    1    0.10   0.1942779  0.9850087  0.1537078
##    2    0.01   0.1430857  0.9916022  0.1112209
##    2    0.05   0.1591559  0.9896354  0.1234740
##    2    0.10   0.1664045  0.9889468  0.1323649
##    3    0.01   0.1423873  0.9918873  0.1080079
##    3    0.05   0.1459278  0.9914473  0.1128254
##    3    0.10   0.1571710  0.9899479  0.1244482
##    4    0.01   0.1392041  0.9921421  0.1069819
##    4    0.05   0.1452682  0.9915475  0.1130510
##    4    0.10   0.1529079  0.9906218  0.1216033
##    5    0.01   0.1354945  0.9925101  0.1051135
##    5    0.05   0.1446901  0.9916229  0.1129133
##    5    0.10   0.1528788  0.9905878  0.1214638
##    6    0.01   0.1369604  0.9924905  0.1067804
##    6    0.05   0.1439068  0.9916783  0.1126916
##    6    0.10   0.1516251  0.9907101  0.1203720
##    7    0.01   0.1357853  0.9925981  0.1050122
##    7    0.05   0.1424040  0.9918398  0.1114361
##    7    0.10   0.1488529  0.9909829  0.1177629
##    8    0.01   0.1361315  0.9925917  0.1044350
##    8    0.05   0.1415429  0.9919295  0.1109504
##    8    0.10   0.1473228  0.9911660  0.1165966
##    9    0.01   0.1394641  0.9922348  0.1065025
##    9    0.05   0.1406364  0.9919811  0.1106308
##    9    0.10   0.1460882  0.9913110  0.1154313
##   10    0.01   0.1418272  0.9919691  0.1088932
```

```
##    10     0.05    0.1401077  0.9921488  0.1086318
##    10     0.10    0.1454704  0.9913712  0.1150258
##    11     0.01    0.1388188  0.9923015  0.1057294
##    11     0.05    0.1402371  0.9921121  0.1090963
##    11     0.10    0.1444131  0.9914865  0.1141884
##    12     0.01    0.1441456  0.9919545  0.1105975
##    12     0.05    0.1403923  0.9920156  0.1100132
##    12     0.10    0.1432021  0.9915987  0.1132627
##    13     0.01    0.1393133  0.9921277  0.1059198
##    13     0.05    0.1388560  0.9922567  0.1089607
##    13     0.10    0.1427168  0.9916670  0.1127959
##    14     0.01    0.1340291  0.9926542  0.1035729
##    14     0.05    0.1400358  0.9921822  0.1083757
##    14     0.10    0.1423295  0.9917235  0.1123464
##    15     0.01    0.1409211  0.9922213  0.1078098
##    15     0.05    0.1406789  0.9920858  0.1090081
##    15     0.10    0.1424610  0.9916823  0.1125479
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 14 and decay = 0.01.
```

Es decir, el mejor modelo utiliza 14 nodos en la capa oculta, y el valor del regularizador *decay* es de 0.01.

Finalmente, presentamos una gráfica con la inflación subyacente observada y el pronóstico de cada uno de estos algoritmos. En ella incluimos un cuarto algoritmo de tipo *Ensemble*, el cual está compuesto por la red neuronal y el modelo de SVM que entrenamos, y cuyo pronóstico no es más que la media de estos dos.



Fuente: Sistema de información económica del Banxico. El area sombreada representa el sample test

2.2. Predicción del empleo (asegurados totales IMSS) para el mes de octubre 2024 (pública en noviembre). Usar como máximo desde el periodo 2015.

El primer algoritmo fue también un SVM:

```
set.seed(1)
modelo_svm <- train(formula,
                    data = as.ts(set_entrenamiento),
                    method = "svmLinear",
                    preProcess = c("center","scale"),
                    tuneGrid = expand.grid(C = seq(0.1, 2, length = 20)))
modelo_svm
```

```
## Support Vector Machines with Linear Kernel
##
## 89 samples
## 17 predictors
##
## Pre-processing: centered (17), scaled (17)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 89, 89, 89, 89, 89, 89, ...
## Resampling results across tuning parameters:
##
##   C    RMSE      Rsquared   MAE
##   0.1  182.6881  0.9645712  135.0448
##   0.2  185.4454  0.9628478  136.1741
##   0.3  185.9889  0.9625810  136.8434
##   0.4  186.5934  0.9622807  137.2292
##   0.5  187.9621  0.9615601  137.7676
##   0.6  188.5517  0.9612420  137.9442
##   0.7  188.7038  0.9611166  137.8533
##   0.8  188.6095  0.9611568  137.6886
##   0.9  188.7886  0.9609805  137.9252
##   1.0  189.0983  0.9608228  138.3378
##   1.1  188.8657  0.9609262  138.4522
##   1.2  188.6544  0.9610081  138.5929
##   1.3  188.6296  0.9610391  138.8116
##   1.4  188.6656  0.9610838  139.0342
##   1.5  188.4539  0.9611044  138.9178
##   1.6  188.3383  0.9611008  138.9388
##   1.7  188.2633  0.9610982  138.9242
##   1.8  188.3556  0.9610443  138.9947
##   1.9  188.5875  0.9609708  139.1993
##   2.0  188.6488  0.9609443  139.2728
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was C = 0.1.
```

El regularizador C seleccionado mediante CV es igual a 0.1.

El segundo algoritmo fue un RF. Utilizamos la otra librería mencionada en este ejemplo, puesto que arrojaba mejores resultados en este caso específico. A continuación, se muestra el código con los resultados de CV y el modelo elegido:

```r
set.seed(1)
modelo_rf2 <-
  train(formula,
        data = as.ts(set_entrenamiento),
        method = 'rf',
        tuneLength = 10,
        prox = TRUE       # Extra information
        )
modelo_rf2 # resultados de CV
```

```
## Random Forest
##
## 89 samples
## 17 predictors
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 89, 89, 89, 89, 89, 89, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##    2    174.9482  0.9703127  134.2716
##    3    173.1370  0.9693087  131.1075
##    5    173.0259  0.9685475  129.1573
##    7    174.8406  0.9679479  130.1126
##    8    174.3592  0.9678953  129.5439
##   10    175.3181  0.9676555  129.9999
##   12    174.8820  0.9678328  129.1645
##   13    175.4030  0.9677802  129.1924
##   15    175.3946  0.9676724  128.8418
##   17    174.9243  0.9679299  128.9194
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 5.
```

```r
modelo_rf2$finalModel # el modelo elegido
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, proximity = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 28125.17
##                     % Var explained: 97.07
```

Es decir, CV indica que debemos usar 5 variables en cada árbol del RF.

Como tercer opción utilizamos LASSO. Para estimar el valor de λ óptimamente mediante CV, elegimos la función `cv.glmnet()` de la librería `glmnet`, que automatiza el proceso de CV y arroja el λ óptimo:

```r
set.seed(1)
modelo_lasso <- cv.glmnet(x_train, y_train, alpha = 1) #alpha = 1 para LASSO
modelo_lasso$lambda.min
```

```
## [1] 10.14123
```

El valor del regularizador sería entonces 10.1412267 según CV. Asimismo, podemos observar cuáles son los coeficientes de cada variable del modelo, e implícitamente, las que según el algoritmo son las que mejor explican el empleo formal:

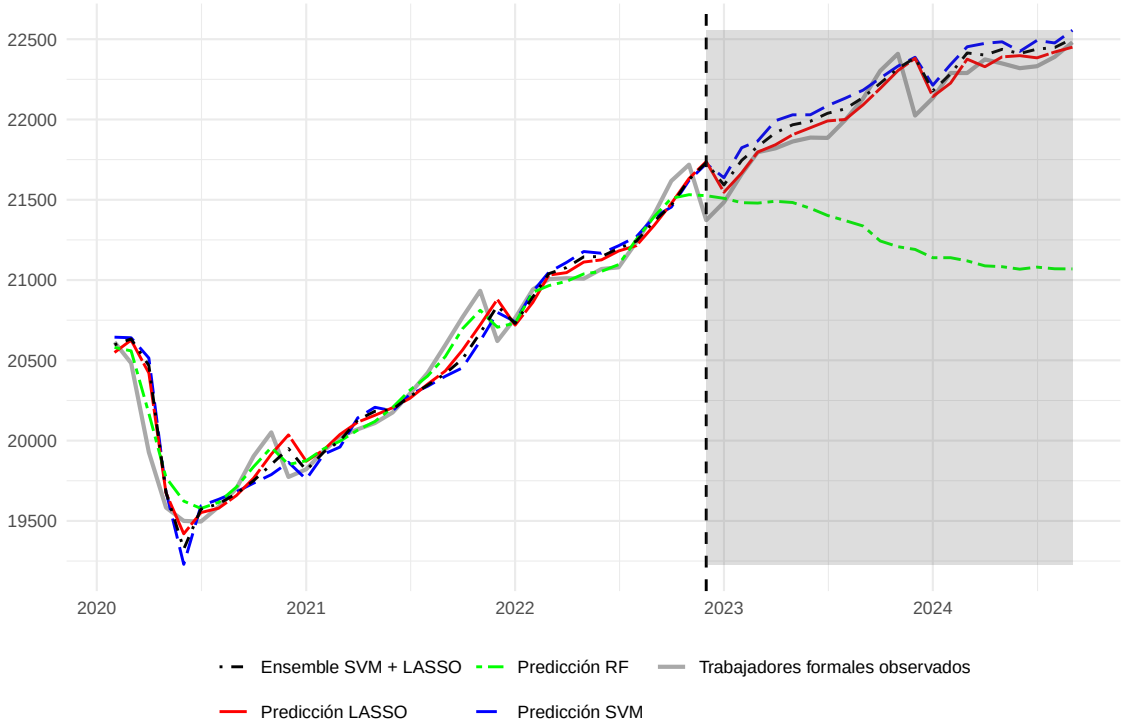
```r
round(coef(modelo_lasso, s = "lambda.min"), 2)
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##                               s1
## (Intercept)           -1339.91
## date                      0.14
## inpc_lag                  .
## shock_sal_lag             .
## shock_energ_lag          22.95
## shock_tc_lag              .
## icco_lag                  0.69
## remuneraciones_lag        .
## productividad_lag         .
## minwage_lag               .
## desocup_lag               .
## tasa_informalidad_lag     .
## igae_lag                 21.25
## ta_lag1                   0.66
## ta_lag2                   .
## ta_lag3                   .
## ta_lag4                   0.14
## ta_lag5                   .
## ta_lag6                   0.03
```

Finalmente, al igual que en el inciso anterior, propusimos un último algoritmo de tipo *Ensemble* compuesto por SVM y LASSO. La siguiente figura muestra los resultados:

Pronóstico del empleo formal (en miles)

Algoritmos utilizados: SVM, LASSO y Random Forest



Fuente: IMSS y sistema de información económica del Banxico. El area sombreada representa el sample test

2.3. Predicción de la pobreza laboral para el Tercer Trimestre 2024 (pública en noviembre).

Primer algoritmo: SVM. A continuación el código de entrenamiento y CV:

```
set.seed(1)
modelo_svm <- train(formula,
                    data = as.ts(set_entrenamiento),
                    method = "svmLinear",
                    preProcess = c("center","scale"),
                    tuneGrid = expand.grid(C = seq(0.1, 2, length = 20)))
#View the model
modelo_svm
```

```
## Support Vector Machines with Linear Kernel
##
## 28 samples
## 11 predictors
##
## Pre-processing: centered (11), scaled (11)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 28, 28, 28, 28, 28, 28, ...
## Resampling results across tuning parameters:
##
##   C    RMSE      Rsquared   MAE
##   0.1  1.579853  0.3510995  1.220362
##   0.2  1.661722  0.3376567  1.266628
##   0.3  1.745989  0.3254559  1.310325
##   0.4  1.805906  0.3246055  1.341802
##   0.5  1.859230  0.3218501  1.372831
##   0.6  1.904990  0.3166319  1.397899
##   0.7  1.943596  0.3081082  1.417843
##   0.8  1.985960  0.2987424  1.441788
##   0.9  2.028437  0.2945220  1.465568
##   1.0  2.055487  0.2911948  1.483109
##   1.1  2.078274  0.2873933  1.498395
##   1.2  2.100615  0.2857575  1.511982
##   1.3  2.122729  0.2840162  1.523508
##   1.4  2.147316  0.2819742  1.536611
##   1.5  2.172968  0.2779404  1.550881
##   1.6  2.196386  0.2736256  1.564484
##   1.7  2.204796  0.2706628  1.571756
##   1.8  2.211931  0.2680936  1.578502
##   1.9  2.221039  0.2651341  1.586179
##   2.0  2.234224  0.2623703  1.595885
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was C = 0.1.
```

El regularizador C seleccionado mediante CV es igual a 0.1.

Segundo algoritmo: RF mediante `caret`:

```
set.seed(1)
cv_randomF <- rfcv(trainx = as.ts(set_entrenamiento)[,-2],
```

```
                  trainy = as.ts(set_entrenamiento)[,2],
                  step = 0.9,
                  cv.fold = 5)

# nos dice cuál es el número óptimo
mtry_CV <- cv_randomF$error.cv[which.min(cv_randomF$error.cv)]
mtry_CV <- as.numeric(names(mtry_CV))
mtry_CV
```

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

Lo anterior indica que el número de variables a utilizar sería 1 según los resultados de CV. Con esta información, podemos entrenar el modelo de RF; por simplicidad, fijaremos en 500 el número de árboles del algoritmo, aunque podríamos buscar el número óptimo mediante CV. El siguiente código entrena el algoritmo:

```
set.seed(2)
modelo_rf = randomForest(formula = formula,
                         data = set_entrenamiento,
                         ntree = 500,
                         mtry = mtry_CV)
modelo_rf
```

```
##
## Call:
##  randomForest(formula = formula, data = set_entrenamiento, ntree = 500,      mtry = mtry_CV)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 2.297583
##                     % Var explained: 28.14
```

Finalmente, estimamos un modelo LASSO:

```
set.seed(2)
modelo_lasso <- cv.glmnet(x_train, y_train, alpha = 1)
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold
```

```
modelo_lasso$lambda.min
```

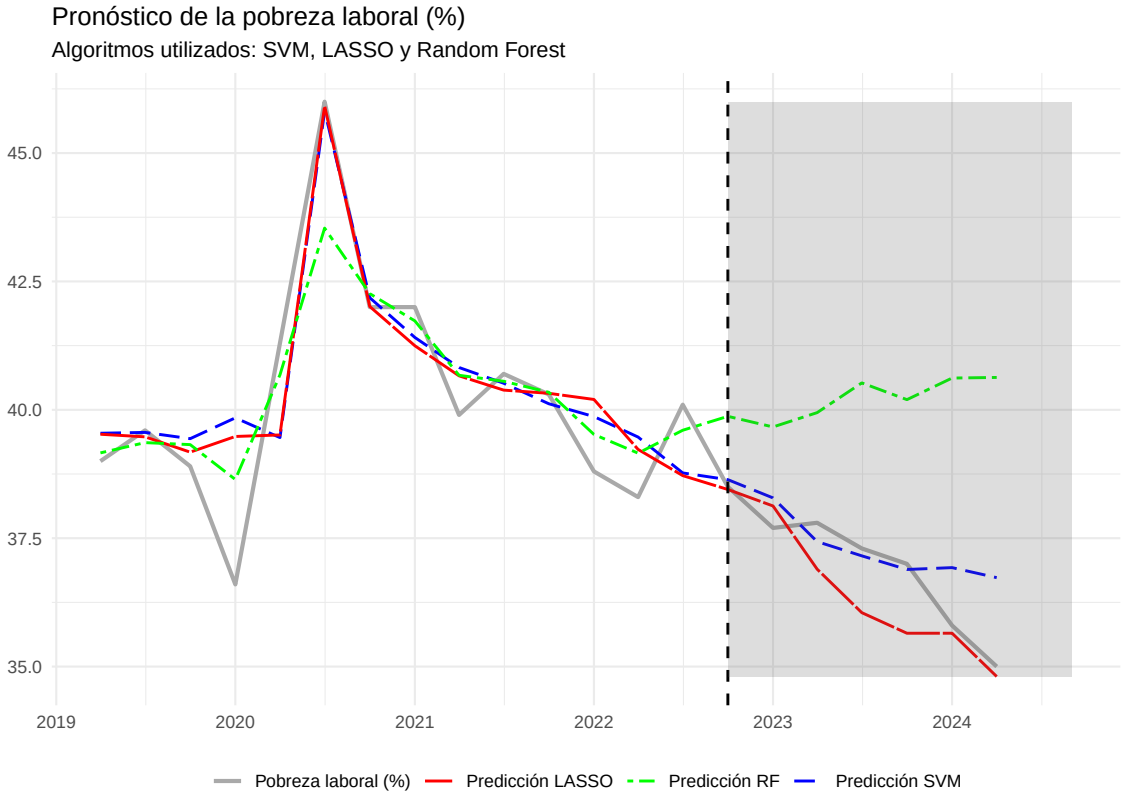
```
## [1] 0.06034792
```

El valor del regularizador sería entonces 0.0603479 según CV. Asimismo, podemos observar cuáles son los coeficientes de cada variable del modelo, e implícitamente, las que según el algoritmo son las que mejor explican la pobreza:

```
round(coef(modelo_lasso, s = "lambda.min"), 2)
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                            s1
## (Intercept)           70.41
## tasa_deso_lag          .
## tasa_ocup_inf_lag     -0.17
## pib_real_lag           0.00
## infl_sub_lag           .
## tc_fix_lag             .
## tiie_28_lag           -0.04
## w_min_lag             -0.02
## inflacion_lag          0.20
## Pobreza_laboral_lag1   .
## Pobreza_laboral_lag2   0.13
## Pobreza_laboral_lag3   0.04
```

Por último, la comparación:



Fuente: IMSS y sistema de información económica del Banxico. El area sombreada representa el sample test

Nombres: Oscar López, Max Serna y Osvaldo Valdés

Problema 1:

- **Modelo ganador**: Ensemble SVM + NN.
- **Variables incluidas**:
  - Seis rezagos de la inflación subyacente (variable dependiente)
  - El primer rezago de las siguientes variables (entre comillas sus códigos del SIE de Banxico:
    * INPC "SP30577",
    * INPC subyacente total "SP74625",
    * INPC subyacente de mercancías "SP74626",
    * INPC subyacente de servicios "SP74628",
    * Indicador de Choques de oferta relacionado a salarios "SP74826",
    * Indicador de Choques de oferta relacionado a energéticos "SP74827",
    * Indicador de Choques de oferta relacionado a tipo de cambio "SP74828",
    * Cetes a 28 días "SF282",
    * Cetes 91 dias "SF3338",
    * Cetes 128 días "SF3270",
    * Cetes 364 dias "SF3367",
    * Indicador de confianzar del consumidor desestacionalizado "SR16064",
    * Tipo de cambio "SF17908",
    * M1 "SF1",
    * TIIE a 28 días "SF283" y
    * Inf. subyacente para el mes en curso "SR14313"
- **Predicción para octubre 2024**: 3.9%

Problema 2:

- **Modelo ganador**: LASSO.
- **Variables incluidas**:
  - Seis rezagos del empleo formal (variable dependiente)
  - El primer rezago de las siguientes variables (entre comillas sus códigos del SIE de Banxico:
    * INPC "SP30577",
    * Indicador de Choques de oferta relacionado a salarios "SP74826",
    * Indicador de Choques de oferta relacionado a energéticos "SP74827",
    * Indicador de Choques de oferta relacionado a tipo de cambio "SP74828",
    * Indicador de confianzar del consumidor desestacionalizado "SR16064",
    * Remuneraciones medias reales por persona ocupada "SL11439",
    * Productividad laboral por persona ocupada "SL11447",
    * Tasa de desocupación abierta en áreas urbanas,
    * Tasa de informalidad laboral,
    * IGAE, y
    * Salarios Mínimos General Índice Real, Dic2018=100 "SL11297"
- **Predicción para octubre 2024**: 22,450

Problema 3:

- **Modelo ganador**: LASSO.
- **Variables incluidas**:
  - Tres rezagos de pobreza laboral (variable dependiente)
  - El primer rezago de las siguientes variables:
    * Tasa de desocupación,
    * Tasa de ocupación informal,
    * PIB real,
    * Inflación subyacente,
    * Tipo de cambio Fix,
    * TIIE 28 días,
    * Salario mínimo, e
    * Inflación general
- **Predicción para el tercer trimestre de 2024**: 34.8%