Econ 491 Applied Machine Learning Assignment 3

Aidan Lynde

Nov 12th

1. LASSO & Thresholding

a.) Show if $\lambda \geq \left| \frac{2}{n} \sum_{i=1}^{n} Y_i X_i \right|$, then $\hat{\beta}_\lambda = 0$ ...

To find minimum, consider case when $\beta \geq 0$. Objective function becomes:

$$\frac{1}{n} \sum_{i=1}^{n} (Y_i - X_i \beta)^2 + \lambda \beta$$

Differentiable with respect to β, and set to zero

$$\frac{-2}{n} \sum_{i=1}^{n} X_i (Y_i - X_i \beta) + \lambda = 0$$

Solving for β

$$\beta = \frac{\sum_{i=1}^{n} X_i Y_i}{\sum_{i=1}^{n} X_i^2} - \frac{n\lambda}{2\sum_{i=1}^{n} X_i^2}$$

Since this is under the assumption $\beta \geq 0$, if this calculated β is negative, it means the minimum occurs at $\beta = 0$.

Similarly for $\beta < 0$, the derivative of the objective function is:

$$\frac{-2}{n} \sum_{i=1}^{n} X_i (Y_i - X_i \beta) - \lambda = 0 \quad \text{(solving for } \beta\text{)} : \beta = \frac{\sum_{i=1}^{n} X_i Y_i}{\sum_{i=1}^{n} X_i^2} + \frac{n\lambda}{2\sum_{i=1}^{n} X_i^2}$$

if this β is positive, then the minimum is at $\beta = 0$.

b.) Compute $\hat{\beta}_\lambda$ for $0 \le \lambda < \left| \frac{2}{n} \sum_{i=1}^{n} Y_i X_i \right|$.

From results of (a) we know that:

if $\lambda \ge \left| \frac{2}{n} \sum_{i=1}^{n} Y_i X_i \right|, \hat{\beta}_\lambda = 0$

if $\lambda < \left| \frac{2}{n} \sum_{i=1}^{n} Y_i X_i \right|$, we need to solve the derived formulas for β and check their sign.

therefore:

$$\hat{\beta}_\lambda = \max\left(0, \frac{\sum_{i=1}^{n} X_i Y_i}{\sum_{i=1}^{n} X_i^2} - \frac{n\lambda}{2\sum_{i=1}^{n} X_i^2}\right)$$

or

$$\hat{\beta}_\lambda = \min\left(0, \frac{\sum_{i=1}^{n} X_i Y_i}{\sum_{i=1}^{n} X_i^2} + \frac{n\lambda}{2\sum_{i=1}^{n} X_i^2}\right)$$

As λ increases, the likelihood of $\hat{\beta}_\lambda$ being zero increases, hence promoting sparsity.

```
##############################
## Inflation Forcasting Exercise ##
##############################



# Data Preperation:
import pandas as pd

file_path = "/Users/aidanlynde/ECON491/Assignment3/code/data/2021-12.csv"
df = pd.read_csv(file_path, header=[0, 1])

df = df.dropna(axis=1)
df = df.iloc[:, :105]
   # drop the 'Transform:' label
df.columns = df.columns.droplevel(1)
   # extract the transformation codes for each column
transformations = df.iloc[0, 1:].astype(int)
   # drop the 'Transform:' row
df = df.drop(df.index[0])
   # set the index to 'sasdate'
df.set_index('sasdate', inplace=True)



# Data transformations
   # define the columns for each group
output_and_income = ['RPI', 'W875RX1', 'INDPRO', 'IPFPNSS', 'IPFINAL', 'IPCONGD',
'IPDCONGD',
                     'IPNCONGD', 'IPBUSEQ', 'IPMAT', 'IPDMAT', 'IPNMAT', 'IPMANSICS',
                       'IPB51222s', 'IPFUELS', ' NAPMPI', 'CUMFNS']
labor_market = ['HWI', 'HWIURATIO', 'CLF16OV', 'CE16OV', 'UNRATE', 'UEMPMEAN',
'UEMPLT5',
                'UEMP5TO14', 'UEMP15OV', 'UEMP15T26', 'UEMP27OV', 'CLAIMSx', 'PAYEMS',
                  'USGOOD', 'CES1021000001', 'USCONS', 'MANEMP', 'DMANEMP',
'NDMANEMP',
                    'SRVPRD', 'USTPU', 'USWTRADE', 'USTRADE', 'USFIRE', 'USGOVT',
                      'CES0600000007', 'AWOTMAN', 'AWHMAN', 'NAPMEI', 'CES0600000008',
                        'CES2000000008', 'CES3000000008']
consumption_and_orders = ['HOUST', 'HOUSTNE', 'HOUSTMW', 'HOUSTS', 'HOUSTW', 'PERMIT',
                          'PERMITNE', 'PERMITMW', 'PERMITS', 'PERMITW']
orders_and_inventories = ['DPCERA3M086SBEA', 'CMRMTSPLx', 'RETAILx', 'NAPM',
'NAPMNOI',
```

```
                        'NAPMSDI', 'NAPMII', 'ACOGNO', 'AMDMNOx', 'ANDENOx',
'AMDMUOx',
                          'BUSINVx', 'ISRATIOx', 'UMCSENTx']
money_and_credit = ['M1SL', 'M2SL', 'M2REAL', 'AMBSL', 'TOTRESNS', 'NONBORRES',
'BUSLOANS',
                    'REALLN', 'NONREVSL', 'CONSPI', 'MZMSL', 'DTCOLNVHFNM',
'DTCTHFNM', 'INVEST']
interest_rate_and_exchange_rates = ['FEDFUNDS', 'CP3Mx', 'TB3MS', 'TB6MS', 'GS1',
'GS5', 'GS10', 'AAA',
                                    'BAA', 'COMPAPFFx', 'TB3SMFFM', 'TB6SMFFM',
'T1YFFM', 'T5YFFM', 'T10YFFM',
                                      'AAAFFM', 'BAAFFM', 'TWEXMMTH', 'EXSZUSx',
'EXJPUSx', 'EXUSUKx', 'EXCAUSx']
prices = ['PPIFGS', 'PPIFCG', 'PPIITM', 'PPICRM', 'OILPRICEx', 'PPICMM', 'NAPMPRI',
'CPIAUCSL', 'CPIAPPSL',
          'CPITRNSL', 'CPIMEDSL', 'CUSR0000SAC', 'CUUR0000SAD', 'CUSR0000SAS',
'CPIULFSL', 'CUUR0000SA0L2',
            'CUSR0000SA0L5', 'PCEPI', 'DDURRG3M086SBEA', 'DNDGRG3M086SBEA',
'DSERRG3M086SBEA']
stock_market = ['S&P 500', 'S&P: indust', 'S&P div yield', 'S&P PE ratio']
   # create a dictionary for grouped data
groups = {
   'output_and_income': output_and_income,
   'labor_market': labor_market,
   'consumption_and_orders': consumption_and_orders,
   'orders_and_inventories': orders_and_inventories,
   'money_and_credit': money_and_credit,
   'interest_rate_and_exchange_rates': interest_rate_and_exchange_rates,
   'prices': prices,
   'stock_market': stock_market
}
   # loop over columns to apply transformations
import numpy as np

for col, transform_code in transformations.items():
   if transform_code == 1:
       pass  # No transformation
   elif transform_code == 2:
       df[col] = df[col].diff()
   elif transform_code == 3:
       df[col] = df[col].diff().diff()
   elif transform_code == 4:
```

```
        df[col] = df[col].apply(lambda x: np.log(x)).diff()
    elif transform_code == 5:
        df[col] = df[col].apply(lambda x: np.log(x))
    elif transform_code == 6:
        df[col] = df[col].apply(lambda x: np.log(x)).diff()
    elif transform_code == 7:
        df[col] = df[col].apply(lambda x: np.log(x)).diff().diff()
    # drop rows with NaN values due to differences or log operations
df.dropna(inplace=True)
df.index = pd.to_datetime(df.index)
df = df.asfreq('MS')
    # construct inflation series for CPIAUSCL
delta_yt = df['CPIAUCSL'].diff()
df['CPIAUCSL_inflation'] = delta_yt / df['CPIAUCSL'].shift()
df.dropna(inplace=True)



# 1. Autoregressive model (AR):
from statsmodels.tsa.ar_model import AutoReg

best_bic = np.inf
best_order = None
endog = df['CPIAUCSL_inflation']

for p in range(1, 21):
    model = AutoReg(endog, lags=p, trend='c')
    results = model.fit()
    if results.bic < best_bic:
        best_bic = results.bic
        best_order = p

print(f"Best order by BIC: {best_order}")



# 2. AR + Principal Component Regression (PCR):
    # determine number of PCs
from sklearn.preprocessing import StandardScaler

X = df.drop('CPIAUCSL_inflation', axis=1)
scaler = StandardScaler()
```

```
X_standardized = scaler.fit_transform(X)

from sklearn.decomposition import PCA

pca = PCA()
pca_result = pca.fit_transform(X_standardized)

import matplotlib.pyplot as plt

explained_variance = pca.explained_variance_ratio_.cumsum()
plt.figure(figsize=(10,5))
plt.plot(range(1, len(explained_variance)+1), explained_variance, marker='o',
linestyle='--')
plt.title('Explained Variance by Components')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.show()

num_components = np.where(explained_variance > 0.95)[0][0] + 1
print(f"Number of components that explain >=95% variance: {num_components}")

   # fit the PCR using the chosen number of components (17)
from sklearn.decomposition import PCA

n_components = 17
pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(X_standardized)

from sklearn.model_selection import train_test_split

y = df['CPIAUCSL_inflation']

X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size=0.2,
random_state=42)

from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train_pca, y_train)

from sklearn.metrics import mean_squared_error
```

```
y_pred = model.predict(X_test_pca)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error using {n_components} principal components: {mse}")



# 3. Ridge Regression (PCR):
from sklearn.linear_model import RidgeCV

X_train, X_test, y_train, y_test = train_test_split(X_standardized, y, test_size=0.2,
random_state=42)

alphas = np.logspace(-6, 6, 13)
ridge_cv = RidgeCV(alphas=alphas, store_cv_values=True)
ridge_cv.fit(X_train, y_train)

best_alpha = ridge_cv.alpha_
print(f"Best alpha: {best_alpha}")

y_pred_ridge = ridge_cv.predict(X_test)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
print(f"Mean Squared Error using Ridge Regression: {mse_ridge}")



# 4. Adaptive LASSO Regression :
from sklearn.linear_model import ElasticNet, LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import numpy as np
   # standardize predictors
scaler = StandardScaler()
X_standardized = scaler.fit_transform(X)
   # initial Linear Regression for weights
lin_reg = LinearRegression().fit(X_standardized, y)
weights = 1 / np.abs(lin_reg.coef_)
   # split the data
X_train, X_test, y_train, y_test = train_test_split(X_standardized, y, test_size=0.2,
random_state=42)
   # modify the data using weights
X_train_weighted = X_train * weights
```

```
X_test_weighted = X_test * weights
   # adaptive Elastic Net with BIC
best_bic = np.inf
best_alpha = None
l1_ratio = 0.5
max_iter = 50000
for alpha in np.logspace(-6, 6, 100):
   elastic_net = ElasticNet(alpha=alpha, l1_ratio=l1_ratio, max_iter=max_iter)
   elastic_net.fit(X_train_weighted, y_train)
   y_pred = elastic_net.predict(X_test_weighted)
   mse = mean_squared_error(y_test, y_pred)
   bic = len(y_test) * np.log(mse) + np.log(len(y_test)) *
np.count_nonzero(elastic_net.coef_)
   if bic < best_bic:
       best_bic = bic
       best_alpha = alpha
print(f"Best alpha by BIC: {best_alpha}")
   # Fit the final model
final_model = ElasticNet(alpha=best_alpha, l1_ratio=l1_ratio, max_iter=max_iter)
final_model.fit(X_train_weighted, y_train)



# 5. Nueral Network Regression
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

   # define the model
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))
   # compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
   # fit the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=100,
batch_size=10)
   # evaluate the model
mse = model.evaluate(X_test, y_test)
print(f"Mean Squared Error: {mse}")
```

```
# 6. Random Forests

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np
import shap

window_size = 492
cumulative_error = 0
errors = []

all_shap_values = []

for start in range(len(df) - window_size):
   end = start + window_size
   train = df.iloc[start:end]
   test = df.iloc[end:end+1]

   X_train = train.drop('CPIAUCSL_inflation', axis=1)
   y_train = train['CPIAUCSL_inflation']
   X_test = test.drop('CPIAUCSL_inflation', axis=1)
   y_test = test['CPIAUCSL_inflation']

       # train the Random Forest model
   rf = RandomForestRegressor()
   rf.fit(X_train, y_train)

       # predict and calculate error
   y_pred = rf.predict(X_test)
   error = mean_squared_error(y_test, y_pred)
   cumulative_error += error
   errors.append(cumulative_error)

       # compute SHAP values
   explainer = shap.Explainer(rf, X_train)
   shap_values = explainer(X_test)
   all_shap_values.append(np.abs(shap_values.values).mean(axis=0))
```

```
   # plot cumulative errors
plt.plot(errors)
plt.xlabel('Forecasting Window')
plt.ylabel('Cumulative Squared Error')
plt.title('Cumulative Squared Errors Over Time')
plt.show()

   # aggregate SHAP values
avg_shap_values = np.mean(np.array(all_shap_values), axis=0)
shap_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Avg_SHAP':
avg_shap_values})
shap_importance_df = shap_importance_df.sort_values(by='Avg_SHAP', ascending=False)

   # display SHAP importance DataFrame
print(shap_importance_df)

# Analysis Questions:
# 1.) Is there a Winner Model?:
# From the provided results, we can compare the Mean Squared Errors (MSE) of different
models:
# Principal Component Regression (PCR) with 17 components: MSE = 4.407155709410761e-06
# Ridge Regression: MSE = 3.6793841558601053e-06
# Neural Network Regression: Final Validation MSE = 3.5414e-04
# The Ridge Regression model has the lowest MSE among the models for which MSE is
reported,
# suggesting it might be the best performing model in terms of prediction accuracy.
#
# 2.) Do the results change over the sample, especially after Covid-19?
# Upon inspecting the cumulative squared errors plot,
# it is evident that the forecasting errors for all models increase around the early
months of 2020,
# coinciding with the global onset of Covid-19.
# This suggests that the pandemic introduced new economic variables or increased
volatility,
# causing models trained on pre-Covid data to be less accurate during the pandemic.
```

Terminal Output:
Best order by BIC: 1
Number of components that explain >=95% variance: 17
Mean Squared Error using 17 principal components: 4.407280210539396e-06
Best alpha: 0.01
Mean Squared Error using Ridge Regression: 3.6793841558601053e-06
Best alpha by BIC: 174.7528400007683
Mean Squared Error: 0.0006525031640194356