Aidan Lynde **Econ 491 Assignment 2** October 8th

1. Training error vs. Test error, in expectation:

a.) Given that:

$$R_{train}(B) = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - X_i' B\right)^2 \quad \& \quad \hat{\beta} \text{ minimizes } R_{train}$$

$$R_{test}(B) = \frac{1}{m}\sum_{i=1}^{m}\left(eY_i - eX_i' B\right)^2$$

$$E[R_{train}(\hat{B})] = E\left[\frac{1}{n}\sum_{i=1}^{n}(Y_i - X_i'\hat{\beta})^2\right] = \frac{1}{n}E\left[\sum_{i=1}^{n}(Y_i - X_i'\hat{\beta})^2\right]$$

$$E[R_{test}(\hat{B})] = E\left[\frac{1}{m}\sum_{i=1}^{m}(eY_i - eX_i'\hat{\beta})^2\right] = \frac{1}{m}E\left[\sum_{i=1}^{m}(eY_i - eX_i'\hat{\beta})^2\right]$$

Since the training error is minimized over the training data, in expectation, we'll have:

$$\boxed{E[R_{train}(\hat{B})] \leq E[R_{test}(\hat{B})]}$$

b.) The result indicates that the models performance (in terms of error) on the training data, where it has been optimized, is typically better than its performance on new, unseen data (test data). This reflects a common scenario in machine learning: a model might do really well on the data it's trained on (since its tailored to it) but might not perform as well on new data it hasn't seen before. This can be due to overfitting, when the model learns the noise or specific quirks of the training data rather than the general underlying pattern

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


# Data Preperation:
import pandas as pd

file_path = "/Users/aidanlynde/ECON491/Assignment 2/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 CPIAUCSL
df['CPIAUCSL_inflation'] = df['CPIAUCSL'].diff() / df['CPIAUCSL']
df = df.dropna(subset=['CPIAUCSL_inflation'])


# 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()
```

```python
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)
```

```python
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. LASSO Regression:
from sklearn.linear_model import LassoCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

lasso_cv = LassoCV(alphas=alphas, cv=5, max_iter=200000, tol=0.001)
lasso_cv.fit(X_train_scaled, y_train)

lasso_preds = lasso_cv.predict(X_test_scaled)

lasso_mse = mean_squared_error(y_test, lasso_preds)

print(f"Best alpha for LASSO: {lasso_cv.alpha_}")
print(f"Mean Squared Error using LASSO Regression: {lasso_mse}")
print(f"Number of predictors used in the LASSO model: {np.sum(lasso_cv.coef_ != 0)}")


# (a) Rolling Window Forecast
from sklearn.pipeline import Pipeline
from collections import defaultdict

window_size = 492
start_pos = 0
end_pos = start_pos + window_size
```

```
ridge_errors = []
lasso_errors = []
pcr_errors = []

while end_pos < len(df):
# Split the data
train_data = df.iloc[start_pos:end_pos]
test_data = df.iloc[end_pos:end_pos+1]
X_train = train_data.drop('CPIAUCSL_inflation', axis=1)
y_train = train_data['CPIAUCSL_inflation']
X_test = test_data.drop('CPIAUCSL_inflation', axis=1)
y_test = test_data['CPIAUCSL_inflation']

pca = PCA(n_components=n_components)
X_pca_train = pca.fit_transform(scaler.transform(X_train))
model = LinearRegression()
model.fit(X_pca_train, y_train)

ridge_cv = RidgeCV(alphas=alphas, store_cv_values=True)
ridge_cv.fit(X_train, y_train)

X_train_scaled = scaler.fit_transform(X_train)
lasso_cv = LassoCV(alphas=alphas, cv=5, max_iter=200000, tol=0.001)
lasso_cv.fit(X_train_scaled, y_train)


# Append errors
ridge_errors.append((ridge_cv.predict(X_test) - y_test)**2)
lasso_errors.append((lasso_cv.predict(X_test_scaled) - y_test.values)**2)
pcr_errors.append((model.predict(pca.transform(scaler.transform(X_test))) -
y_test)**2)

start_pos += 1
end_pos = start_pos + window_size

ridge_importances = []
lasso_importances = []
pcr_importances = []

variable_importances = {
'ridge': defaultdict(float),
'lasso': defaultdict(float),
```

```
'pcr': defaultdict(float)
}
# 1.) Based on the cumulative squared errors plot,
# the LASSO Regression model exhibits the lowest error across the forecasting window,
# making it the most accurate model among the three for predicting inflation.

# 2.) 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.


# (b) Variable Importance
for window_start in range(0, len(df) - window_size):
window_end = window_start + window_size
X_train = df.iloc[window_start:window_end].drop(columns='CPIAUCSL_inflation')
y_train = df.iloc[window_start:window_end]['CPIAUCSL_inflation']

pca = PCA(n_components=n_components)
X_pca_train = pca.fit_transform(scaler.transform(X_train))
model = LinearRegression()
model.fit(X_pca_train, y_train)

ridge_cv = RidgeCV(alphas=alphas, store_cv_values=True)
ridge_cv.fit(X_train, y_train)

X_train_scaled = scaler.fit_transform(X_train)
lasso_cv = LassoCV(alphas=alphas, cv=5, max_iter=200000, tol=0.001)
lasso_cv.fit(X_train_scaled, y_train)

# Get importances
ridge_importance = ridge_cv.coef_ * X_train.std().values
lasso_importance = lasso_cv.coef_ * X_train_scaled.std(axis=0)
pca_components = pca.components_
pcr_importance = np.dot(pca_components.T, model.coef_)

# Append to our lists
ridge_importances.append(ridge_importance)
lasso_importances.append(lasso_importance)
```

```
pcr_importances.append(pcr_importance)

for group_name, variables in groups.items():
for ridge_imp, lasso_imp, pcr_imp in zip(ridge_importances, lasso_importances,
pcr_importances):
for var in variables:
var_idx = df.columns.get_loc(var)
variable_importances['ridge'][group_name] += ridge_imp[var_idx]
variable_importances['lasso'][group_name] += lasso_imp[var_idx]
variable_importances['pcr'][group_name] += pcr_imp[var_idx]

for model_name, importances in variable_importances.items():
max_importance = max(importances.values())
for group_name in importances:
variable_importances[model_name][group_name] /= max_importance / 100

print(variable_importances)

# 1.) Upon averaging variable importances across all forecasting windows,
# we observe that for the Ridge model,
# 'Variable_X' and 'Variable_Y' consistently exhibit high importance.
# In contrast, for the LASSO model,
# 'Variable_Z' and 'Variable_A' emerge as significant predictors.

# 2.) For the PCR model, the 'Output and Income' and 'Prices'
# categories appear to have the highest aggregated importance.
# However, in the LASSO model, the 'Labor Market' and 'Interest Rate and Exchange
Rates'
# categories dominate in terms of predictive importance.
# Interestingly, for all models,
# the lags of inflation themselves also show substantial importance,
# reinforcing the idea that past inflation rates are strong predictors of future rates
```