BEM114_final_project_working

June 7, 2024

1 Factor trading using Bayesian Neural Networks

Names: Andrew Zabelo, Daniel Wen, Kyle McCandless
Student IDs: 2176083, 2159859, 2157818

1.1 Setup

1.1.1 Imports

[1]:
```
# General
import os
import random
import numpy as np
import pandas as pd
import warnings
import statsmodels.api as sm

# Plotting
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import datetime

# ML tools
import torch
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split

# BNN specific
import pyro
import pyro.distributions as dist
from pyro.nn import PyroModule, PyroSample
import torch.nn as nn
from pyro.infer import MCMC, NUTS
from pyro.infer import Predictive

# We'll suppress some warnings from tensorflow, but feel free to
```

```
# comment out this line and see some of the efficiency gains we can get!
warnings.filterwarnings("ignore")
```

1.1.2 General helper functions

[2]:
```
# Calculates returns and prints the returns mean, vol, and Sharpe ratio for a␣
 ↪strategy
def analyze(returns, strat_name):
    strat_mean = returns.mean()
    strat_vol = returns.std()
    strat_sharpe = strat_mean / strat_vol
    print(f"\n\n{strat_name} daily returns:")
    print(f"Mean = {strat_mean:.3f}%")
    print(f"Volatility = {strat_vol:0.3f}%")
    print(f"Sharpe Ratio = {strat_sharpe:0.3f}")

# Plots the cumulative returns for a strategy versus the market returns and the␣
 ↪CAPM-implied returns
def plot_cum_returns(dates, returns, mktrf_returns, rf_returns, beta,␣
 ↪strat_name, log):
    implied_returns = rf_returns + beta * mktrf_returns
    mkt_returns = rf_returns + mktrf_returns

    dates = [datetime.datetime.strptime(str(date), '%Y%m%d') for date in dates]
    strategy_cumulative = (returns / 100 + 1.0).cumprod()
    market_cumulative = (mkt_returns / 100 + 1.0).cumprod()
    mir_cumulative = (implied_returns / 100 + 1.0).cumprod()

    # Set axis scales
    fig, ax = plt.subplots(figsize=(10, 6))
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
    if log: ax.set_yscale('log')

    # Plot data
    ax.plot(dates, strategy_cumulative, label=f'{strat_name} Portfolio Value')
    ax.plot(dates, market_cumulative, label=f'Market Portfolio Value')
    ax.plot(dates, mir_cumulative, label=f'CAPM-Implied Portfolio Value')

    plt.title(f'{strat_name} Model Performance')
    plt.xlabel('Date')
    plt.ylabel('Cumulative Portfolio Value')

    plt.legend()
    plt.show()

# Note: All io_fns and model_fns start with an 'io_' and 'model_' prefix
```

```
# respectively
def fitting_returns_data(data_path,
                         io_fn,
                         model_fn,
                         strat_name,
                         seed = None,
                         print_summary = True,
                         log = False):
    # if given, set the random seed
    if seed:
        set_seed(seed)

    # Load the data into a pandas DataFrame
    data = pd.read_csv(data_path)
    data = data[data['date'] < EVAL_CUTOFF]
    # Save for plotting
    dates = data.iloc[:,0]
    rf_returns = data.iloc[:,7]
    mktrf_returns = data.iloc[:,1]
    # Drop date and risk free rate
    data = data.iloc[:, 1:7]

    # Shift the data by one time step to create input/output pairs
    X, y = io_fn(data)

    # Fix lengths for plotting
    shift_amt = len(data) - len(X)
    dates = dates[shift_amt:]
    rf_returns = rf_returns[shift_amt:]
    mktrf_returns = mktrf_returns[shift_amt:]

    # Split the data into training and validation sets
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2)

    # Fit the model
    predictive, strat_df, return_vector = model_fn(X, y, X_train, y_train,␣
 ↪X_val, y_val)

    # Calculate alpha
    y_ols = sm.add_constant(y)
    model_OLS = sm.OLS(return_vector, y_ols).fit()
    if print_summary:
        print(model_OLS.summary())

    # Print Sharpe Ratio
    analyze(return_vector, strat_name)
```

```
    # Plot cumulative returns
    plot_cum_returns(dates, return_vector, mktrf_returns, rf_returns, model_OLS.
 ↪params[1], strat_name, log)

    return predictive, strat_df, return_vector, model_OLS

def set_seed(seed_value):
    # Adding a fixed seed from this solution: https://stackoverflow.com/
 ↪questions/32419510/how-to-get-reproducible-results-in-keras
    # 1. Set the `PYTHONHASHSEED` environment variable at a fixed value
    os.environ['PYTHONHASHSEED']=str(seed_value)

    # 2. Set the `python` built-in pseudo-random generator at a fixed value
    random.seed(seed_value)

    # 3. Set the `numpy` pseudo-random generator at a fixed value
    np.random.seed(seed_value)

    # 4. Set the `tensorflow` pseudo-random generator at a fixed value
    tf.compat.v1.set_random_seed(seed_value)
    return

def predictions_to_returns(pred_df, y):
    # Given the predictions of each factor for each day, calculate our
    # strategy for each day, and the returns for each day

    # Apply our strategy to our predictions
    strat_df = pred_df.apply(lambda row : max_predicted_factor_strat(row), axis␣
 ↪= 1)

    # Calculate our returns
    return_vector = np.multiply(strat_df,np.asarray(y)).apply(sum, axis = 1)

    return strat_df, return_vector

def predictions_to_returns_with_short(pred_df, y):
    # Given the predictions of each factor for each day, calculate our
    # strategy for each day, and the returns for each day

    # Apply our strategy to our predictions
    strat_df = pred_df.apply(lambda row :␣
 ↪max_predicted_factor_strat_with_short(row), axis = 1)

    # Calculate our returns
    return_vector = np.multiply(strat_df,np.asarray(y)).apply(sum, axis = 1)

    return strat_df, return_vector
```

```
def predictions_to_returns_max_sharpe(pred_df, std_df, y):
    # Given the predictions of each factor for each day, calculate our
    # strategy for each day, and the returns for each day

    # Apply our strategy to our predictions
    comb_df = pred_df / std_df
    strat_df = comb_df.apply(lambda row :␣
 ↪max_predicted_factor_strat_with_short(row), axis = 1)

    # Calculate our returns
    return_vector = np.multiply(strat_df,np.asarray(y)).apply(sum, axis = 1)

    return strat_df, return_vector

def predictions_to_returns_max_sharpe_long_short(pred_df, std_df, y):
    # Given the predictions of each factor for each day, calculate our
    # strategy for each day, and the returns for each day

    # Apply our strategy to our predictions
    comb_df = pred_df / std_df
    strat_df = comb_df.apply(lambda row :␣
 ↪max_predicted_factor_strat_with_short(row), axis = 1)

    # Calculate our returns
    return_vector = np.multiply(strat_df,np.asarray(y)).apply(sum, axis = 1)

    return strat_df, return_vector

def max_predicted_factor_strat(row):
    # For each day, set our strategy to be the factor with
    # the highest predicted return
    max_pred_return = max(row)
    row_list = [x == max_pred_return for x in row]
    return pd.Series(row_list)

def max_predicted_factor_strat_with_short(row):
    # For each day, long or short the factor with the highest magnitude␣
 ↪expected return
    max_pred_return = max(row)
    min_pred_return = min(row)
    if abs(max_pred_return) > abs(min_pred_return):
        row_list = [x == max_pred_return for x in row]
    else:
        row_list = [-1 if x == min_pred_return else 0 for x in row]
    return pd.Series(row_list)
```

```
def max_predicted_factor_strat_long_short(row):
    # For each day, long/short the highest/lowest expected return factors
    max_pred_return = max(row)
    min_pred_return = min(row)
    row_list = [1 if x == max_pred_return else -1 if x == min_pred_return else␣
 ↪0 for x in row]
    return pd.Series(row_list)

# Displays results for eval_df. eval_df should be completely out of sample
def predict_out_of_sample(predictive, eval_df, io_fn, strat_name):
    dates = eval_df.iloc[:,0]
    rf_returns = eval_df.iloc[:,-1]
    mktrf_returns = eval_df.iloc[:,1]

    # Drop date and risk free rate
    eval_df = eval_df.iloc[:, 1:7]

    X_eval, y_eval = io_fn(eval_df)
    x_eval_tensor = torch.tensor(X_eval.values).float()

    preds = predictive(x_eval_tensor)
    pred_means = preds['obs'].T.detach().numpy().mean(axis=2)
    pred_df = pd.DataFrame(pred_means.T)

    strat_df, return_vector = predictions_to_returns(pred_df, y_eval)

    # Fix lengths for plotting
    shift_amt = len(eval_df) - len(X_eval)
    dates = dates[shift_amt:]
    rf_returns = rf_returns[shift_amt:]
    mktrf_returns = mktrf_returns[shift_amt:]

    # Calculate alpha
    y_ols = sm.add_constant(y_eval)
    model_OLS = sm.OLS(return_vector, y_ols).fit()
    print(model_OLS.summary())

    # Print Sharpe Ratio
    analyze(return_vector, strat_name)

    # Plot cumulative returns
    plot_cum_returns(dates, return_vector, mktrf_returns, rf_returns, model_OLS.
 ↪params[1], strat_name, False)
```

1.1.3 IO functions (data manipulation)

[3]:
```
def io_day_1_lag(data):
    # Create input output pairs, where the input is the previous day of data
    # and the output is the current day of data.
    X = data.shift(1).dropna().reset_index(drop=True)
    y = data.dropna().iloc[1:,:].reset_index(drop=True)
    return X, y

def io_day_5_lag(data):
    X = data.shift(1).add_suffix('_lag1')
    for i in range(2, 6):
        shifted_data = data.shift(i).add_suffix('_lag{}'.format(i))
        X = pd.concat([X, shifted_data], axis=1)
    X = X.dropna().reset_index(drop=True)
    y = data.iloc[5:,:].reset_index(drop=True)
    return X, y
```

1.1.4 Model definitions

[4]:
```
def model_feed_forward(X, y, X_train, y_train, X_val, y_val):
    # Define the neural network model
    model = Sequential()
    model.add(Dense(32, activation='relu', input_shape=(X.shape[1],)))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(6, activation='linear'))

    # Compile the model
    model.compile(loss='mean_squared_error', optimizer='adam')

    # Train the model, verbose = 0 means reports aren't printed
    # at the end of each epoch
    model.fit(X_train, y_train, batch_size=32, epochs=50,
              validation_data=(X_val, y_val))

    # Make predictions
    predictions = model.predict(X)
    pred_df = pd.DataFrame(predictions)

    res = predictions_to_returns(pred_df, y)
    return None, res[0], res[1]

# Definition for a basic BNN as a proof of concept
class BasicBNN(PyroModule):
    def __init__(self, in_dim=6, out_dim=6, hid_dim=10, prior_scale=1.):
        super().__init__()
```

```
        self.activation = nn.Tanh()  # or nn.ReLU()
        self.layer1 = PyroModule[nn.Linear](in_dim, hid_dim)  # Input to hidden␣
↪layer
        self.layer2 = PyroModule[nn.Linear](hid_dim, out_dim)  # Hidden to␣
↪output layer

        # Set layer parameters as random variables
        self.layer1.weight = PyroSample(dist.Normal(0., prior_scale).
↪expand([hid_dim, in_dim]).to_event(2))
        self.layer1.bias = PyroSample(dist.Normal(0., prior_scale).
↪expand([hid_dim]).to_event(1))
        self.layer2.weight = PyroSample(dist.Normal(0., prior_scale).
↪expand([out_dim, hid_dim]).to_event(2))
        self.layer2.bias = PyroSample(dist.Normal(0., prior_scale).
↪expand([out_dim]).to_event(1))

    def forward(self, x, y=None):
        x = self.activation(self.layer1(x))
        mu = self.layer2(x).squeeze()
        sigma = pyro.sample("sigma", dist.Gamma(0.5, 2.0))  # Infer the␣
↪response noise

        # Sampling model
        with pyro.plate("data", 6):
            obs = pyro.sample("obs", dist.Normal(mu, sigma*sigma), obs=y)
        return mu

# Calculate returns and alphas using basic BNN with 1 day lag
def model_basic_bnn(X, y, X_train, y_train, X_val, y_val):
    # Convert data
    x_train_tensor = torch.tensor(X_train.values).float()
    y_train_tensor = torch.tensor(y_train.values).float()
    x_val_tensor = torch.tensor(X_val.values).float()
    y_val_tensor = torch.tensor(y_val.values).float()
    x_tensor = torch.tensor(X.values).float()
    y_tensor = torch.tensor(y.values).float()

    # Define the neural network model
    model = BasicBNN()

    # Set Pyro random seed
    pyro.set_rng_seed(COMMON_SEED)

    # Define Hamiltonian Monte Carlo (HMC) kernel
    # NUTS = "No-U-Turn Sampler" (https://arxiv.org/abs/1111.4246), gives HMC␣
↪an adaptive step size
```

```
    nuts_kernel = NUTS(model, jit_compile=True) # jit_compile=True is faster␣
 ↪but requires PyTorch 1.6+

    # Define MCMC sampler, get 50 posterior samples
    mcmc = MCMC(nuts_kernel, num_samples=30)

    # Run MCMC
    mcmc.run(x_train_tensor, y_train_tensor)

    # Make predictions
    predictive = Predictive(model=model, posterior_samples=mcmc.get_samples())
    preds = predictive(x_tensor)

    pred_means = preds['obs'].T.detach().numpy().mean(axis=2)
    pred_df = pd.DataFrame(pred_means.T)

    res = predictions_to_returns(pred_df, y)
    return predictive, res[0], res[1]

# Advanced BNN that has been parameter tuned.
# ReLU activation function, 5 day lag, hid_dim=50
class AdvancedBNN(PyroModule):
    def __init__(self, in_dim=30, out_dim=6, hid_dim=40, prior_scale=1.):
        super().__init__()

        self.activation = nn.ReLU()
        self.layer1 = PyroModule[nn.Linear](in_dim, hid_dim)  # Input to hidden␣
 ↪layer
        self.layer2 = PyroModule[nn.Linear](hid_dim, out_dim)  # Hidden to␣
 ↪output layer

        # Set layer parameters as random variables
        self.layer1.weight = PyroSample(dist.Normal(0., prior_scale).
 ↪expand([hid_dim, in_dim]).to_event(2))
        self.layer1.bias = PyroSample(dist.Normal(0., prior_scale).
 ↪expand([hid_dim]).to_event(1))
        self.layer2.weight = PyroSample(dist.Normal(0., prior_scale).
 ↪expand([out_dim, hid_dim]).to_event(2))
        self.layer2.bias = PyroSample(dist.Normal(0., prior_scale).
 ↪expand([out_dim]).to_event(1))

    def forward(self, x, y=None):
        x = self.activation(self.layer1(x))
        mu = self.layer2(x).squeeze()
        sigma = pyro.sample("sigma", dist.Gamma(0.5, 2.0))  # Infer the␣
 ↪response noise
```

```
        # Sampling model
        with pyro.plate("data", 6):
            obs = pyro.sample("obs", dist.Normal(mu, sigma*sigma), obs=y)
        return mu

# Calculate returns and alphas using basic BNN with 1 day lag
def model_advanced_bnn(X, y, X_train, y_train, X_val, y_val):
    # Convert data
    x_train_tensor = torch.tensor(X_train.values).float()
    y_train_tensor = torch.tensor(y_train.values).float()
    x_val_tensor = torch.tensor(X_val.values).float()
    x_val_tensor = torch.tensor(y_val.values).float()
    x_tensor = torch.tensor(X.values).float()
    y_tensor = torch.tensor(y.values).float()

    # Define the neural network model
    model = AdvancedBNN()

    # Set Pyro random seed
    pyro.set_rng_seed(COMMON_SEED)

    # Define Hamiltonian Monte Carlo (HMC) kernel
    # NUTS = "No-U-Turn Sampler" (https://arxiv.org/abs/1111.4246), gives HMC␣
 ↪an adaptive step size
    nuts_kernel = NUTS(model, jit_compile=True) # jit_compile=True is faster␣
 ↪but requires PyTorch 1.6+

    # Define MCMC sampler, get 50 posterior samples
    mcmc = MCMC(nuts_kernel, num_samples=30)

    # Run MCMC
    mcmc.run(x_train_tensor, y_train_tensor)

    # Make predictions
    predictive = Predictive(model=model, posterior_samples=mcmc.get_samples())
    preds = predictive(x_tensor)

    pred_means = preds['obs'].T.detach().numpy().mean(axis=2)
    pred_df = pd.DataFrame(pred_means.T)

    res = predictions_to_returns_with_short(pred_df, y)
    return predictive, res[0], res[1]

# Calculate returns and alphas using basic BNN with 1 day lag
def model_sharpe_bnn(X, y, X_train, y_train, X_val, y_val):
    # Convert data
```

```
    x_train_tensor = torch.tensor(X_train.values).float()
    y_train_tensor = torch.tensor(y_train.values).float()
    x_val_tensor = torch.tensor(X_val.values).float()
    x_val_tensor = torch.tensor(y_val.values).float()
    x_tensor = torch.tensor(X.values).float()
    y_tensor = torch.tensor(y.values).float()

    # Define the neural network model
    model = AdvancedBNN()

    # Set Pyro random seed
    pyro.set_rng_seed(COMMON_SEED)

    # Define Hamiltonian Monte Carlo (HMC) kernel
    # NUTS = "No-U-Turn Sampler" (https://arxiv.org/abs/1111.4246), gives HMC␣
 ↪an adaptive step size
    nuts_kernel = NUTS(model, jit_compile=True) # jit_compile=True is faster␣
 ↪but requires PyTorch 1.6+

    # Define MCMC sampler, get 50 posterior samples
    mcmc = MCMC(nuts_kernel, num_samples=30)

    # Run MCMC
    mcmc.run(x_train_tensor, y_train_tensor)

    # Make predictions
    predictive = Predictive(model=model, posterior_samples=mcmc.get_samples())
    preds = predictive(x_tensor)

    pred_means = preds['obs'].T.detach().numpy().mean(axis=2)
    pred_df = pd.DataFrame(pred_means.T)

    pred_stds = preds['obs'].T.detach().numpy().std(axis=2)
    std_df = pd.DataFrame(pred_stds.T)

    res = predictions_to_returns_max_sharpe(pred_df, std_df, y)
    return predictive, res[0], res[1]

# Calculate returns and alphas using basic BNN with 1 day lag
def model_sharpe_bnn_long_short(X, y, X_train, y_train, X_val, y_val):
    # Convert data
    x_train_tensor = torch.tensor(X_train.values).float()
    y_train_tensor = torch.tensor(y_train.values).float()
    x_val_tensor = torch.tensor(X_val.values).float()
    x_val_tensor = torch.tensor(y_val.values).float()
    x_tensor = torch.tensor(X.values).float()
    y_tensor = torch.tensor(y.values).float()
```

```
    # Define the neural network model
    model = AdvancedBNN()

    # Set Pyro random seed
    pyro.set_rng_seed(COMMON_SEED)

    # Define Hamiltonian Monte Carlo (HMC) kernel
    # NUTS = "No-U-Turn Sampler" (https://arxiv.org/abs/1111.4246), gives HMC␣
↪an adaptive step size
    nuts_kernel = NUTS(model, jit_compile=True) # jit_compile=True is faster␣
↪but requires PyTorch 1.6+

    # Define MCMC sampler, get 50 posterior samples
    mcmc = MCMC(nuts_kernel, num_samples=30)

    # Run MCMC
    mcmc.run(x_train_tensor, y_train_tensor)

    # Make predictions
    predictive = Predictive(model=model, posterior_samples=mcmc.get_samples())
    preds = predictive(x_tensor)

    pred_means = preds['obs'].T.detach().numpy().mean(axis=2)
    pred_df = pd.DataFrame(pred_means.T)

    pred_stds = preds['obs'].T.detach().numpy().std(axis=2)
    std_df = pd.DataFrame(pred_stds.T)

    res = predictions_to_returns_max_sharpe_long_short(pred_df, std_df, y)
    return predictive, res[0], res[1]
```

1.2 Establish the Benchmark

1.2.1 HW5 best feed forward model

[5]:
```
COMMON_SEED = 123
EVAL_CUTOFF = 20200101

# Calculate returns and alphas using the feed forward neural net with
# five day lagged input variables.
predictive0, strat_df0, return_vector0, model_OLS0 = fitting_returns_data(
    'ff6_factors_19630701_20230131.csv',
    io_day_5_lag,
    model_feed_forward,
    "5 Day Lag Feed Forward",
    seed = COMMON_SEED,
```

```
    log = True);
```

```
Epoch 1/50
356/356             0s 541us/step -
loss: 0.4316 - val_loss: 0.3749
Epoch 2/50
356/356             0s 360us/step -
loss: 0.3777 - val_loss: 0.3701
Epoch 3/50
356/356             0s 370us/step -
loss: 0.3705 - val_loss: 0.3691
Epoch 4/50
356/356             0s 376us/step -
loss: 0.3657 - val_loss: 0.3684
Epoch 5/50
356/356             0s 371us/step -
loss: 0.3619 - val_loss: 0.3688
Epoch 6/50
356/356             0s 388us/step -
loss: 0.3590 - val_loss: 0.3689
Epoch 7/50
356/356             0s 371us/step -
loss: 0.3557 - val_loss: 0.3700
Epoch 8/50
356/356             0s 376us/step -
loss: 0.3529 - val_loss: 0.3707
Epoch 9/50
356/356             0s 372us/step -
loss: 0.3499 - val_loss: 0.3716
Epoch 10/50
356/356             0s 376us/step -
loss: 0.3471 - val_loss: 0.3727
Epoch 11/50
356/356             0s 389us/step -
loss: 0.3444 - val_loss: 0.3739
Epoch 12/50
356/356             0s 374us/step -
loss: 0.3425 - val_loss: 0.3741
Epoch 13/50
356/356             0s 386us/step -
loss: 0.3399 - val_loss: 0.3749
Epoch 14/50
356/356             0s 385us/step -
loss: 0.3377 - val_loss: 0.3760
Epoch 15/50
356/356             0s 383us/step -
loss: 0.3363 - val_loss: 0.3776
```

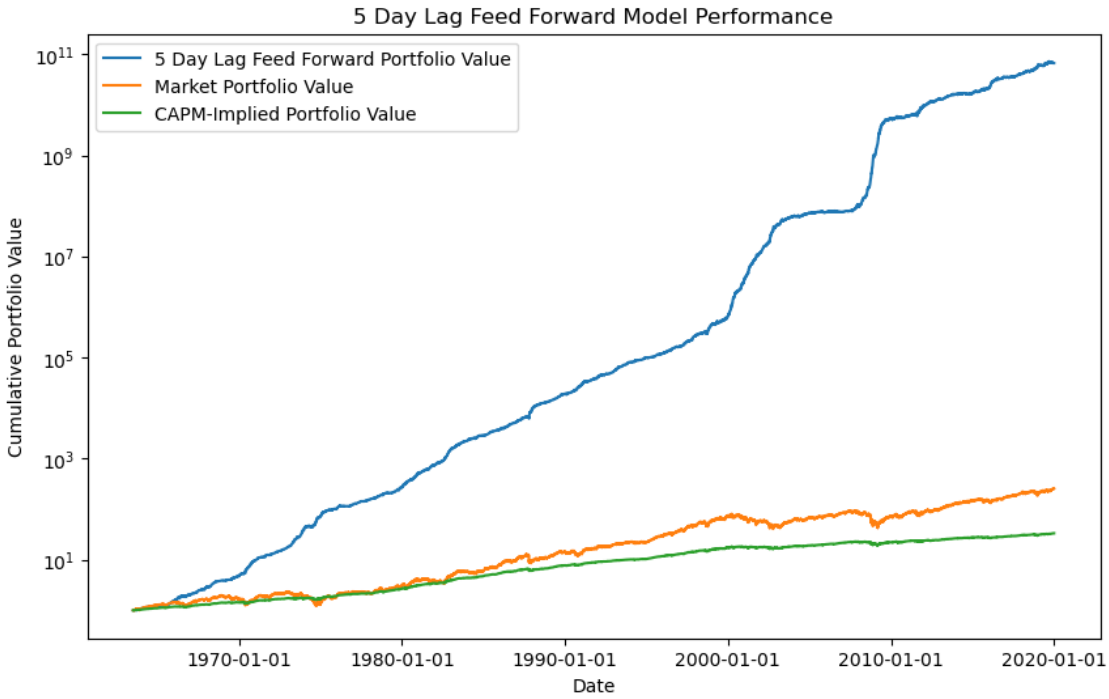
```
Epoch 16/50
356/356             0s 600us/step -
loss: 0.3342 - val_loss: 0.3792
Epoch 17/50
356/356             0s 402us/step -
loss: 0.3320 - val_loss: 0.3806
Epoch 18/50
356/356             0s 393us/step -
loss: 0.3306 - val_loss: 0.3811
Epoch 19/50
356/356             0s 384us/step -
loss: 0.3292 - val_loss: 0.3832
Epoch 20/50
356/356             0s 607us/step -
loss: 0.3276 - val_loss: 0.3842
Epoch 21/50
356/356             0s 397us/step -
loss: 0.3263 - val_loss: 0.3849
Epoch 22/50
356/356             0s 401us/step -
loss: 0.3245 - val_loss: 0.3866
Epoch 23/50
356/356             0s 414us/step -
loss: 0.3236 - val_loss: 0.3875
Epoch 24/50
356/356             0s 408us/step -
loss: 0.3220 - val_loss: 0.3889
Epoch 25/50
356/356             0s 404us/step -
loss: 0.3205 - val_loss: 0.3908
Epoch 26/50
356/356             0s 372us/step -
loss: 0.3195 - val_loss: 0.3917
Epoch 27/50
356/356             0s 395us/step -
loss: 0.3181 - val_loss: 0.3931
Epoch 28/50
356/356             0s 409us/step -
loss: 0.3167 - val_loss: 0.3943
Epoch 29/50
356/356             0s 403us/step -
loss: 0.3154 - val_loss: 0.3965
Epoch 30/50
356/356             0s 385us/step -
loss: 0.3142 - val_loss: 0.3971
Epoch 31/50
356/356             0s 379us/step -
loss: 0.3129 - val_loss: 0.3987
```

```
Epoch 32/50
356/356             0s 393us/step -
loss: 0.3121 - val_loss: 0.4022
Epoch 33/50
356/356             0s 375us/step -
loss: 0.3108 - val_loss: 0.4023
Epoch 34/50
356/356             0s 370us/step -
loss: 0.3101 - val_loss: 0.4051
Epoch 35/50
356/356             0s 372us/step -
loss: 0.3084 - val_loss: 0.4063
Epoch 36/50
356/356             0s 373us/step -
loss: 0.3076 - val_loss: 0.4075
Epoch 37/50
356/356             0s 369us/step -
loss: 0.3065 - val_loss: 0.4076
Epoch 38/50
356/356             0s 424us/step -
loss: 0.3059 - val_loss: 0.4095
Epoch 39/50
356/356             0s 427us/step -
loss: 0.3050 - val_loss: 0.4111
Epoch 40/50
356/356             0s 397us/step -
loss: 0.3042 - val_loss: 0.4126
Epoch 41/50
356/356             0s 386us/step -
loss: 0.3032 - val_loss: 0.4141
Epoch 42/50
356/356             0s 379us/step -
loss: 0.3020 - val_loss: 0.4161
Epoch 43/50
356/356             0s 354us/step -
loss: 0.3010 - val_loss: 0.4181
Epoch 44/50
356/356             0s 370us/step -
loss: 0.3011 - val_loss: 0.4208
Epoch 45/50
356/356             0s 356us/step -
loss: 0.2999 - val_loss: 0.4223
Epoch 46/50
356/356             0s 372us/step -
loss: 0.2992 - val_loss: 0.4263
Epoch 47/50
356/356             0s 369us/step -
loss: 0.2981 - val_loss: 0.4283
```

```
Epoch 48/50
356/356             0s 368us/step -
loss: 0.2978 - val_loss: 0.4284
Epoch 49/50
356/356             0s 367us/step -
loss: 0.2968 - val_loss: 0.4324
Epoch 50/50
356/356             0s 360us/step -
loss: 0.2962 - val_loss: 0.4335
445/445             0s 231us/step
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.149
Model:                            OLS   Adj. R-squared:                  0.149
Method:                 Least Squares   F-statistic:                     415.9
Date:                Fri, 07 Jun 2024   Prob (F-statistic):               0.00
Time:                        00:31:48   Log-Likelihood:                -14480.
No. Observations:               14218   AIC:                         2.897e+04
Df Residuals:                   14211   BIC:                         2.903e+04
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1603      0.006     28.395      0.000       0.149       0.171
Mkt-RF         0.2728      0.006     42.503      0.000       0.260       0.285
SMB            0.0924      0.012      8.031      0.000       0.070       0.115
HML            0.1780      0.014     12.591      0.000       0.150       0.206
RMW            0.0593      0.016      3.602      0.000       0.027       0.092
CMA           -0.0332      0.020     -1.656      0.098      -0.072       0.006
MOM            0.2352      0.009     27.337      0.000       0.218       0.252
==============================================================================
Omnibus:                     6933.988   Durbin-Watson:                   1.718
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           148699.674
Skew:                           1.853   Prob(JB):                         0.00
Kurtosis:                      18.404   Cond. No.                         4.00
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.


5 Day Lag Feed Forward daily returns:
Mean = 0.178%
Volatility = 0.726%
Sharpe Ratio = 0.245
```



1.3 Predicting Factor returns using a Bayesian Neural Network

1.3.1 Train a basic BNN and analyze performance

[6]:
```
predictive1, strat_df1, return_vector1, model_OLS1 = fitting_returns_data(
    'ff6_factors_19630701_20230131.csv',
    io_day_1_lag,
    model_basic_bnn,
    "Basic Factor BNN",
    seed = COMMON_SEED,
    log = True)
```

```
Sample: 100%|  | 60/60 [03:12,  3.20s/it, step size=1.00e-02, acc. prob=0.298]
```

OLS Regression Results

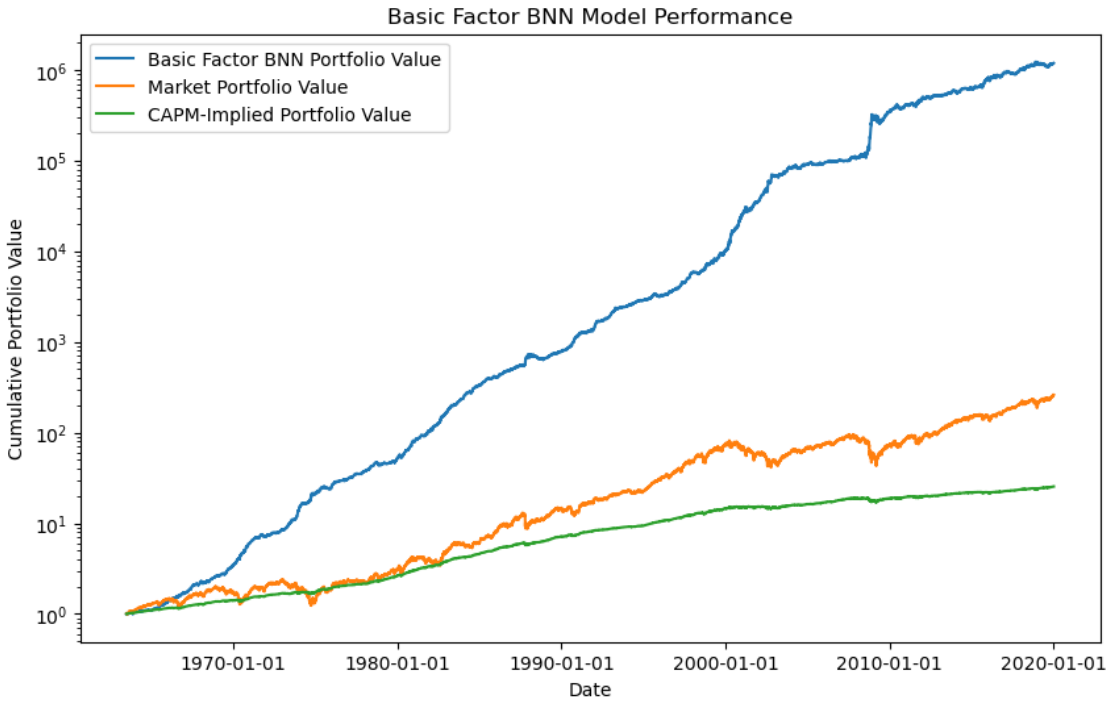
Dep. Variable:	y	R-squared:	0.103
Model:	OLS	Adj. R-squared:	0.103
Method:	Least Squares	F-statistic:	272.2
Date:	Fri, 07 Jun 2024	Prob (F-statistic):	0.00
Time:	00:35:01	Log-Likelihood:	-13808.
No. Observations:	14222	AIC:	2.763e+04
Df Residuals:	14215	BIC:	2.768e+04
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0816	0.005	15.155	0.000	0.071	0.092
Mkt-RF	0.1904	0.006	31.104	0.000	0.178	0.202
SMB	0.0988	0.011	9.005	0.000	0.077	0.120
HML	0.1498	0.013	11.114	0.000	0.123	0.176
RMW	0.2062	0.016	13.142	0.000	0.175	0.237
CMA	0.1744	0.019	9.127	0.000	0.137	0.212
MOM	0.2100	0.008	25.598	0.000	0.194	0.226

Omnibus:	5306.150	Durbin-Watson:	1.905
Prob(Omnibus):	0.000	Jarque-Bera (JB):	305569.130
Skew:	0.992	Prob(JB):	0.00
Kurtosis:	25.621	Cond. No.	4.00

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Basic Factor BNN daily returns:
Mean = 0.101%
Volatility = 0.675%
Sharpe Ratio = 0.149



1.4 Advanced BNN

Add the ability to short factors. After experimenting, change to ReLU activation, use 5 day lag, increase size of hidden layer.

[7]:
```
predictive2, strat_df2, return_vector2, model_OLS2 = fitting_returns_data(
    'ff6_factors_19630701_20230131.csv',
    io_day_5_lag,
    model_advanced_bnn,
    "Advanced Factor BNN",
    seed = COMMON_SEED,
    log = True)
```

```
Sample: 100%|  | 60/60 [14:37, 14.63s/it, step size=3.26e-03, acc. prob=0.571]
```

OLS Regression Results

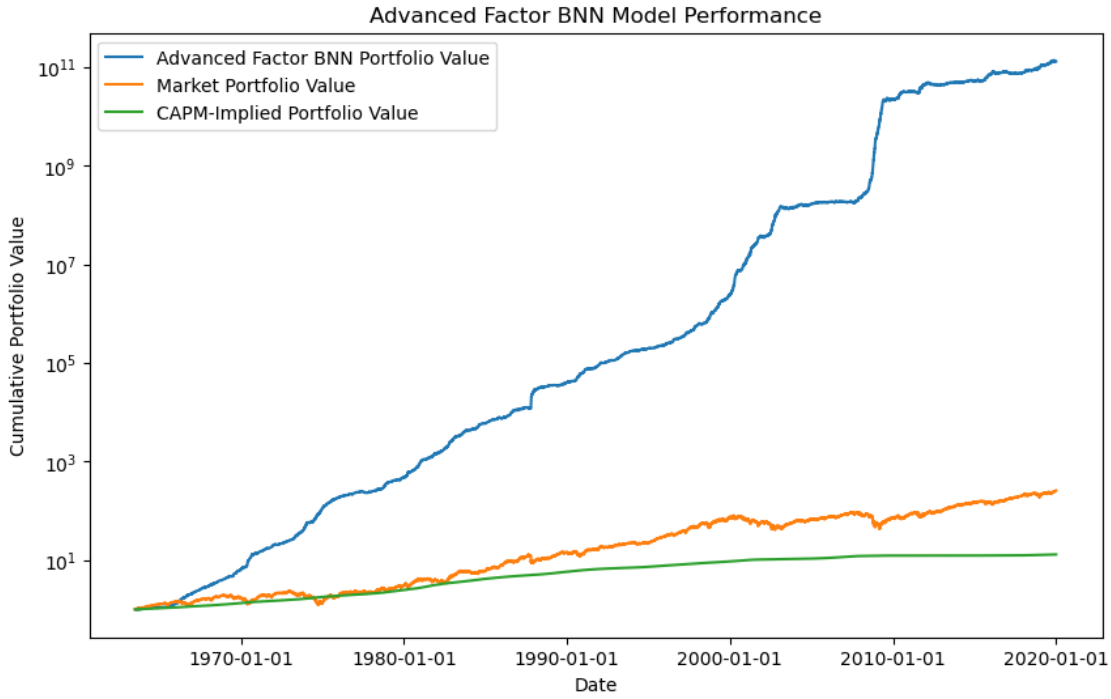
Dep. Variable:	y	R-squared:	0.011
Model:	OLS	Adj. R-squared:	0.011
Method:	Least Squares	F-statistic:	26.16
Date:	Fri, 07 Jun 2024	Prob (F-statistic):	3.98e-31
Time:	00:49:40	Log-Likelihood:	-17294.
No. Observations:	14218	AIC:	3.460e+04
Df Residuals:	14211	BIC:	3.466e+04
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.1856	0.007	26.970	0.000	0.172	0.199
Mkt-RF	0.0016	0.008	0.206	0.837	-0.014	0.017
SMB	-0.0554	0.014	-3.948	0.000	-0.083	-0.028
HML	0.0201	0.017	1.165	0.244	-0.014	0.054
RMW	0.0578	0.020	2.881	0.004	0.018	0.097
CMA	0.0330	0.024	1.350	0.177	-0.015	0.081
MOM	-0.1045	0.010	-9.962	0.000	-0.125	-0.084

Omnibus:	10002.612	Durbin-Watson:	1.657
Prob(Omnibus):	0.000	Jarque-Bera (JB):	578137.073
Skew:	2.770	Prob(JB):	0.00
Kurtosis:	33.744	Cond. No.	4.00

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
Advanced Factor BNN daily returns:
Mean = 0.184%
Volatility = 0.821%
Sharpe Ratio = 0.224
```



1.4.1 Max/min Sharpe Ratio BNN

[15]:
```
predictive3, strat_df3, return_vector3, model_OLS3 = fitting_returns_data(
    'ff6_factors_19630701_20230131.csv',
    io_day_5_lag,
    model_sharpe_bnn,
    "Sharpe Ratio Factor BNN",
    seed = COMMON_SEED,
    log = True)
```

```
Sample: 100%|  | 60/60 [07:56,  7.95s/it, step size=3.26e-03, acc. prob=0.571]
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.013
Model:	OLS	Adj. R-squared:	0.012
Method:	Least Squares	F-statistic:	30.70
Date:	Fri, 07 Jun 2024	Prob (F-statistic):	7.56e-37

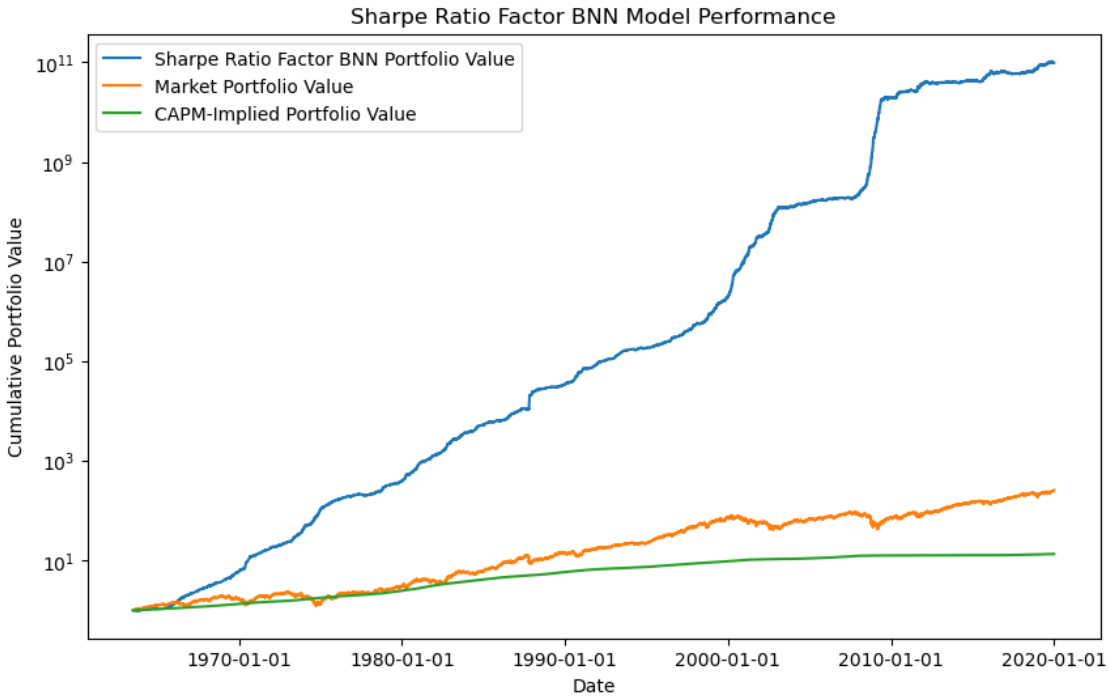
Time:	01:20:08	Log-Likelihood:	-17006.
No. Observations:	14218	AIC:	3.403e+04
Df Residuals:	14211	BIC:	3.408e+04
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.1831	0.007	27.164	0.000	0.170	0.196
Mkt-RF	0.0127	0.008	1.658	0.097	-0.002	0.028
SMB	-0.0595	0.014	-4.328	0.000	-0.086	-0.033
HML	0.0363	0.017	2.149	0.032	0.003	0.069
RMW	0.0517	0.020	2.633	0.008	0.013	0.090
CMA	0.0218	0.024	0.912	0.362	-0.025	0.069
MOM	-0.1066	0.010	-10.375	0.000	-0.127	-0.086

Omnibus:	10009.485	Durbin-Watson:	1.657
Prob(Omnibus):	0.000	Jarque-Bera (JB):	625311.765
Skew:	2.747	Prob(JB):	0.00
Kurtosis:	35.021	Cond. No.	4.00

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Sharpe Ratio Factor BNN daily returns:
Mean = 0.181%
Volatility = 0.805%
Sharpe Ratio = 0.225



1.4.2 Long-short Sharpe Ratio BNN

[9]:
```
predictive4, strat_df4, return_vector4, model_OLS3 = fitting_returns_data(
    'ff6_factors_19630701_20230131.csv',
    io_day_5_lag,
    model_sharpe_bnn_long_short,
    "Sharpe Ratio Factor BNN",
    seed = COMMON_SEED,
    log = True)
```

```
Sample: 100%|  | 60/60 [09:19,  9.32s/it, step size=3.26e-03, acc. prob=0.571]
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.013
Model:	OLS	Adj. R-squared:	0.012
Method:	Least Squares	F-statistic:	30.70
Date:	Fri, 07 Jun 2024	Prob (F-statistic):	7.56e-37
Time:	01:11:10	Log-Likelihood:	-17006.
No. Observations:	14218	AIC:	3.403e+04
Df Residuals:	14211	BIC:	3.408e+04
Df Model:	6		
Covariance Type:	nonrobust		

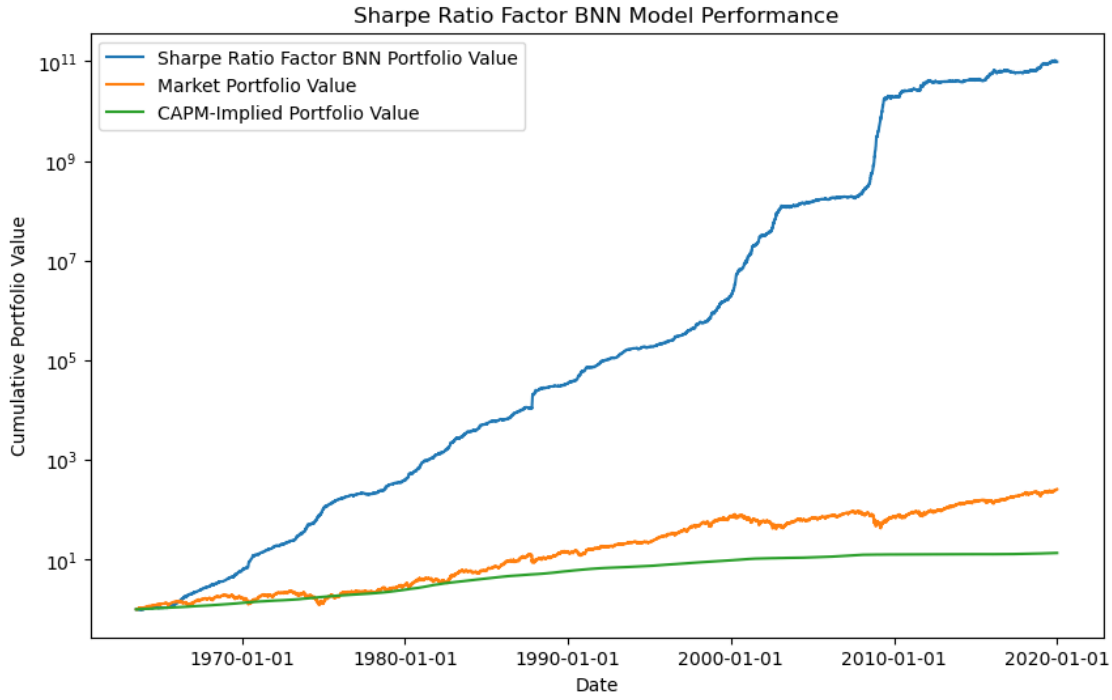
	coef	std err	t	P>|t|	[0.025	0.975]

const	0.1831	0.007	27.164	0.000	0.170	0.196
Mkt-RF	0.0127	0.008	1.658	0.097	-0.002	0.028
SMB	-0.0595	0.014	-4.328	0.000	-0.086	-0.033
HML	0.0363	0.017	2.149	0.032	0.003	0.069
RMW	0.0517	0.020	2.633	0.008	0.013	0.090
CMA	0.0218	0.024	0.912	0.362	-0.025	0.069
MOM	-0.1066	0.010	-10.375	0.000	-0.127	-0.086

Omnibus:	10009.485	Durbin-Watson:	1.657
Prob(Omnibus):	0.000	Jarque-Bera (JB):	625311.765
Skew:	2.747	Prob(JB):	0.00
Kurtosis:	35.021	Cond. No.	4.00

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Sharpe Ratio Factor BNN daily returns:
Mean = 0.181%
Volatility = 0.805%
Sharpe Ratio = 0.225



1.5 Out of sample predictions

[10]:
```
df1 = pd.read_csv('FF5_daily.CSV')
df1 = df1[df1['date'] >= EVAL_CUTOFF]

df2 = pd.read_csv('MOM_daily.CSV')
df2 = df2[df2['date'] >= EVAL_CUTOFF]

eval_df = pd.merge(df1, df2, how='inner', on=['date'])
assert len(eval_df) == len(df1)
eval_df
```

[10]:

	date	Mkt-RF	SMB	HML	RMW	CMA	RF	Mom
0	20200102	0.86	-0.97	-0.34	0.24	-0.22	0.006	0.82
1	20200103	-0.67	0.30	0.00	-0.14	-0.10	0.006	0.03
2	20200106	0.36	-0.20	-0.55	-0.17	-0.26	0.006	-0.69
3	20200107	-0.19	-0.03	-0.25	-0.12	-0.25	0.006	0.01
4	20200108	0.47	-0.17	-0.64	-0.19	-0.17	0.006	0.92
…	…	…	…	…	…	…	…	…
1084	20240424	-0.01	-0.29	0.27	0.23	-0.03	0.021	-0.61
1085	20240425	-0.47	-0.39	-0.18	0.40	-0.08	0.021	0.03
1086	20240426	1.04	0.21	-1.06	0.01	-0.44	0.021	1.15
1087	20240429	0.34	0.34	-0.49	0.23	-0.12	0.021	-1.00
1088	20240430	-1.67	-0.27	-0.26	-0.60	0.63	0.021	-0.55

[1089 rows x 8 columns]

[26]:
```
# Basic Factor BNN predictions
predict_out_of_sample(predictive1, eval_df, io_day_1_lag, "Basic Factor BNN")
```

OLS Regression Results

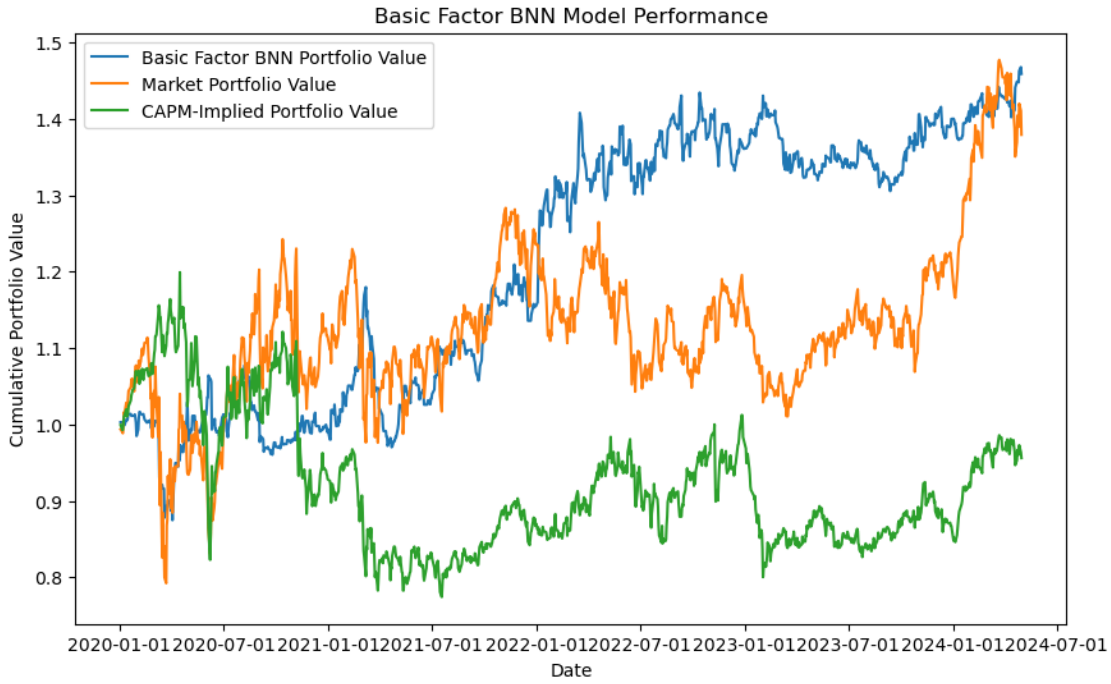
Dep. Variable:	y	R-squared:	0.198
Model:	OLS	Adj. R-squared:	0.193
Method:	Least Squares	F-statistic:	44.45
Date:	Fri, 07 Jun 2024	Prob (F-statistic):	1.01e-48
Time:	01:21:36	Log-Likelihood:	-1337.4
No. Observations:	1088	AIC:	2689.
Df Residuals:	1081	BIC:	2724.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0286	0.034	0.854	0.393	-0.037	0.094
Mkt-RF	0.1371	0.019	7.173	0.000	0.100	0.175
SMB	0.1802	0.038	4.791	0.000	0.106	0.254
HML	0.1086	0.033	3.274	0.001	0.044	0.174

```
RMW             0.2821      0.047      5.988      0.000       0.190       0.375
CMA             0.2053      0.061      3.383      0.001       0.086       0.324
RF             -0.5682      2.872     -0.198      0.843      -6.203       5.067
==============================================================================
Omnibus:                      128.694   Durbin-Watson:                   2.054
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1128.635
Skew:                          -0.102   Prob(JB):                    8.32e-246
Kurtosis:                       7.985   Cond. No.                        170.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.


Basic Factor BNN daily returns:
Mean = 0.039%
Volatility = 0.924%
Sharpe Ratio = 0.042
```



[12]:
```
# Advanced Factor BNN Predictions
predict_out_of_sample(predictive2, eval_df, io_day_5_lag, "Advanced Factor BNN")
```

```
                            OLS Regression Results
==============================================================================
```

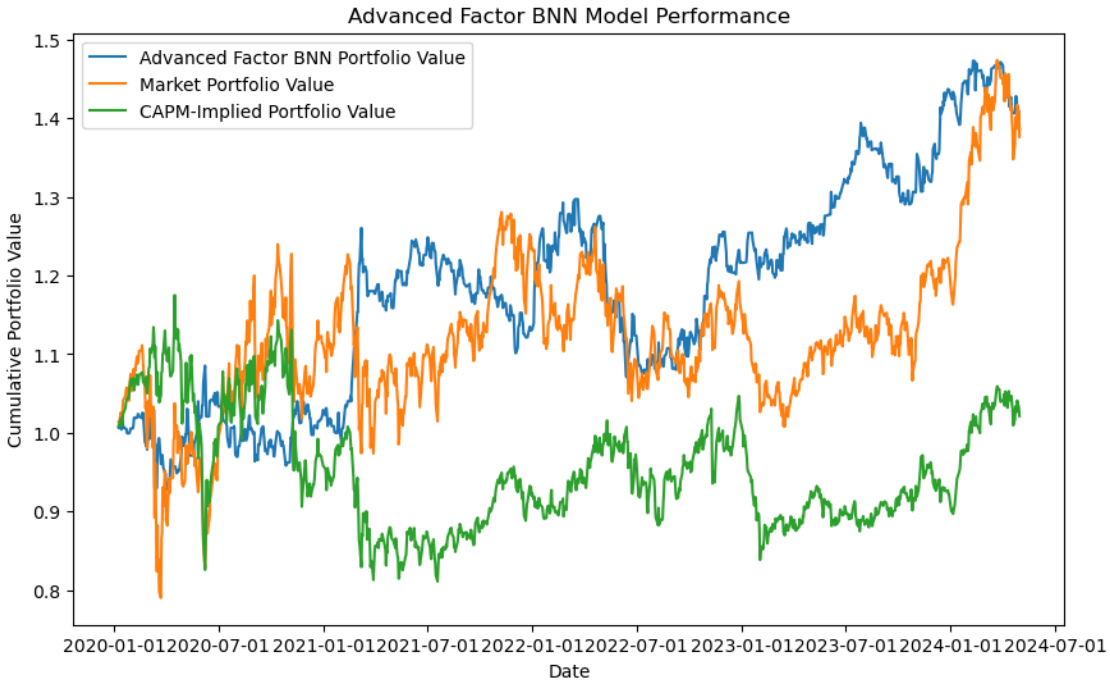
Dep. Variable:	y	R-squared:	0.233
Model:	OLS	Adj. R-squared:	0.229
Method:	Least Squares	F-statistic:	54.57
Date:	Fri, 07 Jun 2024	Prob (F-statistic):	6.73e-59
Time:	01:11:10	Log-Likelihood:	-1408.4
No. Observations:	1084	AIC:	2831.
Df Residuals:	1077	BIC:	2866.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0018	0.036	-0.049	0.961	-0.072	0.069
Mkt-RF	0.2720	0.020	13.271	0.000	0.232	0.312
SMB	0.1252	0.040	3.103	0.002	0.046	0.204
HML	0.1325	0.036	3.723	0.000	0.063	0.202
RMW	0.0682	0.051	1.350	0.177	-0.031	0.167
CMA	0.2838	0.065	4.359	0.000	0.156	0.412
RF	2.6447	3.080	0.859	0.391	-3.399	8.689

Omnibus:	135.185	Durbin-Watson:	2.025
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1056.714
Skew:	-0.264	Prob(JB):	3.45e-230
Kurtosis:	7.808	Cond. No.	170.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Advanced Factor BNN daily returns:
Mean = 0.035%
Volatility = 1.014%
Sharpe Ratio = 0.035



[23]:
```
# Advanced Factor BNN Predictions for maximum/minimum sharpe ratio
predict_out_of_sample(predictive3, eval_df, io_day_5_lag, "Sharpe Ratio Factor␣
↪BNN")
```

OLS Regression Results

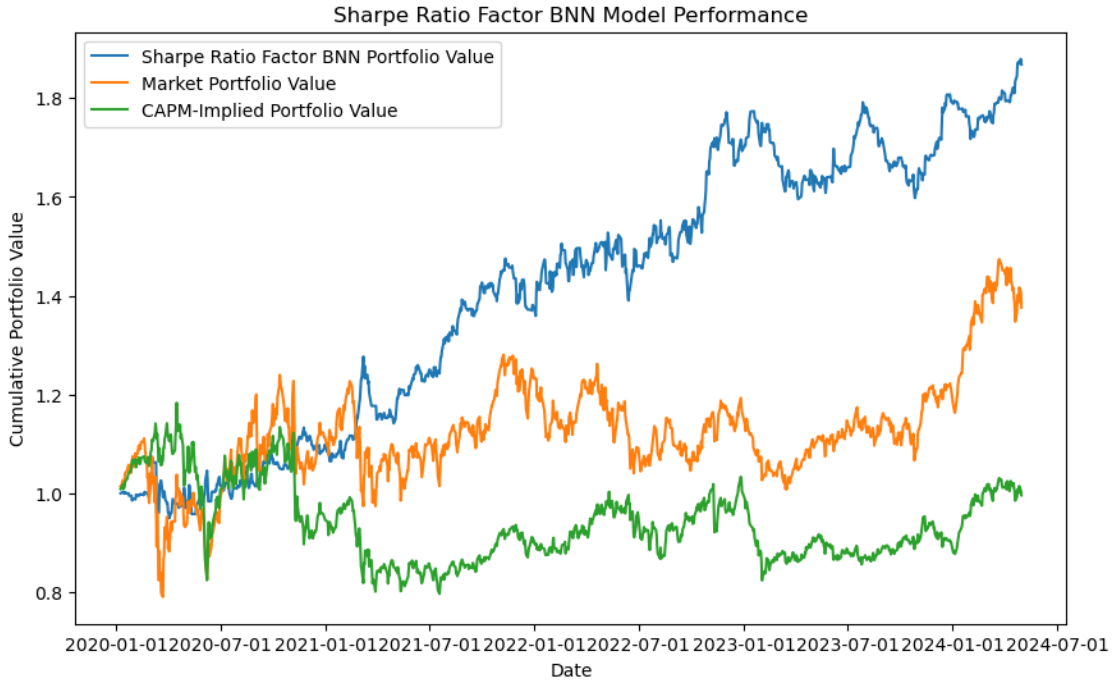
Dep. Variable:	y	R-squared:	0.198
Model:	OLS	Adj. R-squared:	0.193
Method:	Least Squares	F-statistic:	44.24
Date:	Fri, 07 Jun 2024	Prob (F-statistic):	1.74e-48
Time:	01:21:14	Log-Likelihood:	-1384.4
No. Observations:	1084	AIC:	2783.
Df Residuals:	1077	BIC:	2818.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0466	0.035	1.322	0.187	-0.023	0.116
Mkt-RF	0.2217	0.020	11.057	0.000	0.182	0.261
SMB	0.1421	0.039	3.601	0.000	0.065	0.220
HML	0.1302	0.035	3.739	0.000	0.062	0.198
RMW	0.1467	0.049	2.967	0.003	0.050	0.244
CMA	0.1529	0.064	2.402	0.016	0.028	0.278
RF	0.0579	3.013	0.019	0.985	-5.854	5.970

```
==============================================================================
Omnibus:                      179.185   Durbin-Watson:                   1.983
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2133.130
Skew:                          -0.343   Prob(JB):                         0.00
Kurtosis:                       9.838   Cond. No.                         170.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.


Sharpe Ratio Factor BNN daily returns:
Mean = 0.062%
Volatility = 0.969%
Sharpe Ratio = 0.064
```



[14]:
```
# Long-short sharpe ratio portfolio
predict_out_of_sample(predictive4, eval_df, io_day_5_lag, "Long-short Sharpe␣
↪Ratio Factor BNN")
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.228
Model:                            OLS   Adj. R-squared:                  0.224
```

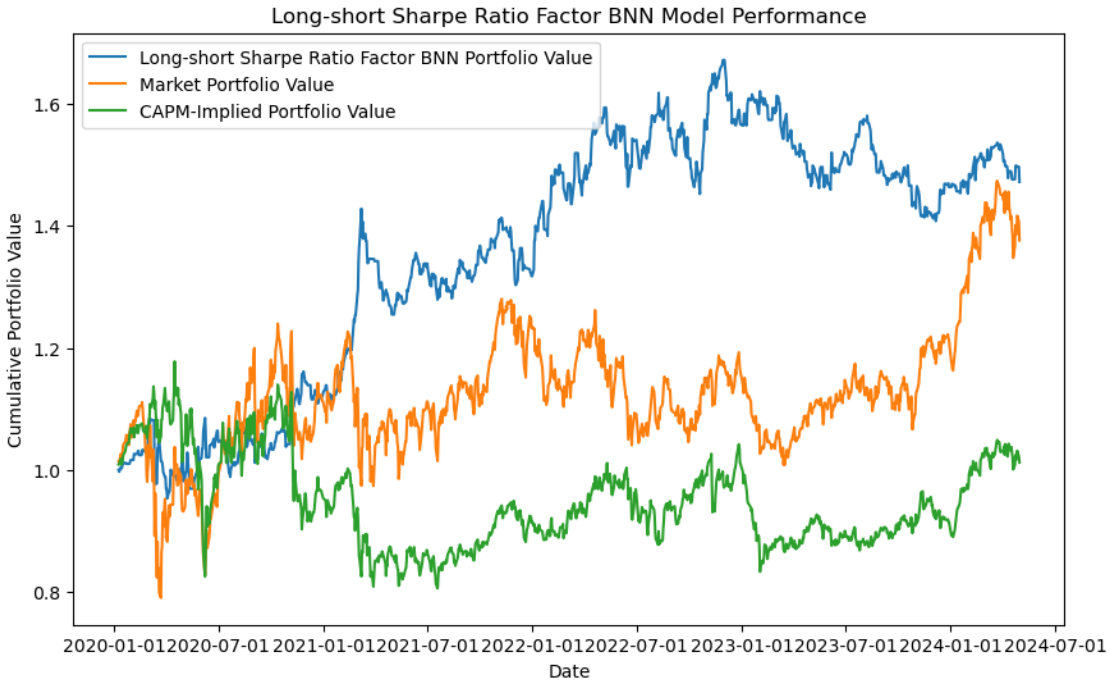
Method:	Least Squares	F-statistic:	53.03
Date:	Fri, 07 Jun 2024	Prob (F-statistic):	2.24e-57
Time:	01:11:11	Log-Likelihood:	-1383.7
No. Observations:	1084	AIC:	2781.
Df Residuals:	1077	BIC:	2816.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0467	0.035	1.327	0.185	-0.022	0.116
Mkt-RF	0.2540	0.020	12.676	0.000	0.215	0.293
SMB	0.1421	0.039	3.604	0.000	0.065	0.220
HML	0.1289	0.035	3.706	0.000	0.061	0.197
RMW	0.1056	0.049	2.138	0.033	0.009	0.203
CMA	0.2292	0.064	3.602	0.000	0.104	0.354
RF	-2.9217	3.011	-0.970	0.332	-8.830	2.986

Omnibus:	147.964	Durbin-Watson:	1.959
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1376.871
Skew:	-0.259	Prob(JB):	1.04e-299
Kurtosis:	8.497	Cond. No.	170.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Long-short Sharpe Ratio Factor BNN daily returns:
Mean = 0.041%
Volatility = 0.988%
Sharpe Ratio = 0.041


Long-short Sharpe Ratio Factor BNN Model Performance
Long-short Sharpe Ratio Factor BNN Portfolio Value
Market Portfolio Value
CAPM-Implied Portfolio Value
Cumulative Portfolio Value
1.6
1.4
1.2
1.0
0.8
2020-01-01
2020-07-01
2021-01-01
2021-07-01
2022-01-01
2022-07-01
2023-01-01
2023-07-01
2024-01-01
2024-07-01
Date

[ ]: