Final Report of the Stock Project

Rui Li

June 15th 2023

Data Preparation

Data Perparation

- WRDS
- https://feng-cityuhk.github.io/EquityCharacteristics/

generate characteristics to predict stock return

Avoid data leakage

Predicting the next month's stock return using data in current months.

- ```
  data['year_month'] = pd.to_datetime(data['date']).dt.to_period('M')
  data['ret_fut'] = data.groupby('permno')['ret'].shift(-1)
  data = data.dropna(axis=0,subset=['ret_fut']) #drop column with na
  ```
- Data preprocessing within each rolling window
  ```

Architecture

```python
for test_date in test_dates:
  # training set
  X_ptrain,y_ptrain = data[data.date < test_date]
  #test set
  X_test,y_test = datap[data.date == test date]

  #Hyperparameter tuning
  if test_dates.index == 0 or test_dates.index%12 == 11:

    #training set for fine tuning
    X_train,y_train = data[data.date < test_date - 2 month]
    # 2 months span validation set
    X_valid,y_valid = data[(data.date >= test_date-2month)
&(data.date < test_date)]

                    """
    Use optuna to do Hyperparameter fine tuning
                    """

  #train model
  model = Model(**best_params)
  model.fit(X_ptrain,y_ptrain)
  y_pred =  model.predict(X_test)
```

Rolling windows:

- A monthly expand training set
- Initialize with time span of 32 months
- Data perprocessing within each rolling
- Fine tuning per year
- Creating validation set with the last two months each year

Data :

- Data for training model: (X_ptrain , y_ptrain)
- Data for fine tuning: (X_train , y_train) , (X_valid , y_valid)
- Data for prediction and calculate R^2 : (X_test , y_test)

Model:

- OLS , lasso
- (No justifications for fine-tuning)
- RF , Xgboost , Lightgbm , pca + Lightgbm
- NN3

```
for test_date in test_dates:
  # training set
  X_ptrain,y_ptrain = data[data.date < test_date]
  #test set
  X_test,y_test = datap[data.date == test date]

  #drop categorical variables
  s = (data.dtypes == 'int64')
  object_cols = list(s[s].index)
  data = data.drop(object_cols, axis=1)

  #standardlize by column

  scaler = StandardScaler()

  for col in object_cols:
      scaler.fit(X_test[col].values.reshape(-1,1))

      X_test[col] =
      scaler.transform(X_test[col].values.reshape(-1,1))

      scaler.fit(X_ptrain[col].values.reshape(-1,1))

      X_ptrain[col] =
      scaler.transform(X_ptrain[col].values.reshape(-1,1))
```

Data Preprocessing within rollowing

- standardlize by column within each rollowing window
- drop categorical variables for OLS, Lasso and NN3
- drop future return to aviod data leakage

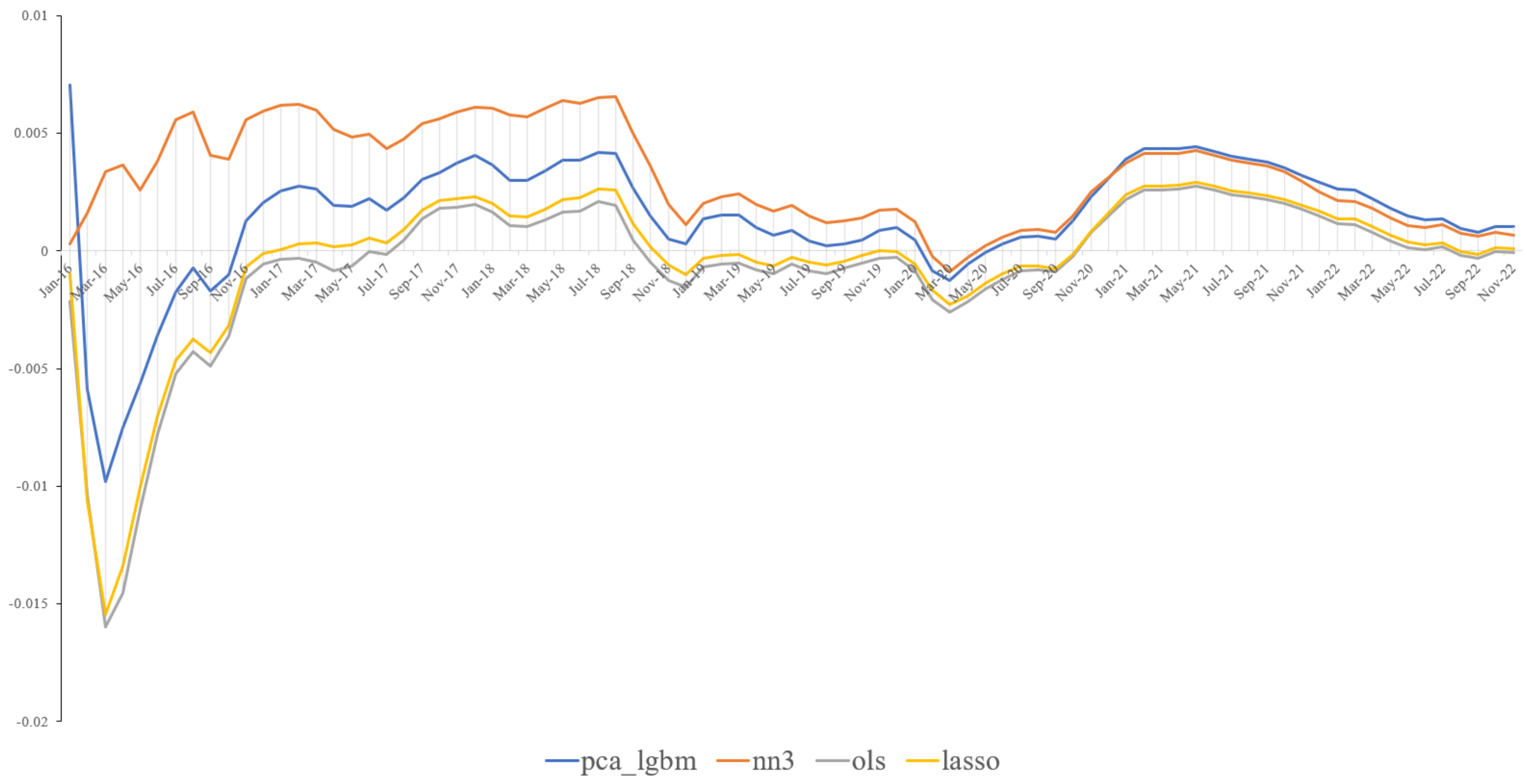
Method and Result

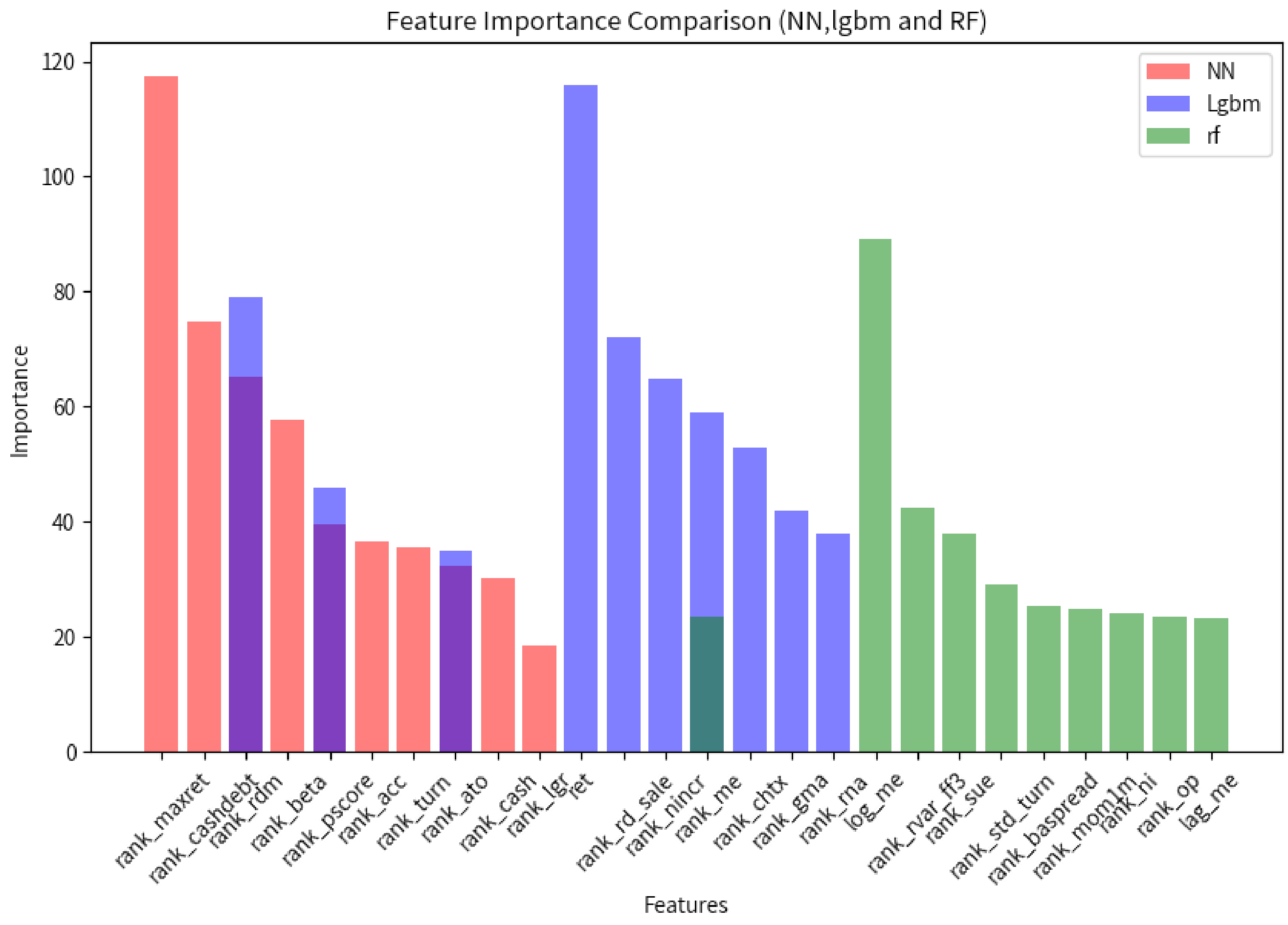
Models and Hyperparameters

Random Forest	GBRT	XGBoost	LightGBM	NN3
max_depth = [1, 6] n_estimators = 300 max_features = {3, 5, 10, 20, 30, 50,...}	max_depth = [1, 2] learning_rate = {0.001,0.1} n_estimators = [1,1000]			l1 = (1e-5 , 1e-3) learning_rate = {1e-3, 1e-2} batch_size = 10000 epochs = 100 Patience = 5 optimizer=keras.optimizers.Adam
max_depth = [1, 6] n_estimators = 300 max_features = {3, 5, 10, 20, 30, 50}		max_depth = [1, 2] n_estimators = [1,1000] learning_rate = [0.001,0.1]	max_depth = [1, 2] learning_rate = [0.001,0.1] n_estimators = [50,500] num_leaves = [10,100] min_child_samples = [1,50] subsample = [0.1,1] colsample_bytree = [0.1,1]	l1 = (1e-5 , 1e-3) learning_rate = {1e-3, 1e-2} batch_size = 10000 epochs = 100 Patience = 5 optimizer=keras.optimizers.Adam activation = {relu, sigmoid} num_neurons[i] = (8, 256)

Remark: Hyperparameters in the first row are chosen by *Shihao Gu (2020)* and hyperparameters in the second row are hyperparameters used in my program.

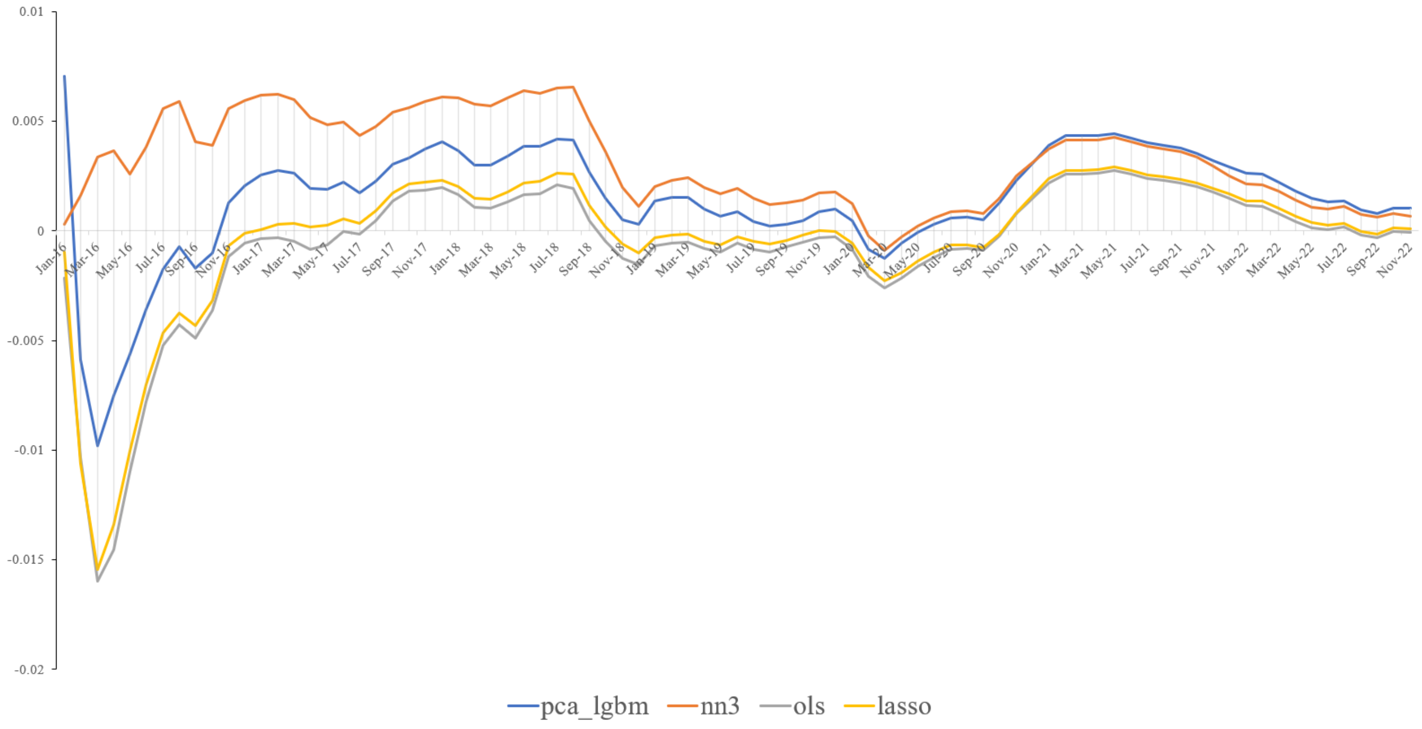
R-squared


0.01
0.005
0
-0.005
-0.01
-0.015
-0.02
Jan-16
Mar-16
May-16
Jul-16
Sep-16
Nov-16
Jan-17
Mar-17
May-17
Jul-17
Sep-17
Nov-17
Jan-18
Mar-18
May-18
Jul-18
Sep-18
Nov-18
Jan-19
Mar-19
May-19
Jul-19
Sep-19
Nov-19
Jan-20
Mar-20
May-20
Jul-20
Sep-20
Nov-20
Jan-21
Mar-21
May-21
Jul-21
Sep-21
Nov-21
Jan-22
Mar-22
May-22
Jul-22
Sep-22
Nov-22
pca_lgbm
nn3
ols
lasso



Feature Importance

- Here we only display the top 10 important features for each model.



- Best Model: NN3
- Conicide with *Gu(2020)*
- Negative R concern

Negative R concern

Out of sample R squared:

$$R_{OOS}^2 = 1 - \frac{\sum\left(ret_{OOS} - \widehat{ret}_{OOS}\right)^2}{\sum ret_{OOS}^2} = 1 - \frac{MSE\ Loss_{OOS}}{\sum ret_{OOS}^2}$$

Objective Function for Model in Training:

$$MSE\ Loss = \sum\left(ret_{IS} - \widehat{ret}_{IS}\right)^2$$

- We try to minimize out of sample $MSE\ Loss$ through minimizing insample $MSE\ Loss$ .It's also worth to note that OLS estimator is the BLUE estimator.However, when we fixed all $\widehat{ret}_{OOS} = 0$, R_{OOS}^2 equals 0, which means that in some cases,the BLUE estimator performs worse than fixing every predicted value to 0.
- Simultaneous variation in R_{OOS}^2 performance within different model might implies our data has low ability in capture the vloatility of stock market.
- From my perspective, negative R^2 might caused by some weaknesses of data.

Weaknesses of Data

Noise and Multi-colinearity

I use PCA to denoise and eliminate multi-colinearity exists between the variables.

```
#dimension-reduction method
pca = PCA(n_components='mle',svd_solver='full')
pca.fit(X_ptrain)
X_ptrain_pca = pca.transform(X_ptrain)
X_test_pca = pca.transform(X_test)
```

Remark: Another reason is to fasten the speed of training by reducing the number of variables.

Short-term trend

Add an indicator function to capture short term trend (rise/fall) per stock:

```
data['trend'] = data.apply(lambda x: 1 if data['ret'] - data['ret_l1'] > 0 else 0 if
data['ret'] - data['ret_l1'] == 0 else -1, axis=1) #avoid data leakage
```

Do contribute to R^2_{OOS} performance when trend in stock market is stable.

Weaknesses of Data

Volatility of the stock market

- Our data show poor ability in capture the volatility of the stock market due to leak of characteristics. *Gu(2020)* uses 94 characteristics and we are only allowed to construct 63 of them.
- No β Factor in our data. Hard to capture stock's volatility in relation to the market.

Other Attempt

Training Neural Network with Pytorch using GPU

Activate GPU device:

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

Data Preparation:

```
# 1. convert data into tensor format
X_train = torch.tensor(X_train,dtype = torch.float)
y_train = torch.tensor(train_y,dtype = torch.float)

#2.tansport data to device
X_train = X_train.to(device)
y_train = y_train.to(device)
```

Training Neural Network with Pytorch using GPU

Model Preparation:

```
#transport model to device
net = AssetPricingNN(**best_params).to(device)
```

Detach data to numpy:

```
#transport data back to cpu
predict_y = predict_y.detach().cpu().numpy()
```

```python
class AssetPricingNN(nn.Module):
    def __init__(self, input_size, hidden_size_0,hidden_size_1,hidden_size_2,
output_size):
        super().__init__()
        self.emb0 = nn.Embedding(40000,64)
        self.emb1 = nn.Embedding(40000,64)
        self.emb2 = nn.Embedding(4000,16)
        self.emb3 = nn.Embedding(4000,16)
        self.fc1 = nn.Linear(input_size, hidden_size_0)
        self.fc2 = nn.Linear(hidden_size_0, hidden_size_1)
        self.fc3 = nn.Linear(hidden_size_1, hidden_size_2)
        self.predict = nn.Linear(hidden_size_2,output_size)
        self.emb =  nn.Embedding(3,2)

    def forward(self,x,cat0,cat1,cat2,cat3):
        cat0 = cat0.long()
        cat1 = cat1.long()
        cat2 = cat2.long()
        cat3 = cat3.long()

        cat0 = self.emb0(cat0)
        cat1 = self.emb1(cat1)
        cat2 = self.emb2(cat2)
        cat3 = self.emb3(cat3)

        x = torch.cat((x, cat0,cat1,cat2,cat3), dim = 1)

        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = self.predict(x)
        return x
```

Embedding

- In comparision to model that delete categorical variables directly, NN3 with embedding layers show no significiant improvement in R^2_{OOS} performance and run slower.

```python
#Model architecture
class FactorAE(nn.Module):

    def __init__(self):
        super(FactorAE,self).__init__()
        #encoder architecture
        self.factor_output_layer = nn.Linear(58,3)
        self.batch1 = nn.BatchNorm2d(1,eps = 1e-5,affine = True)
        self.batch2 = nn.BatchNorm2d(1,eps = 1e-5,affine = True)
        self.beta_layer1 = nn.Linear(1,32)
        self.beta_layer2 = nn.Linear(32,16)
        self.beta_layer3 = nn.Linear(16,3)
        self.relu = nn.ReLU()

    def forward(self,data,y_return):

        OLS = []
        for i in range(X.shape[0]):
            x = X[i]
            x = x.reshape(1,-1)
            try:
                a = np.linalg.inv(np.dot(x.T,x))
            except:
                 a = np.linalg.pinv(np.dot(x.T,x))
            b = np.dot(x.T,y[i])
            OLS.append(np.dot(a,b))

        OLS_tensor = OLS_tensor.squeeze(2)

        factor_output = self.factor_output_layer(OLS_tensor)
        #beta part
        beta = self.relu(self.batch1(self.beta_layer1(data.unsqueeze(1))))
        beta = self.relu(self.batch2(self.beta_layer2(beta)))
        beta = self.beta_layer3(beta).squeeze(1)
        reconstuct_return =
        torch.matmul(beta,factor_output.unsqueeze(2)).squeeze(2)

        return reconstuct_return
```

AutoEncoder

Pervious Model:

- use data before next month to generate parameters for model
- use data in current month to make prediction

AutoEncoder:

- use data before next month to generate parameters for model
- use all historical data to make prediction

Motivation: To capture market volaitility by model itself using historical data