11/5/2025

MET CS 767 Assignment 3: CNN's

Brendan Torok

1. Modification of code to attempt improvement (2 page max)

1.1 Description of changes and reason they *could reasonably be* an improvement

The first change I made was to add softmax to the output layer to normalize the outputs to make them comparable on the same scale; this slightly improved the model performance. I then tested adding more convoluted and dense hidden layers with different neuron amounts in each layer to allow the model to learn more complex features. The model then began overfitting on training data. To manage overfitting, I experimented with adding dropouts in both the convoluted and dense layers. This change decreased overfitting improving generalizability of the model. To further decrease overfitting, I experimented with l2 regularization in the dense and convoluted layers. This further increased accuracy on the test dataset. My model was then slightly underfitting, so I tuned the values for the dropout and regularization to stop underfitting. Afterwards I added batch normalization which stabilizes the activations to use higher learning rates in the optimizer and the model performance improved. I then tried different learning rates with a learning rate schedule and increased the epochs of the model with early stopping. A learning rate schedule can allow the model to change the learning rate when the model no longer is learning as fast, and more epochs allows the model to learn for longer. These both improved the model performance, but not as much as expected.

To further improve I attempted to use SGD momentum, as it may have increased performance with a more complex architecture, but performance did not improve. Since Nadam can outperform Adam, I tested it next but again the performance did not improve. I then tested the GeLu activation function as it has a more complex shape and found that this increased performance. Afterwards I tested some hyperparameter changes and model performance stagnated. AI then suggested data augmentation which introduces slight changes to the training data to increase variance and make the model more generalizable. The data augmentation used introduced random flips, random crops, random rotations, and random contrast. While using data augmentation, the model initially was underfitting very heavily due to all the dropout and regularization, so I relaxed the dropout and regularization and then replaced flatten with GlobalAveragePooling2D in order to reduce the model size and make feature maps to be meaningful. This achieved a good balance between over and underfitting and improved the model performance without adjusting the original training dataset to decrease the original model performance. I also attempted to try a random search for parameters to tune the parameters, the overall training time ended up being ~10hrs but the results did not outperform my manual tuning.

1.2 Comparison of the actual result with the original output, explained

Model	Loss	Accuracy
Original	0.9997	68.29%
Final	0.6447	81.81%

*Performance diagram can be seen in appendix 1

Overall, the accuracy on the test data of my final model increased by 13.52% and the loss decreased by 0.355. This is a substantial increase in the performance of the model. A large part of this increase was driven by the architectural changes to the model as well as adding data augmentation to training. The substantial improvement with the addition of data augmentation is because creates new synthetic data samples by transforming the original data, increasing the dataset size and introducing variation which exposes the model to more examples to learn from. This makes the model less likely to overfit and therefore more generalizable. The code of the final model can be seen in appendix 2.

1.3 URL of your Colab code

https://colab.research.google.com/drive/16DEG283vvsM4Xbi0hdmMT0xqlOJby7Dj?usp=sharing

Link to ChatGPT conversation: https://chatgpt.com/share/6918fc00-b42c-8006-a92f-61a3c6c80558

2. *Your* CNN Application (3 pg max)

2.1 Give 2-4 requirements for a highly capable, unique application you'll implement

This application will utilize a hybrid convoluted neural network and multilayer percepton model that can classify stocks based on their projected price increase relative to a benchmark in the next trading day based on the current company fundamentals and the past 30 days of daily data. This model will provide a list of the top 5 companies that are most likely to have the highest percentage increase over the baseline as well as the top 5 companies that are most likely to underperform compared to a benchmark. Users are required to provide history of the past 30 trading days, as well as current company fundamentals for stocks in the S&P1500 for a classification to be generated. Additionally, if a user provides only one stock with the required data, the model will predict its classification. The inverse list can also be provided if the user would like to know which stocks are most likely to underperform if the user is interested in identifying which stocks to take up a short position in.

2.2 Uniqueness of Your Application

This model is unique in that it uses convoluted neural networks which are typically used for 2d image processing neural networks to make predictions on the stock market. Since the convoluted kernel slides across the time series with a fixed size, it is best at identifying local interactions that occur over a limited time window. These limited time windows may include something like a 3-day momentum swing, or a short-term price reversal which makes this model good for predicting short term stock movements. Additionally, since classification is made against a benchmark this model can provide both a projected increase through classification, but also a projected decrease. This allows the model to be used for taking both long and short positions in the stock market. The training of this model is unique as it uses real time stock market data to make predictions. Updates could be made to this model to take an input of streaming data and make predictions on more granular changes in the market, taking the predictions from days to minutes, allowing for trading algorithms to then make fast trades for quick profit.

2.3 Sample I/O

All input/output actual results can be seen in appendix 3

1) This model uses predicted performance to classify stocks as overperform, neutral, or overperform but it is unclear how what fraction of the stocks with each rating actually have the proper frequency. A calibration curve can be used to determine the probability reliability. We can see that when the mean p_over for a bin greater than 0.75, all of the stocks actually outperform the benchmark.

Input: predicted probability, actual performance label

Output: actual overperformance frequency for each bin center

2) Since the previous version of this application provided the most likely to outperform stocks, I added functionality for this model to predict the stocks that are most likely to underperform the benchmark in the next trading day. The probability of being in the outperform class is used to get the margin that this is greater than other classes. This is used to get the confidence that a stock will underperform. This confidence is then sorted to return the stocks most likely to underperform.

Input: p_under and margin_under_vs_next for each stock

Output: output is the confidence a stock will underperform and the top 5 stocks most likely to underperform using this underperform confidence metric along with some key metrics.

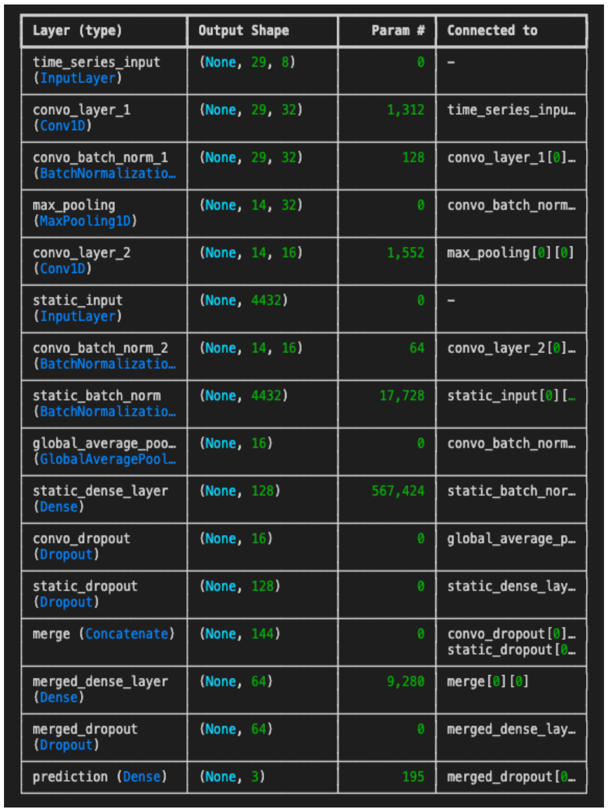
3) Since our dataset already provides analyst ratings of the stocks, we can see if our 3 label version goes against analyst recommendations, with assigning a high (>0.6) probability to overperform to stocks that are rated as ‘hold’, ‘none’, ‘underperform’, or ‘none’.

Input: analyst recommendation keys and p_over

Output: crosstab of recommendation key and p_over

2.4 The CNN Architecture

Model summary diagram with named layers:



```
Layer (type)                 Output Shape        Param #  Connected to
time_series_input            (None, 29, 8)             0  -
(InputLayer)
convo_layer_1                (None, 29, 32)        1,312  time_series_inpu…
(Conv1D)
convo_batch_norm_1           (None, 29, 32)          128  convo_layer_1[0]…
(BatchNormalizatio…
max_pooling                  (None, 14, 32)            0  convo_batch_norm…
(MaxPooling1D)
convo_layer_2                (None, 14, 16)        1,552  max_pooling[0][0]
(Conv1D)
static_input                 (None, 4432)              0  -
(InputLayer)
convo_batch_norm_2           (None, 14, 16)           64  convo_layer_2[0]…
(BatchNormalizatio…
static_batch_norm            (None, 4432)         17,728  static_input[0][…
(BatchNormalizatio…
global_average_poo…          (None, 16)                0  convo_batch_norm…
(GlobalAveragePool…
static_dense_layer           (None, 128)         567,424  static_batch_nor…
(Dense)
convo_dropout                (None, 16)                0  global_average_p…
(Dropout)
static_dropout               (None, 128)               0  static_dense_lay…
(Dropout)
merge (Concatenate)          (None, 144)               0  convo_dropout[0]…
                                                          static_dropout[0…
merged_dense_layer           (None, 64)            9,280  merge[0][0]
(Dense)
merged_dropout               (None, 64)                0  merged_dense_lay…
(Dropout)
prediction (Dense)           (None, 3)               195  merged_dropout[0…
```

ASCII diagram included in Appendix 4

2.5 Uniqueness of Your Architecture

This model is a hybrid CNN + MLP model that combines both time series and static data in two separate branches that are merged. The preprocessing of the static and time series data is done separately. There is a multi-channel input of time series data which learns short-term temporal patterns in stock price/volume behavior. The static branch learns how the fundamentals of the stock shift the odds of performance. These two models are then merged and two dense layers then learn a joint representation of the data to make a final prediction for which class the stock will be in. This is a unique way of using both time series and static data to predict future performance of stocks compared to a benchmark. Additionally this model is uniquely tuned to work on 30 days of data which is not particularly large for this application. Despite this constraint, the model achieved an accuracy of 54.49% on the 3 way classification problem with a loss of 1.0004. The macro F1 is 0.5409 and weighted F1 is 0.5404. Additional weighting is added to the neutral class to increase performance, but maximum neutral precision achieved was 0.42.

All performance metrics in appendix 5

Example of an excellent starting prompt with context and rules document is provided in appendix 6

2.6 Key code

Provide snippets of the essential core code of your implementation.

```
# time series branch
ts_input = tf.keras.Input(shape=(n_timesteps, n_channels), name="time_series_input")

# define the time series branch of the convoluted NN
# follow similar structure as the neural network trained in part 1
# train static branch
# add max pooling
ts_branch = layers.Conv1D(filters=32, kernel_size=5, activation='relu',
padding='causal',
                          name='convo_layer_1')(ts_input)
ts_branch = layers.BatchNormalization(name='convo_batch_norm_1')(ts_branch)
ts_branch = layers.MaxPooling1D(pool_size=2, name='max_pooling')(ts_branch)
ts_branch = layers.Conv1D(filters=16, kernel_size=3, activation='relu',
                          kernel_regularizer=tf.keras.regularizers.l2(1e-3),
padding='causal',
                          name='convo_layer_2')(ts_branch)
ts_branch = layers.BatchNormalization(name='convo_batch_norm_2')(ts_branch)
ts_branch = layers.GlobalAveragePooling1D(name='global_average_pooling')(ts_branch)
ts_branch = layers.Dropout(0.15, name='convo_dropout')(ts_branch)

# now train static branch
static_input = tf.keras.Input(shape=(n_static,), name='static_input')
# added batch normalization inside raw static input
static_branch = layers.BatchNormalization(name='static_batch_norm')(static_input)
# reduced dense layer neurons to 128
static_branch = layers.Dense(128, activation='relu',
                             kernel_regularizer=tf.keras.regularizers.l2(1e-4),
                             name='static_dense_layer')(static_branch)
static_branch = layers.Dropout(0.2, name='static_dropout')(static_branch)
# merge branches
merged = layers.Concatenate(name='merge')([ts_branch, static_branch])
# add dense layer after merging
merged = layers.Dense(64, activation='relu', name='merged_dense_layer')(merged)
merged = layers.Dropout(0.3, name='merged_dropout')(merged)
output = layers.Dense(3, activation='softmax', name='prediction')(merged)
```

Example of an excellent starting prompt with context and rules document is provided in the appendix 5

2.7 URL of your Colab code

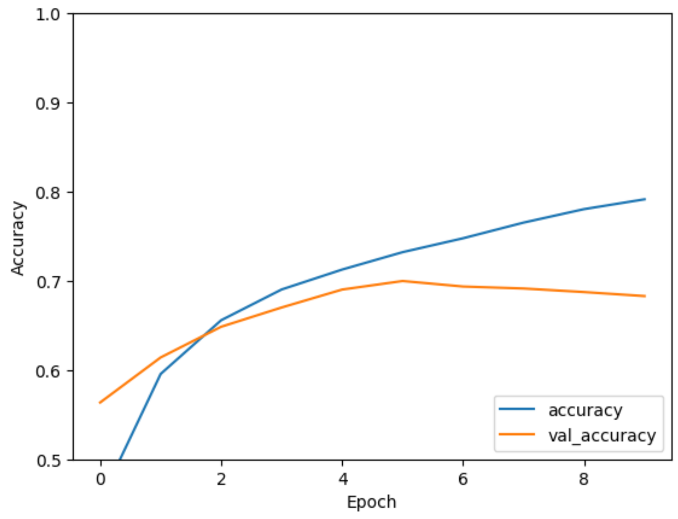
Link to github: https://github.com/btorok-bu/METCS767_hw3/blob/main/cs767_btorok_hw3.ipynb

Link to updating data source chat: https://chatgpt.com/share/69195518-70dc-8006-b6f8-22b3e58d4035

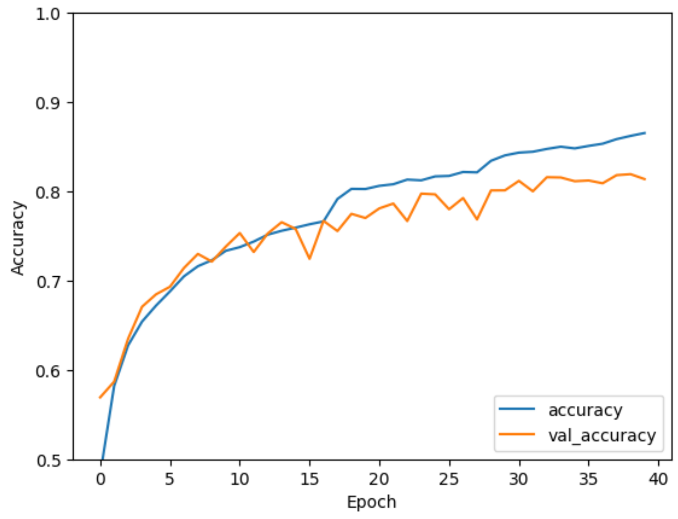
Link to training neural network and data wrangling chat: https://chatgpt.com/share/691bfcc7-aed4-8006-a820-0a373cc4886e

Appendix 1

First model diagram:



Final model diagram:



Appendix 2

Code of final model:

```
# add more complex data augmentation
data_augmentation = tf.keras.Sequential([layers.RandomCrop(32,32),
                                         layers.RandomFlip('horizontal'),
                                         layers.RandomRotation(0.1),
                                         layers.RandomContrast(0.1)])

# add batch normalization between layers
model_21 = models.Sequential()
# add data augmentation to model
model_21.add(data_augmentation)
model_21.add(layers.Conv2D(32, (3, 3), use_bias=False, input_shape=(32,
32, 3)))
model_21.add(layers.BatchNormalization())
model_21.add(layers.Activation('relu'))
model_21.add(layers.MaxPooling2D((2, 2)))
model_21.add(layers.Conv2D(64, (3, 3), use_bias=False))
model_21.add(layers.BatchNormalization())
model_21.add(layers.Activation('relu'))
model_21.add(layers.MaxPooling2D((2, 2)))
# added additional conv layer to make model capable of learning more
complex features
model_21.add(layers.Conv2D(128, (3, 3), use_bias=False))
model_21.add(layers.BatchNormalization())
model_21.add(layers.Activation('relu'))
model_21.add(layers.SpatialDropout2D(0.05))
# Increased final Conv2D layer to 256 to make the net enter higher
dimensional space for more complex features
# add l2 regularizer, tune to lower regularization
model_21.add(layers.Conv2D(256, (3, 3), use_bias=False,
            kernel_regularizer=tf.keras.regularizers.l2(5e-5)))
model_21.add(layers.BatchNormalization())
model_21.add(layers.Activation('relu'))
# replace flatten with GlobalAveragePooling2D
model_21.add(layers.GlobalAveragePooling2D())
model_21.add(layers.Dropout(0.1))
model_21.add(layers.Dense(64, activation='relu',
                          kernel_regularizer=tf.keras.regularizers.l2(5e-
5)))
model_21.add(layers.Dense(10, activation='softmax'))

# add learning rate schedule
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)
```

```
model_21.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

lr_schedule = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5, # reduce LR by half
    patience=3, # wait 3 epochs with no improvement
    min_lr=1e-6
)

# add early stopping
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=7, # epochs with no improvement before stopping
    restore_best_weights=True
)

# increase epochs to 40
history = model_21.fit(train_images, train_labels, epochs=40,
                    validation_data=(test_images, test_labels),
                      callbacks=[early_stopping, lr_schedule])
```

Appendix 3

Input 1:

```
predicted over performance probability
[0.27556458 0.28320086 0.20126888 0.14986669 0.5686222  0.15832336
 0.17724665 0.19045573 0.6673262  0.49587107]
actual value = over perform
[0 1 0 0 0 1 0 0 1 0]
```

Output 1:

```
Bin center p_over≈0.06 -> actual over freq=0.08
Bin center p_over≈0.16 -> actual over freq=0.17
Bin center p_over≈0.24 -> actual over freq=0.24
Bin center p_over≈0.34 -> actual over freq=0.36
Bin center p_over≈0.44 -> actual over freq=0.55
Bin center p_over≈0.55 -> actual over freq=0.70
Bin center p_over≈0.65 -> actual over freq=0.75
Bin center p_over≈0.75 -> actual over freq=1.00
Bin center p_over≈0.84 -> actual over freq=1.00
```

Input 2:

```
    p_under  margin_under_vs_next
0  0.073268             -0.412959
```

```
1  0.163022             -0.556540
2  0.067744             -0.556890
3  0.160065             -0.568843
4  0.449732             -0.003156
```

Output 2:

```
     symbol  under_confidence recommendationKey  currentPrice  forwardPE
1427   THRY          0.966578               buy          5.88   7.000000
1318   PLAY          0.960635               buy         14.15   3.803763
346     NVR          0.920910              none       7271.89  14.037044
1284    NWL          0.910765               buy          3.42   4.441559
693    IRDM          0.903632              none         16.54  17.229168
```

Input 3:

```
recommendations
0     buy
1    hold
2     buy
3     buy
4     buy

probability of overperforming
0    0.486227
1    0.117416
2    0.307621
3    0.728908
4    0.452888
```

Output 3:

```
Model says "overperform"  False  True
Analyst recommendation
buy                         679   152
hold                        239    32
none                        175    30
sell                          1     0
strong_buy                  140    44
underperform                  6     3
```

Appendix 4

Time-series input: time_series_input (batch, 29 timesteps, 8 channels)

```
  └─ Conv1D (convo_layer_1, 32 filters, kernel=5, padding='causal')
      └─ BatchNorm (convo_batch_norm_1)
      └─ MaxPooling1D (max_pooling, pool_size=2)
      └─ Conv1D (convo_layer_2, 16 filters, kernel=3, padding='causal', L2=1e-3)
          └─ BatchNorm (convo_batch_norm_2)
```

```
    └─ GlobalAveragePooling1D (global_average_pooling)
    └─ Dropout (convo_dropout, rate=0.15)
              \
               \
                ──▶ Concatenate (merge)
               /
Static input: static_input (batch, n_static)
  └─ BatchNorm (static_batch_norm)
  └─ Dense (static_dense_layer, units=128, relu, L2=1e-4)
    └─ Dropout (static_dropout, rate=0.2)


After merge:
  └─ Dense (merged_dense_layer, units=64, relu)
    └─ Dropout (merged_dropout, rate=0.3)
    └─ Dense (prediction, units=3, softmax)
```

Appendix 5

```
10/10 — 1s — 55ms/step — acc: 0.5349 — loss: 1.0004 — prec_neutral: 0.6809 —
prec_over: 0.6600 — prec_under: 0.7692 — rec_neutral: 0.3122 — rec_over: 0.1719 —
rec_under: 0.1688 — top2_acc: 0.8405
loss: 1.0004
compile_metrics: 0.5349

Macro F1: 0.5409
Weighted F1: 0.5404
              precision    recall  f1-score   support

       under       0.60      0.52      0.55        93
     neutral       0.42      0.59      0.49       104
        over       0.68      0.50      0.58       104

    accuracy                           0.53       301
   macro avg       0.57      0.53      0.54       301
weighted avg       0.57      0.53      0.54       301

[[48 40  5]
 [24 61 19]
 [ 8 44 52]]
```

Appendix 6

Starting prompt for part 2:

I have an existing script that was used to train a simple neural network on stock market data to predict whether a stock will outperform, be neutral, or underperform relative to a benchmark but it used anchor dates from the stock market alongside current company fundamental information. I would like to update this to use a convoluted neural network and instead of using anchor dates I would like to use trading history from the past 30 days and current company fundamentals.

1) what changes do I need to make to my current code in order to correctly get the benchmark performance for each day given the structure of my sp1500_company_with_history_wide.csv and benchmark_history.csv files?

2) When making the suggestions please make the changes to create new features and generate the z scores to be done within loops instead of explicitly labeling every new column as there will be 30 different days to compute for and mistakes are more likely to happen.

Please refer to the ai_rules_par2.txt and benchmark_history documents I have uploaded before answering.

```
# First create new label based on stock performance relative to other stocks and benchmark

# Use the direct path to the file in the environment

input_file = 'sp1500_company_with_history_wide.csv'

benchmark_input_file = 'benchmark_history.csv'

df_sp500 = pd.read_csv(benchmark_input_file)

df_raw = pd.read_csv(input_file)

benchmark_df = df_sp500[['anchor', 'close']]

benchmark_close = benchmark_df.set_index('anchor')['close'].to_dict()

# use log return - measurement of investment performance by ratio of final price to initial price

def log_return(final, initial):

return np.log(final / initial)

# define the benchmark log returns for the S&P500 in the same windows

benchmark_returns = {

'r_4y_2y': log_return(benchmark_close['2y'], benchmark_close['4y']),
'r_2y_1y': log_return(benchmark_close['1y'], benchmark_close['2y']),

'r_1y_6m': log_return(benchmark_close['6m'], benchmark_close['1y']),

'r_6m_3m': log_return(benchmark_close['3m'], benchmark_close['6m']),

'r_3m_1m': log_return(benchmark_close['1m'], benchmark_close['3m']),

'r_1m_1d': log_return(benchmark_close['1d'], benchmark_close['1m'])

}

# get returns for each stock in the same windows, subtract benchmark return to get relative
performance

# to benchmark

r_4y_2y = log_return(df_raw.get('2y_close'), df_raw.get('4y_close')) -
benchmark_returns['r_4y_2y']

r_2y_1y = log_return(df_raw.get('1y_close'), df_raw.get('2y_close')) -
benchmark_returns['r_2y_1y']

r_1y_6m = log_return(df_raw.get('6m_close'), df_raw.get('1y_close')) -
benchmark_returns['r_1y_6m']

r_6m_3m = log_return(df_raw.get('3m_close'), df_raw.get('6m_close')) -
benchmark_returns['r_6m_3m']

r_3m_1m = log_return(df_raw.get('1m_close'), df_raw.get('3m_close')) -
benchmark_returns['r_3m_1m']

r_1m_1d = log_return(df_raw.get('currentPrice'), df_raw.get('1m_close')) -
benchmark_returns['r_1m_1d']

# calculate z score for performance relative to others in each window

def zscore(s: pd.Series) -> pd.Series:

mu = s.mean(skipna=True)

sd = s.std(ddof=0, skipna=True)

return (s - mu) / sd

# compute z score for all stocks
```

```
z_4y_2y = zscore(r_4y_2y)
z_2y_1y = zscore(r_2y_1y)
z_1y_6m = zscore(r_1y_6m)
z_6m_3m = zscore(r_6m_3m)
z_3m_1m = zscore(r_3m_1m)
z_1m_1d = zscore(r_1m_1d)
# now create weights dict, more weight for more recent windows apart from 1 month window
# gives momentum score, but doesn't break momentum if stock market is in a brief down cycle
weights ={
'z_4y_2y': 0.05,
'z_2y_1y': 0.10,
'z_1y_6m': 0.15,
'z_6m_3m': 0.20,
'z_3m_1m': 0.30,
'z_1m_1d': 0.20
}
# create a dataframe of the z scores
z_df = pd.DataFrame({
'z_4y_2y': z_4y_2y,
'z_2y_1y': z_2y_1y,
'z_1y_6m': z_1y_6m,
'z_6m_3m': z_6m_3m,
'z_3m_1m': z_3m_1m,
'z_1m_1d': z_1m_1d
})
# multiply each z score by respective weights
for col, w in weights.items():
z_df[col] = z_df[col] * w
# tracks how much weight to include for each row
# boolean mask - true for non-NaN cells -> then convert to float and multiply by weights
# this gives an actual numeric weight if cell is valid or 0.0 if cell is missing
weight_mask = pd.DataFrame({
col: (~z_df[col].isna()).astype(float) * weights[col] for col in z_df.columns
})
# calculate the weighted sum of all z-scores for each stock ignoring missing values
weighted_sum = z_df.sum(axis=1, skipna=True)
# now compute the actual sum of the weights used
sum_w = weight_mask.sum(axis=1)
# get the weighted average z-score, and normalize so missing data doesn't
# lower score unfairly - prevents penalizing stocks with NaN
momentum = weighted_sum / sum_w.replace(0, np.nan)
# now bin the momentum into percentile ranks
pct = momentum.rank(pct=True, method='average')
df_with_label = df_raw.copy()
# label 0 = underperform, 1 = neurtral, 2 = outperform
df_with_label['label'] = pd.cut(pct,
bins=[-np.inf, 0.33, 0.66, np.inf],

labels=[0, 1, 2])

# drop na values from the label df

df_with_label = df_with_label.dropna(subset=['label']).copy()

# make the label an int, not categorical or object

df_with_label['label'] = df_with_label['label'].astype('int32')

# add distance to 52 week extremes

rng = (df_with_label['fiftyTwoWeekHigh'] - df_with_label['fiftyTwoWeekLow']).replace(0,
np.nan)

df_with_label['pos_in_52w'] = (df_with_label['currentPrice'] -
df_with_label['fiftyTwoWeekLow']) / (rng + 1e-9)

# add distance to moving averages

df_with_label['dist_ma50'] = (df_with_label['currentPrice'] - df_with_label['fiftyDayAverage']) /
(df_with_label['fiftyDayAverage'] + 1e-9)

df_with_label['dist_ma200'] = (df_with_label['currentPrice'] -
df_with_label['twoHundredDayAverage']) / (df_with_label['twoHundredDayAverage'] + 1e-9)

df_with_label['ma_cross'] = (df_with_label['fiftyDayAverage'] -
df_with_label['twoHundredDayAverage']) / (df_with_label['twoHundredDayAverage'] + 1e-9)

df_new_labels = df_with_label[['currentPrice', 'fiftyDayAverage',
'twoHundredDayAverage','dist_ma50', 'dist_ma200', 'ma_cross']].head(5)

print(df_new_labels.to_string(index=False))

# add liquidity size, shares outstanding ratios

df_with_label['log_mcap'] = np.log(df_with_label['marketCap'] + 1)

# high outstanding shares = lots of flots is short and performance may be poor

df_with_label['float_to_out'] = df_with_label['floatShares'] / (df_with_label['sharesOutstanding']
+ 1e-9)

# add z scores to the table for neural net to see each z score
```

```
df_with_label['z_4y_2y'] = z_4y_2y

df_with_label['z_2y_1y'] = z_2y_1y

df_with_label['z_1y_6m'] = z_1y_6m

df_with_label['z_6m_3m'] = z_6m_3m

df_with_label['z_3m_1m'] = z_3m_1m

df_with_label['z_1m_1d'] = z_1m_1d
```