

AI-Driven Stock Market Forecasting Using Financial Statement Analysis

Report (2)

By

Marwa Othman Esaa

Midocean University

Advisor

Dr. Ashraf Elsayed

Data Description

The dataset used in this study comprises historical financial ratios for 105 publicly listed Saudi companies included in the Tadawul All Share Index (TASI). These companies were selected based on the availability of consistent stock return records dating back to the first quarter of 2017 (2017Q1) through the first quarter of 2025 (2025Q1), covering a span of 33

The financial data includes a wide range of .consecutive quarters quantitative indicators, such as profitability ratios, leverage metrics, liquidity ratios, and market performance measures. This comprehensive time-series dataset enables a robust analysis of stock return behavior

To compile this dataset, financial statements and historical .over time stock returns were scraped and aggregated from multiple reputable sources includes: Investing.com - Tadawul (Saudi Stock Exchange) – MarketScreener - Yahoo Finance

Data Preprocessing

As part of the data preparation phase, several preprocessing operations were applied to ensure the dataset was clean, normalized All numerical columns were selected using type inference, ensuring that the dataset only retained quantitative variables necessary for time series modeling and stock return prediction. To enhance model training stability and convergence. The target variable (Stock Return) was normalized using Z-score standardization, computed as:

$$Z = \frac{X - \mu}{\sigma}$$

Finally, to handle missing values in numerical features were interpolated and forward back filled. Any remaining missing values after interpolation were filled with a sentinel value (-1.0) to avoid runtime errors during model training.

```

numeric_cols = df.select_dtypes(include='number').columns

target_col = "Stock Return"
return_mean = df[target_col].mean()
return_std = df[target_col].std()
df[target_col] = (df[target_col] - return_mean) / return_std

df[numeric_cols] = df.groupby("Name")[numeric_cols].transform(
    lambda group: group.interpolate(limit_direction='both').ffill().bfill()
)
df.fillna(-1.0, inplace=True)

missing = df.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print(missing)

Series([], dtype: int64)

```

GRU Modeling

prepare time-series data

```

class CompanyStockDataset(Dataset):
    def __init__(self, df, seq_len, feature_cols, target_col="Return"):
        self.sequences = []
        self.targets = []
        self.ids = []

        for co_id, group in df.groupby("CO_ID"):
            group = group.sort_values("YearQuarter")
            X = group[feature_cols].values
            y = group[target_col].values

            for i in range(len(X) - seq_len):
                self.sequences.append(X[i:i+seq_len])
                self.targets.append(y[i+seq_len])
                self.ids.append(co_id)

        self.sequences = torch.tensor(self.sequences, dtype=torch.float32)
        self.targets = torch.tensor(self.targets, dtype=torch.float32).unsqueeze(1)

    def __len__(self):
        return len(self.sequences)

    def __getitem__(self, idx):
        return self.sequences[idx], self.targets[idx]

```

Define GRU Model

Activation Function	No. Layers	LR		
Tanh	2	0.001		

```
class GRU(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, dropout=0.0):
        super(GRU, self).__init__()
        self.rnn = nn.GRU(input_dim, hidden_dim, batch_first=True, num_layers=2)
        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(hidden_dim, output_dim)
        self.activation = nn.Tanh() # bounded output in [-1, 1]

    def forward(self, x):
        out, _ = self.rnn(x)
        out = self.dropout(out[:, -1, :])
        out = self.fc(out)
        return self.activation(out)
```

Train GRU Model

```
model = GRU(input_dim=len(features), hidden_dim=64, output_dim=1).to(device)

def init_weights(m):
    if isinstance(m, nn.Linear):
        nn.init.xavier_uniform_(m.weight)
        nn.init.zeros_(m.bias)
model.apply(init_weights)

criterion = nn.SmoothL1Loss() # Huber Loss
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, factor=0.1, patience=5)

import matplotlib.pyplot as plt

train_losses = []
val_losses = []
val_accuracies = []

epochs = 100
for epoch in range(epochs):
    model.train()
    train_loss = 0
    for x, y in train_loader:
        x, y = x.to(device), y.to(device)
        optimizer.zero_grad()
        pred = model(x)
        loss = criterion(pred, y)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()

    model.eval()
    val_loss = 0
    with torch.no_grad():
        for x, y in val_loader:
            x, y = x.to(device), y.to(device)
            pred = model(x)
            val_loss += criterion(pred, y).item()
    scheduler.step(val_loss)
    train_losses.append(train_loss)
    val_losses.append(val_loss)
```

```

#accuracy per epoch
correct = 0
count = 0
with torch.no_grad():
    for x, y in val_loader:
        x, y = x.to(device), y.to(device)
        pred = model(x)
        correct += (torch.sign(pred) == torch.sign(y)).sum().item()
        count += y.size(0)
val_acc = correct / count
val accuracies.append(val_acc)

print(f"Epoch {epoch+1}/{epochs} - Train Loss: {train_loss:.4f} - Val Loss: {val_loss:.4f} - Val Acc: {val_acc:.2%}")

```

Epoch 100/100 - Train Loss: 51.6645 - Val Loss: 2.7501 - Val Acc: 75.93%

```

plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(train_losses, label='Train Loss')
plt.plot(val_losses, label='Val Loss')
plt.title('Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

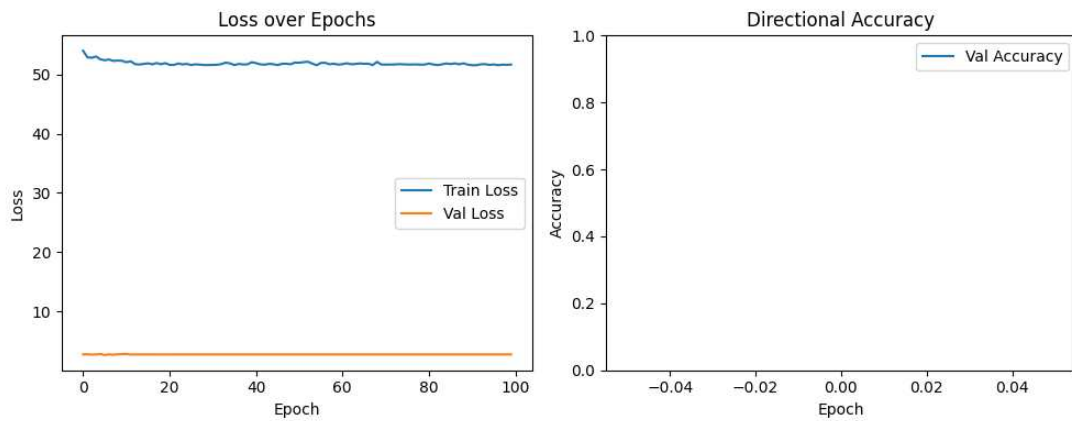
plt.subplot(1, 2, 2)
plt.plot(val_accuracies, label='Val Accuracy')
plt.title('Directional Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.legend()

plt.tight_layout()
plt.show()

# ==== Step 8: Sort top 20 ====
top_20 = sorted(final_scores.items(), key=lambda x: x[1], reverse=True)[:20]

print("\nTop 20 Companies by Predicted Return:")
for i, (name, score) in enumerate(top_20, 1):
    print(f"{i:2d}. {name:<25} Predicted Return: {score:.4f}")

```



Top 20 Companies by Predicted Return:

1. SAUDI ARAMCO	Predicted Return: 4.3685
2. SFICO	Predicted Return: 3.7025
3. MAHARAH	Predicted Return: 2.5154
4. SHAKER	Predicted Return: 1.2134
5. EIC	Predicted Return: 1.1269
6. LEEJAM SPORTS	Predicted Return: 0.9436
7. BAWAN	Predicted Return: 0.8870
8. SVCP	Predicted Return: 0.6822
9. ASTRA INDUSTRIAL	Predicted Return: 0.6764
10. ALBILAD	Predicted Return: 0.6735
11. ALRAJHI	Predicted Return: 0.5858
12. MEPCO	Predicted Return: 0.5486
13. RIBL	Predicted Return: 0.5357
14. MESC	Predicted Return: 0.4311
15. ARTEX	Predicted Return: 0.4258
16. SAUDI CABLE	Predicted Return: 0.4247
17. FOURTH MILLING	Predicted Return: 0.4220
18. AMIANTIT	Predicted Return: 0.4076
19. SSP	Predicted Return: 0.3895
20. SAUDI CERAMICS	Predicted Return: 0.3712

```
#Rolling Window

df["YearQuarter"] = df["Year"].astype(str) + "Q" + df["Quarter"].astype(str)

all_quarters = sorted(df["YearQuarter"].unique())
time_windows = []

# Use all 33 quarters dynamically by computing based on available quarters
total_quarters = len(all_quarters)

# Slide the window as long as we can get 12 for training, 4 for validation, and 4 for testing
for i in range(total_quarters - (12 + 4 + 4) + 1):
    train_q = all_quarters[i:i+12] # 12 quarters for training (~3 years)
    val_q = all_quarters[i+12:i+16] # 4 quarters for validation (~1 year)
    test_q = all_quarters[i+16:i+20] # 4 quarters for testing (~1 year)
    time_windows.append({"train": train_q, "val": val_q, "test": test_q})

seq_len = 4
epochs = 120

for window in time_windows:
    print(f"\n== Window: Train {window['train']} | Val {window['val']} | Test {window['test']} ==")
    train_df = df[df["YearQuarter"].isin(window["train"])]
    val_df = df[df["YearQuarter"].isin(window["val"])]
    test_df = df[df["YearQuarter"].isin(window["test"])]

    initial_val_count = val_df["CO_ID"].nunique()
    val_df = val_df.groupby("CO_ID").filter(lambda g: len(g) >= seq_len + 1)
    filtered_val_count = val_df["CO_ID"].nunique()
    print(f"Filtered out {initial_val_count - filtered_val_count} companies from val set.")

    test_df = test_df.groupby("CO_ID").filter(lambda g: len(g) >= seq_len)

    if val_df.empty or len(val_df["CO_ID"].unique()) < 2:
        print("Warning: Validation DataFrame is too small. Skipping this window.")
        continue
```