NYCU_Artificial_Intelligence Capstone_HW1

- This dataset is available at: here (https://github.com/hsiuyee/NYCU_Artificial_Intelligence_Capstone_Labs/tree/main/Lab1), in collaboration with teammate 111652040.
- This pdf is available at: here (https://hackmd.io/@Origamyee/B1pfNatqJl)
- This code is available at: here (https://github.com/hsiuyee/NYCU_Artificial_Intelligence_Capstone/tree/main/Lab1)

Part 0. Overview

In this homework, our goal is to predict the BTC/USDT cryptocurrency close price change five hours ahead and use this prediction to design our trading strategy. Below, we outline the process step by step:

1. Create Dataset
2. Clean Up Dataset
3. Algorithm and Analysis
4. Experiments
5. Discussion
6. References

Part 1. Create Dataset

To make predictions, we need OHLC (open, high, low, close) price data. Therefore, we use the Binance API to retrieve such data (details available at: Binance Public Data (https://github.com/binance/binance-public-data) and Kline/Candlestick Data (https://developers.binance.com/docs/derivatives/usds-margined-futures/market-data/rest-api/Kline-Candlestick-Data)). The implementation is handled by `preprocess.py` . We set the K-line parameters as follows:

a. `symbol = BTCUSDT`

b. `interval = '1h'`

c. `start_date = '2024-11-01'`

d. `end_date = '2025-02-01'`

If some column values are missing, we simply use `dropna` to remove them. After running `preprocess.py`, we obtain a CSV file named `klines_BTC.csv`, which follows the format below:

klines_BTC

open_time	open	high	low	close	volume	taker_buy_base_asset_volume	taker_buy_quote_asset_volume
2024-11-01 00:00:00	70321.9	70500.0	70160.9	70188.6	6293.539	3122.97	219650270.5021
2024-11-01 01:00:00	70188.7	70384.6	69312.2	69424.9	23094.976	9133.652	637407952.6586
2024-11-01 02:00:00	69424.9	69621.2	68870.4	69592.7	25645.369	12269.503	849623340.7967
2024-11-01 03:00:00	69592.7	69739.0	69357.1	69397.7	6709.285	3069.884	213563233.8988
2024-11-01 04:00:00	69397.7	69650.0	69363.9	69618.5	3543.899	1949.728	135516392.2082

Part 2. Clean Up Dataset

2.1 Feature Engineering

We add the target label **target_pct_change** to the dataset, defined by $(\mathrm{Close}_{i+1} - \mathrm{Close}_i)/\mathrm{Close}_i$. Since `OHLC, Volume, Taker buy base asset volume, and Taker buy quote asset volume` alone may not fully represent the price trend, we incorporate additional common factors to enhance the model's ability to fit price changes.

Specifically, we add volatility, price_range, ma_7, and other similar features. The implementation is handled by `add_factors.py`. After running `add_factors.py`, we obtain a CSV file named `klines_BTC_with_factors.csv`, which follows the format below:

open_time	open	high	low	close	volume	taker_buy_base_asset_volume	taker_buy_quote_asset_volume	return	log_return	volatility	price_range	ma_7	ma_25	ma_99
2024-11-08 12:00:00	75952.0	76189.9	75940.0	76078.9	5211.542	2915.219	221671978.1392	0.001672110901768110	0.001670714480764680	0.004598882336064350	0.0032902359384873900	76018.25714285710	75901.02800000000	72607.94343434340
2024-11-08 13:00:00	76078.4	76300.0	75855.0	76280.3	7908.982	3965.444	301591667.8091	0.0026472517347124700	0.0026437539355157900	0.004495756127947890	0.0058492292161770000	76075.04285714290	75951.53600000000	72683.3303030303
2024-11-08 18:00:00	76390.0	77069.8	76347.1	76803.0	22926.757	12278.253	942062948.4588	0.00540646681502821	0.00539190433743863	0.0041582460364935800	0.009460662390365190	76235.8	76090.948	73066.16767676770
2024-11-08 19:00:00	76802.8	77182.9	76728.4	76894.4	15879.932	7713.071	593816108.5131	0.0011900576800385500	0.0011893501226980300	0.004157422764118060	0.005917752998588590	76352.3	76102.452	73158.66767676770
2024-11-08 20:00:00	76894.4	77080.4	76401.1	76563.5	15592.359	7334.344	562098766.1862	-0.004303304271832450	-0.00431259013516597	0.004245478524186070	0.008834193387294630	76392.75714285720	76103.66	73247.05555555560
2024-11-08 21:00:00	76563.5	76675.1	76352.7	76463.4	6415.22	3090.686	236435371.0107	-0.0013074114950336600	-0.001308266903103850	0.003962874544606640	0.004210883776212020	76493.44285714290	76105.552	73336.20505050500
2024-11-08 22:00:00	76463.4	76600.0	76278.5	76350.0	4623.227	2149.277	164366345.8872	-0.0014830624847965700	-0.0014841633104936300	0.00397988918803213	0.004204626004075150	76482.97142857140	76125.004	73422.10707070710
2024-11-08 23:00:00	76350.0	76502.0	76280.1	76482.2	3136.759	1627.846	124379920.9532	0.001731499672560590	0.0017300023551564500	0.003990098900186820	0.002906352324819830	76563.78571428570	76149.59600000000	73514.44141414140
2024-11-09 00:00:00	76482.2	76618.3	76339.1	76409.5	4076.52	1978.202	151353173.1061	-0.0009505479706388710	-0.0009510000278518610	0.003983451465706030	0.003650522605259750	76566.57142857140	76171.6	73608.50707070710
2024-11-09 01:00:00	76409.4	76529.6	76230.7	76300.0	3846.096	1832.402	139908609.45	-0.001433067877685380	-0.0014340957015346600	0.003996444810516500	0.003911822367405170	76494.71428571430	76183.348	73693.29494949500
2024-11-09 02:00:00	76300.0	76418.7	76122.4	76418.6	5312.114	2530.895	193046493.0996	0.001554390563564880	0.0015531837489651200	0.003929638227357040	0.0038833551769332000	76426.74285714290	76200.84	73780.00505050510
2024-11-09 03:00:00	76418.6	76434.1	76235.0	76330.0	2557.821	1133.323	86476310.7729	-0.0011594036006941100	-0.0011600762289974700	0.00391801284271743	0.002605386646706510	76393.3857142857	76225.216	73864.050505050[illegible]

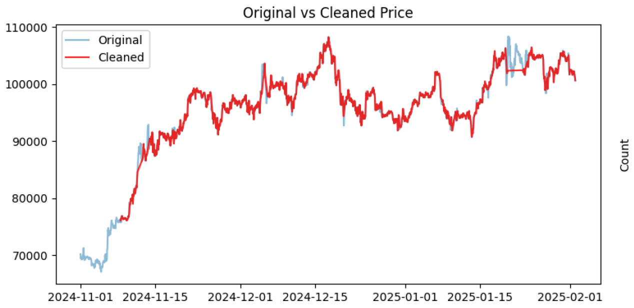
klines_BTC_with_factors

atr	ema_12	ema_26	macd	macd_signal	macd_hist	rsi	bb_middle	bb_upper	bb_lower	bb_width	bb_pct
703.7040000000000	75984.27301388220	75754.99882960540	229.2741842768160	283.91645710162300	-54.64227282480720	54.33561559591660	76048.855	76546.08809571210	75551.62190428790	0.013076675400624700	0.530212188467635
706.4319999999990	76029.81562713110	75793.91005954540	235.90556758568000	274.3142791984340	-38.4087116127543	57.994301994302200	76049.115	76546.85139302860	75551.37860697140	0.013089866805907700	0.7322363837947480
715.162666666666	76198.45815721030	75952.41402007140	246.044137138917	234.39598828793800	11.648148850979000	60.702071795777100	76049.32000000000	76583.57638079370	75515.0636192063	0.01405026056232180	1.2053542335613600
707.4039999999990	76305.52613302410	76022.19079563690	283.33533738719400	244.18385810778900	39.151479279404900	60.159710106026000	76101.07	76746.7554834799	75455.38451652010	0.016969156504104000	1.1143316059425500
708.6106666666660	76345.2144202512	76062.28779305860	282.9266271926320	251.93241192475800	30.994215267874000	57.24410957154720	76128.93000000000	76804.77766922560	75453.0823307744	0.01775534397306280	0.8214999620387290
705.8520000000000	76363.39681713560	76091.99982164380	271.3969954918720	255.82532863818100	15.571666853691400	53.24154139506000	76153.035	76840.99344205710	75465.07655794290	0.018067787897281700	0.7255695846045130
688.8973333333330	76361.33576834550	76111.11095386140	250.22481448414300	254.7052258073730	-4.48041132322993	56.15499742762260	76184.50500000000	76846.25711601500	75522.75288398500	0.017372354549392400	0.6250430455716630
680.6559999999990	76379.93026552310	76138.5990418675	241.3312236556380	252.03042537702600	-10.699201721388500	56.05325457656520	76213.595	76873.91110444330	75553.27889555670	0.017328039818703300	0.7033912229253190
677.1853333333330	76384.47945544270	76158.66578661400	225.81366882861800	246.78707406734400	-20.973405238726800	51.92060827490870	76242.425	76882.80468301530	75602.04531698480	0.0167985129805418	0.630449953700374
671.4293333333330	76371.48261614380	76169.13499103880	202.3476251049430	237.89918427486400	-35.55155916992100	54.51105409118010	76258.125	76887.3334346289	75628.9165653711	0.01650206937631640	0.5332759366335390
667.1293333333330	76378.73144442940	76187.61388620260	191.11755822680400	228.5428590652520	-37.42530083844800	54.41180290397170	76284.91500000000	76892.07746772030	75677.75253227970	0.01591828391550970	0.6100899735304370
641.0200000000000	76371.23429913250	76198.16100870850	173.07329042405800	217.44894533701300	-44.375654912955700	50.664954108800900	76292.865	76897.90727351030	75687.82272648970	0.01586104476507110	0.5306879383687930
631.7306666666660	76370.98286849680	76210.86019655450	160.1226719422270	205.983690658056	-45.8610187158294	59.385560675883300	76320.16500000000	76883.9183140038	75756.41168599620	0.01477337775681660	0.5438445316169490
603.4026666666660	76396.98550411260	76235.2409286125	161.74457550013900	197.13586762647300	-35.39129212633360	52.11359437823740	76348.88	76894.83652651080	75802.92347348920	0.01430162502739440	0.675032251396876
574.2746666666660	76411.83388809530	76254.37123446640	157.4626536288710	189.20122482695200	-31.738571198081600	62.54129716571030	76360.86000000000	76908.55350783440	75813.16649216570	0.01434487531529550	0.6210896223003050
562.5866666666660	76429.07482838830	76274.33633233560	154.73849605277100	182.30867907211600	-27.57018301934530	53.604694987347100	76389.46000000000	76906.09351725910	75872.82648274090	0.013526303687945000	0.6301115737837310

volume_change	volume_ma_7	volume_relative	obv	hour	day_of_week	is_weekend	month	target_binary	target_pct_change	target_abs_change	target_multiclass	target_log_return	target_normalized_change
-0.05847999807777180	5737.050714285720	0.9084008943868740	334093.0600000000	12	4	0	11	1	-0.26402622957697500	201.40000000000900	2	0.0026437539355157900	0.8100828433795870
0.517589611673474	6432.078	1.229615374689180	342002.0420000000	13	4	0	11	0	0.6886346896589850	-521.6999999999970	3	-0.006862744304934330	-2.0962903549067200
0.3619498300292120	16429.414714285700	1.3954701003478100	376888.4350000000	18	4	0	11	1	-0.11886431261574100	91.39999999999420	2	0.0011893501226980300	0.3421578226677970
-0.3073624848032370	17953.47042857140	0.8845048684697990	392768.3670000000	19	4	0	11	0	0.43219027343315300	-330.8999999999940	2	-0.00431259013516597	-1.0249572228729000
-0.018109208528096900	19051.095714285700	0.8184494600123110	377176.0080000000	20	4	0	11	0	0.1309123057567470	-100.10000000000600	2	-0.001308266903103850	-0.29622059691134400
-0.5885664253882300	17591.203285714300	0.36468340998650000	370760.7880000000	21	4	0	11	0	0.14852652259331700	-113.39999999999400	2	-0.0014841633104936300	-0.3296710762366190
-0.2793346136219800	14162.328	0.3264454120819680	366137.5610000000	22	4	0	11	1	-0.17285067636652100	132.1999999999970	2	0.0017300023551564500	0.3995453790023290
-0.3215217422808790	12201.147000000000	0.2570872230291140	369274.3200000000	23	4	0	11	0	0.09514523717599750	-72.69999999999710	2	-0.0009510000278518610	-0.22019756753005700
0.2995961755429730	10378.682	0.39277819669202700	365197.8000000000	0	5	1	11	0	0.14351245085189000	-109.5	2	-0.0014340957015346600	-0.3419846160136860
-0.0565246828176974	7652.873285714290	0.5025688857516500	361351.7040000000	1	5	1	11	1	-0.1551978183321980	118.60000000000600	2	0.0015531837489651200	0.3769815961521640
0.38117041280300800	6143.185000000000	0.8647165924516360	366663.8180000000	2	5	1	11	0	0.11607493777021600	-88.60000000000580	2	-0.0011600762289974700	-0.2918493968581190
-0.5184928260199240	4281.108142857140	0.5974670376565050	364105.9970000000	3	5	1	11	1	-0.051853093377474100	39.60000000000580	2	0.0005186654174306320	0.13089994446186900
-0.32316100305689900	3611.9671428571400	0.47930474767014600	365837.2300000000	4	5	1	11	1	-0.22262869088057700	170.39999999999400	2	0.0022287687697463900	0.6045197279240010
2.1630179184431000	3733.7805714285700	1.4665888622118100	371313.1510000000	5	5	1	11	0	0.06078947884460280	-46.5	2	-0.0006077100952547070	-0.17034323335913300
0.3965995857135270	4378.196285714290	1.7467624795520800	363665.4820000000	6	5	1	11	1	-0.03972615091493160	30.39999999999420	2	0.0003973404384070550	0.11101099269915600
-0.553334617384722	4283.829	0.7974055453660730	367081.4310000000	7	5	1	11	1	-0.021034753070292600	16.10000000000580	2	0.0002103696568476020	0.062326579527007900
0.22924024919575800	4334.247	0.9688008089986560	371280.4530000000	8	5	1	11	1	-0.0067933717594281800	5.19999999999709	2	6.79360251938475E-05	0.020835934367829200

2.2 Data Cleaning with Autoencoders and Z-score

Due to the high level of noise in the market, I use autoencoders and Z-score to filter out noise while preserving useful data. The refined result is shown below:



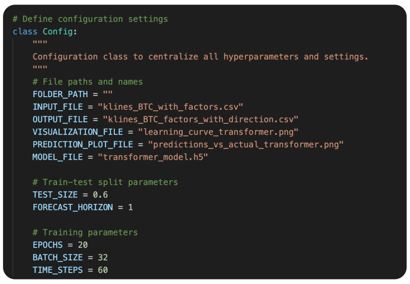
Part 3. Algorithm and Analysis

Models Selection

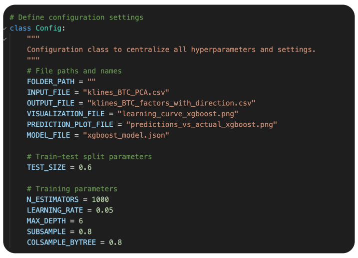
We use the following models from the `sklearn` library to predict `target_pct_change` or `target_multiclass` :

- Supervised learning: XGBoost regression and Transformer
- Unsupervised learning: K-Means (Class 1: lower than -2%, Class 2: -2% to -0.5%, Class 3: -0.5% to 0.5%, Class 4: 0.5% to 2%, Class 4: greater than 2%)

And the default hyperparameters and description are below:

<table>
<tr><th>Transformer</th><th>XGBoost</th><th>Kmeans</th></tr>
<tr>
<td>
<pre><code># Define configuration settings
class Config:
    """
    Configuration class to centralize all hyperparameters and settings.
    """
    # File paths and names
    FOLDER_PATH = ""
    INPUT_FILE = "klines_BTC_with_factors.csv"
    OUTPUT_FILE = "klines_BTC_factors_with_direction.csv"
    VISUALIZATION_FILE = "learning_curve_transformer.png"
    PREDICTION_PLOT_FILE = "predictions_vs_actual_transformer.png"
    MODEL_FILE = "transformer_model.h5"

    # Train-test split parameters
    TEST_SIZE = 0.6
    FORECAST_HORIZON = 1

    # Training parameters
    EPOCHS = 20
    BATCH_SIZE = 32
    TIME_STEPS = 60</code></pre></td>
<td>
<pre><code># Define configuration settings
class Config:
    """
    Configuration class to centralize all hyperparameters and settings.
    """
    # File paths and names
    FOLDER_PATH = ""
    INPUT_FILE = "klines_BTC_PCA.csv"
    OUTPUT_FILE = "klines_BTC_factors_with_direction.csv"
    VISUALIZATION_FILE = "learning_curve_xgboost.png"
    PREDICTION_PLOT_FILE = "predictions_vs_actual_xgboost.png"
    MODEL_FILE = "xgboost_model.json"

    # Train-test split parameters
    TEST_SIZE = 0.6

    # Training parameters
    N_ESTIMATORS = 1000
    LEARNING_RATE = 0.05
    MAX_DEPTH = 6
    SUBSAMPLE = 0.8
    COLSAMPLE_BYTREE = 0.8</code></pre></td>
<td>
<pre><code># Define configuration settings
class Config:
    """
    Configuration class to centralize all hyperparameters and settings.
    """
    # File paths and names
    INPUT_FILE = "klines_BTC_with_factors.csv"
    OUTPUT_FILE = "klines_BTC_clusters.csv"
    CLUSTER_PLOT_FILE = "kmeans_clusters.png"

    # Clustering parameters
    TIME_STEPS = 60
    N_CLUSTERS = 5 # 分成 4 群
    RANDOM_STATE = 42 # 保證每次結果一樣</code></pre></td>
</tr>
</table>

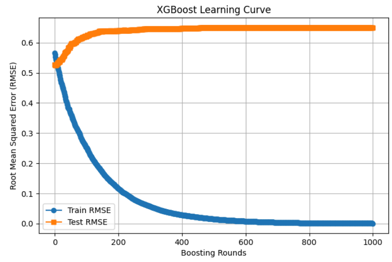
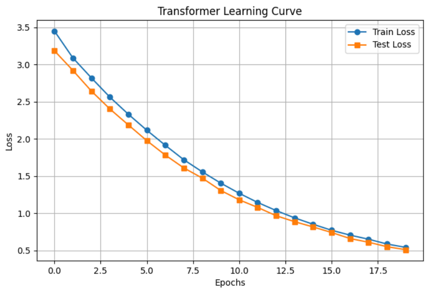
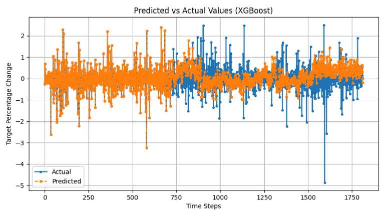
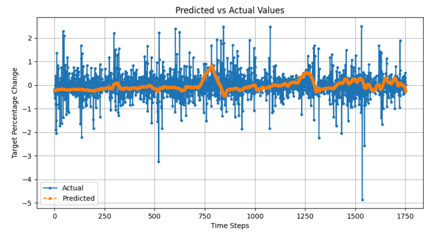
- XGBoost: A gradient boosting algorithm that builds decision trees sequentially, optimizing for efficiency and performance.
- Transformer: A deep learning model using self-attention to capture long-range dependencies in sequential data.
- K-Means: An unsupervised clustering algorithm that partitions data into K groups by minimizing intra-cluster variance.

Reference Public Libraries

- [scikit-learn (https://scikit-learn.org/stable/)](https://scikit-learn.org/stable/)

- [XGBoost (https://xgboost.readthedocs.io/en/stable/)](https://xgboost.readthedocs.io/en/stable/)
- [TensorFlow (https://www.tensorflow.org/?hl=zh-tw)](https://www.tensorflow.org/?hl=zh-tw)

Evaluate the performance (supervised learning)

compare	XGBoost	Transformer
loss		
prediction		

Evaluate the performance (unsupervised learning)

Kmeans

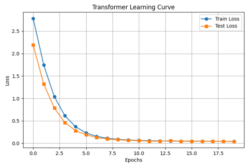
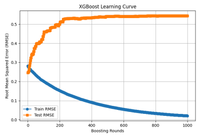
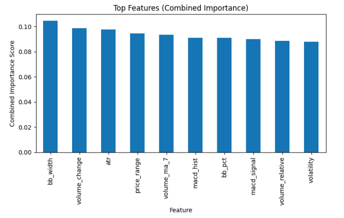
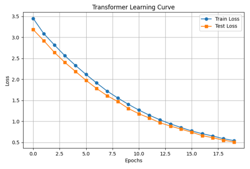
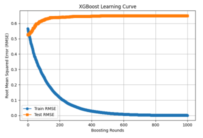
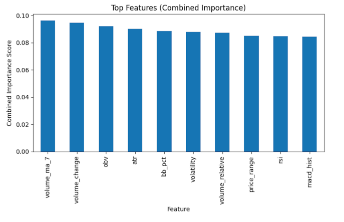
1. Silhouette Score: 0.0959 (higher is better)
2. Inertia (SSE): 2260203.5053 (lower is better)
3. Adjusted Rand Index (ARI): 0.0064 (higher is better)
4. Normalized Mutual Information (NMI): 0.0063 (higher is better)
5. Accuracy: 0.3333 (higher is better)

Part 4. Experiments

Problem 1

> What if we use a different time interval?

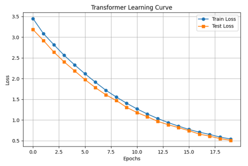
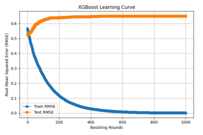
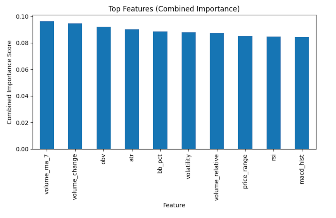
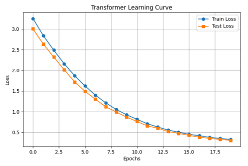
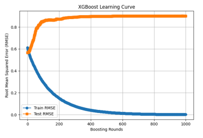
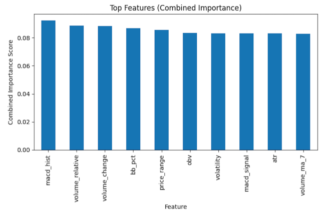
We modified the dataset's time interval setting from `1h` to `15m`, and the comparison results are shown below.

time_interval	Transformer	XGBoost	important features
15m			
1h			

As we can see, both models achieve better performance with a smaller time interval, as indicated by their lower testing loss. Intuitively, a smaller time interval functions similarly to data augmentation. Additionally, the Transformer converges more efficiently, while XGBoost overfits at a slower rate. Moreover, the important features remain unchanged.

Problem 2

> What if we do not perform data cleaning with Autoencoders and Z-score?

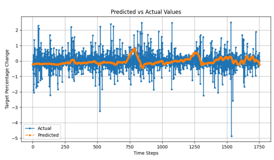
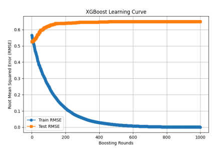
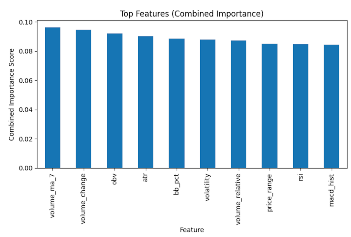
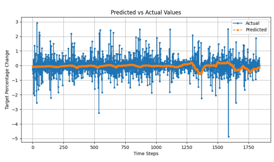
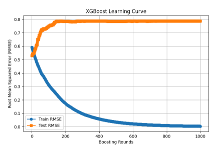
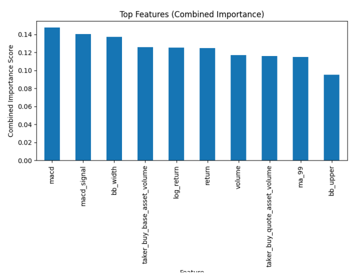
data_cleaning	Transformer	XGBoost	Important features
YES			
NO			

Intuitively, this would lead to poor training and testing results.
We compare the effect of data cleaning using Autoencoders and Z-score above. As we can see, if we do not perform data cleaning, the training and testing performance of XGBoost deteriorates significantly, while the Transformer model remains nearly the same. Additionally, the important features differ slightly; for example, `rsi` and `macd_signal` do not appear in the same diagram. Why do we obtain such a result? Since we did not perform data cleaning, XGBoost may learn from noise, whereas the Transformer is more resistant to noise as it can capture temporal behavior.

Problem 3

> What if we delete some important features?

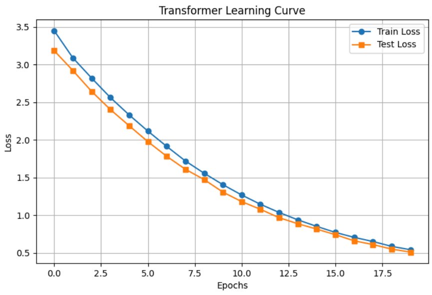
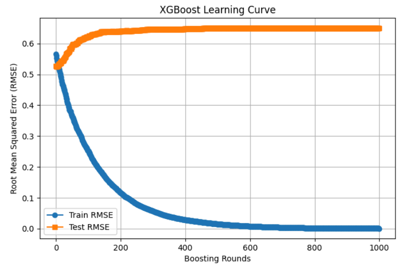
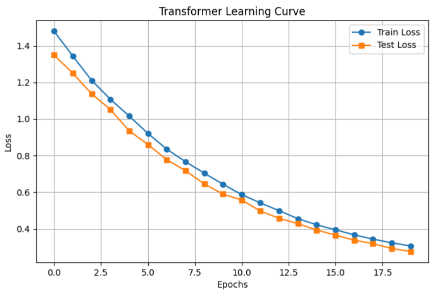
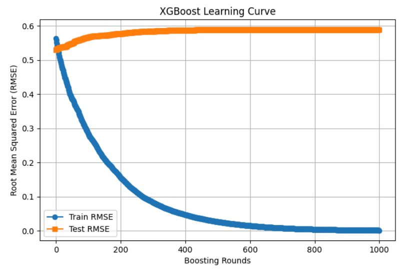
First, we define "importance" as the total decrease in impurity when a feature is used for splitting across all trees. Second, we remove the top 10 most important features and analyze their effect. The experiment results are shown below.

compare	Transformer	XGBoost	Important features
before			
after			

As we can see, when we change only the feature vectors, the Transformer model becomes more cautious after removing important features. Without key features, the model has less confidence and predicts price changes more conservatively. Additionally, this leads to an increase in XGBoost's testing loss.

Problem 4

> What if we apply the PCA method to reduce noise, remove unimportant features, and decrease training time?

compare	Transformer	XGBoost
before		
after		

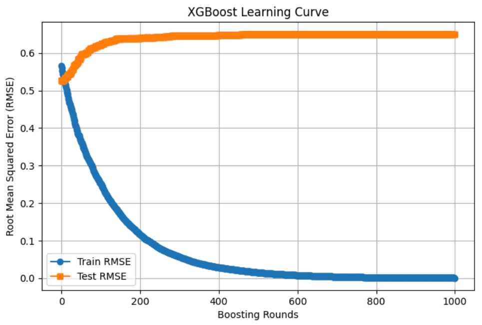
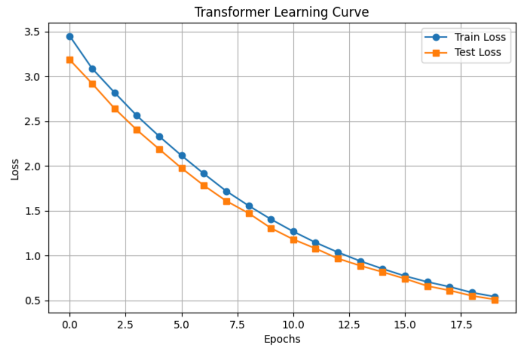
```
✅ Initial feature count (excluding targets): 34
✅ PCA reduced dimensions to 12, preserving 95.0% variance.
❌ Dropped 22 features (from 34 → 12)
✅ Retained feature names: ['open', 'high', 'low', 'close', 'volume', 'taker_buy_base_asset_volume'
, 'taker_buy_quote_asset_volume', 'return', 'log_return', 'volatility', 'price_range', 'ma_7']
✅ Explained variance plot saved to pca_explained_variance.png
✅ PCA-transformed data saved to klines_BTC_PCA.csv
 origamyee@Hsiu-IdeMacBook-Pro  ~/Desktop/NYCU_Artificial_Intelligence_Capstone/Lab1  main ±
python3.12 predict_transformer.py
✅ Selected 12 features for training.
```

As we can see, when we apply the PCA method to the dataset, the training and testing loss significantly decrease after a few epochs in both models. This is because the PCA model reduces correlation among feature vectors and simplifies the complexity of the problem.

Problem 5

> What if we use a non-temporal model to predict temporal behavior compared to using a temporal model?

The XGBoost model tends to overfit compared to the Transformer model. Below is an example where we use similar loss functions: Pseudo-Huber error in XGBoost and Huber loss in the Transformer model.

XGBoost	Transformer
	

As we can see, the testing loss increases as the number of training iterations increases. This is a typical sign of overfitting. Intuitively, since the data depends on time, the model may fail to capture temporal effects properly.

Part 5. Discussion

Problem 1

> Describe experiments that you would do if there were more time available.

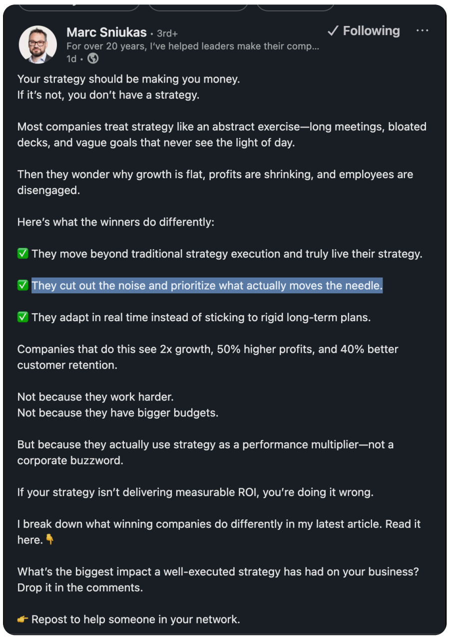
1. Add more statistical indices and analyze their effects.
2. Expand our project—since we have the next price change, we can develop a long/short strategy and evaluate its performance using PnL as a metric.
3. Predict more hyperparameters, such as the stop-loss threshold, to improve balance.
4. Try more models and analyze their effects.

Problem 2

> What I have learned from the experiments as well as your remaining questions?

Parts 1 and 2

In Parts 1 and 2, I learned how to clean up data. Initially, I thought we could simply add statistical indices to enhance our fitting power. However, after reading a LinkedIn post, I realized that the data might contain noise. This naturally led to an important question: "How can we filter out noise while keeping useful data?"



I tried to find the answer on the Internet, and the top three posts that inspired me are:

1. Towards Denoised Market Indices Pt. 1 — Static Indices (https://medium.com/@lavaroninic/towards-denoised-market-indices-pt-1-static-indices-ac9004e930b1)
2. Autoencoders Simplified: Real-World Data Cleaning Application Satejraste (https://medium.com/@satejraste/autoencoders-simplified-real-world-data-cleaning-application-5dda1b3c686d)
3. AutoEncoder (一)-認識與理解 (https://medium.com/ml-note/autoencoder-%E4%B8%80-%E8%AA%8D%E8%AD%98%E8%88%87%E7%90%86%E8%A7%A3-725854ab25e8)

Therefore, I used an autoencoder an Z-score for denoising.

Part 3

In Part 3, I learned how to use PCA for denoising. Thanks to the following tutorial, I gained an understanding of its mathematical intuition and why it works intuitively.

1. Principal Component Analysis(PCA) (https://www.geeksforgeeks.org/principal-component-analysis-pca/)
2. Learning Model : Unsupervised Machine Learning_主成分分析（PCA）原理詳解 (https://medium.com/ai%E5%8F%8D%E6%96%97%E5%9F%8E/preprocessing-data-%E4%B8%BB%E6%88%90%E5%88%86%E5%88%86%E6%9E%90-pca-%E5%8E%9F%E7%90%86%E8%A9%B3%E8%A7%A3-afe1fd044d4f)
3. 【机器学习】降维——PCA（非常详细） (https://zhuanlan.zhihu.com/p/77151308)

Part 6. References

1. Binance Public Data (https://github.com/binance/binance-public-data)
2. Kline/Candlestick Data (https://developers.binance.com/docs/derivatives/usds-margined-futures/market-data/rest-api/Kline-Candlestick-Data)
3. scikit-learn (https://scikit-learn.org/stable/)
4. XGBoost (https://xgboost.readthedocs.io/en/stable/)
5. TensorFlow (https://www.tensorflow.org/?hl=zh-tw)
6. Towards Denoised Market Indices Pt. 1 — Static Indices (https://medium.com/@lavaroninic/towards-denoised-market-indices-pt-1-static-indices-ac9004e930b1)
7. Autoencoders Simplified: Real-World Data Cleaning Application Satejraste (https://medium.com/@satejraste/autoencoders-simplified-real-world-data-cleaning-application-5dda1b3c686d)
8. AutoEncoder (一)-認識與理解 (https://medium.com/ml-note/autoencoder-%E4%B8%80-%E8%AA%8D%E8%AD%98%E8%88%87%E7%90%86%E8%A7%A3-725854ab25e8)
9. Principal Component Analysis(PCA) (https://www.geeksforgeeks.org/principal-component-analysis-pca/)
10. Learning Model : Unsupervised Machine Learning_主成分分析（PCA）原理詳解 (https://medium.com/ai%E5%8F%8D%E6%96%97%E5%9F%8E/preprocessing-data-%E4%B8%BB%E6%88%90%E5%88%86%E5%88%86%E6%9E%90-pca-%E5%8E%9F%E7%90%86%E8%A9%B3%E8%A7%A3-afe1fd044d4f)
11. 【机器学习】降维——PCA（非常详细） (https://zhuanlan.zhihu.com/p/77151308)
12. ChatGPT (https://chatgpt.com/)