PA026 : Project report

Introduction

In recent years, digital marketplaces have emerged as thriving ecosystems, allowing users to buy, sell, and trade virtual items. Among these platforms, the Steam Marketplace [1] stands out as one of the largest and most vibrant, with millions of active users engaging in transactions every day. The Steam Marketplace provides various in-game items, ranging from character skins and weapons to trading cards and virtual accessories.

The fluctuating nature of item prices in the Steam Marketplace presents a unique challenge and opportunity for individuals seeking to make profitable investments similar to more established markets such as the stock market and forex.

This project focuses on evaluating the effectiveness of different machine and deep learning models for predicting future prices of items with the main target of making a profit based on their predictions.

Data

As the task is price prediction, primary data are historical prices of items from the steam marketplace (see 1). As these are not directly available as datasets, the first step was to scrape them from Steam marketplace. It was relatively straightforward as steam market private API has an endpoint that returns the whole history for a given item in a nice clean JSON 2.

Additionally, I have used another source to obtain some fundamental information relevant to the predictions. That includes Google trends for related keywords such as item name, steam marketplace, etc. Other data were obtained from gaming subreddits on Reddit. However, this is not included in this final evaluation as the data provider (Push shift API [2]) is currently not functional because of some legal problems with Reddit.

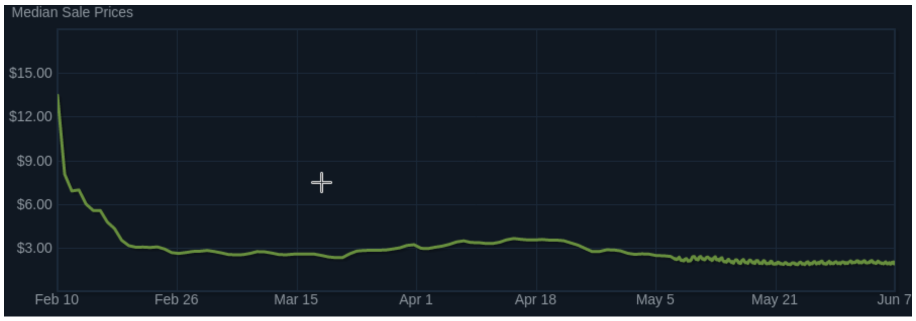
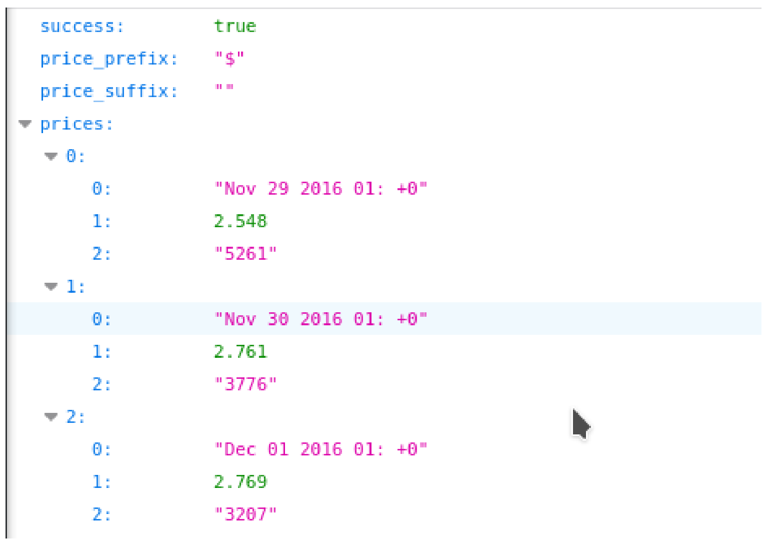


Figure 1: Item prices history from steam market place.



```
success:        true
price_prefix:   "$"
price_suffix:   ""
▼ prices:
  ▼ 0:
      0:        "Nov 29 2016 01: +0"
      1:        2.548
      2:        "5261"
  ▼ 1:
      0:        "Nov 30 2016 01: +0"
      1:        2.761
      2:        "3776"
  ▼ 2:
      0:        "Dec 01 2016 01: +0"
      1:        2.769
      2:        "3207"
```

Figure 2: JSON with historical item prices obtained from private Steam API.

Aproach

The primary focus was on regression-based models, starting with basic models such as autoregressive moving average models (ARIMA, SARIMA) and linear regression. However, these simple models did not yield satisfactory results, prompting the exploration of more complex alternatives. That included deep learning models for time series predictions using implementation for the TSAI library. A list of evaluated model architectures can be found in the evaluation section.

Training and Inference

The inference is closely connected with training as it's done as a forward walk through the data where in each step model is trained on history relative to the current position. Using this model next timestamp is predicted. Prediction is made on multiple items in parallel (in the case of evaluation, nine items were used).

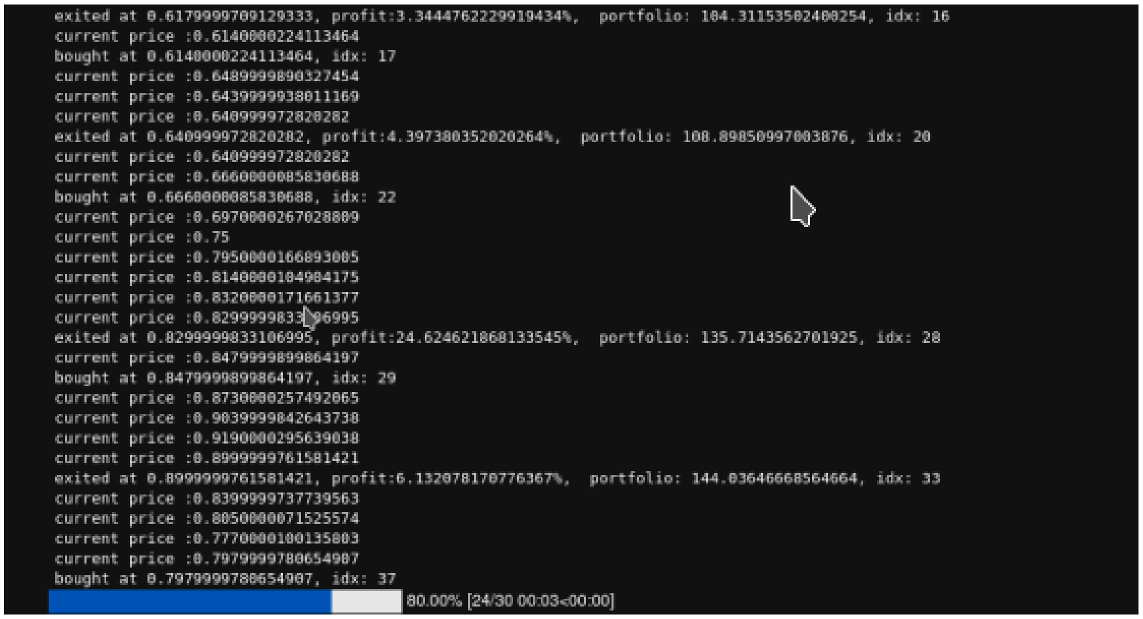
Evaluation and results

Evaluation can be divided into two parts. The first one is based on several metrics such as MSE, MAE, and directional accuracy (accuracy when only accounting for predicted direction from the regression model). Secondly, the potential profitability of the prediction is evaluated by simulated trading on unseen historical data (known as backtesting in the evaluation of trading systems).

For this purpose, trading is simulated using simple entry logic - from predicted items the one with the highest predicted price gain is selected, and if this gain is higher than the given threshold item is bought.

The output of this is the profit percentage achieved during the testing period. This was also evaluated considering realistic trading fees, as these have a crucial effect on the final result. Figure 3 shows running evaluation with partial results from already executed trades.

Results for all evaluated models can be seen in the table below 1. Results with different realistic fee levels are shown in table 2



```
exited at 0.6179999709129333, profit:3.3444762229919434%,  portfolio: 104.31153502400254, idx: 16
current price :0.6140000224113464
bought at 0.6140000224113464, idx: 17
current price :0.6489999890327454
current price :0.6439999938011169
current price :0.640999972820282
exited at 0.640999972820282, profit:4.397380352020264%,  portfolio: 108.89850997003876, idx: 20
current price :0.640999972820282
current price :0.6660000085830688
bought at 0.6660000085830688, idx: 22
current price :0.6970000267028809
current price :0.75
current price :0.7950000166893005
current price :0.8140000104904175
current price :0.8320000171661377
current price :0.8299999833106995
exited at 0.8299999833106995, profit:24.624621868133545%,  portfolio: 135.7143562701925, idx: 28
current price :0.8479999899864197
bought at 0.8479999899864197, idx: 29
current price :0.8730000257492065
current price :0.9039999842643738
current price :0.9190000295639038
current price :0.8999999761581421
exited at 0.8999999761581421, profit:6.132078170776367%,  portfolio: 144.03646668564664, idx: 33
current price :0.8399999737739563
current price :0.8050000071525574
current price :0.7770000100135803
current price :0.7979999780654907
bought at 0.7979999780654907, idx: 37
```

80.00% [24/30 00:03<00:00]

Figure 3: Backtesting- evaluation by simulated trading using the model on unseen historical data.

Entry treshold

As stated before, the threshold was used to determine whether to enter the trade based on predicted gain. For the tables above, this threshold was set to 0, which means if the predicted gain is positive the item is bought. This is possibly not the optimal approach, especially if there are significant trading fees. The figure 4 shows how changing the threshold influences the result.

Implementation details and running instructions

Implementation is provided in the form of Jupiter notebooks. It uses a few standard data science libraries such as pandas, numpy and TSAI [3] library,

Model name	Directional accuracy	MAE	Return 0 % fee	avg trade
GRUPlus	93.77%	1.19	288.07%	11.39%
InceptionTimePlus	98.5%	6.37	217.48%	83.07%
gMLP	93.39%	0.987	418.9%	9.85%
OmniScaleCNN	97.62%	4.04	129.22%	19.67%
XCM	98.2%	5.95	190.91%	46.54%
LSTMPlus	94.82%	1.63	470.33%	21.96%
mWDNPlus	95.21%	1.43	68.10%	4.07%
Random (benchmark)	51%	-	-26.69%	-1.28%
Lookahead (benchmark)	100%	0	1190033%	6.647%

Table 1: This table shows a comparison of different models. It includes returns from backtesting and a few other metrics, such as MAE and directional accuracy, which means the accuracy of the regression model when accounting only for direction. Model names are same as in TSAI library [3]

Model name	Return 1 % fee	Return 3 % fee	Return 5 % fee
GRUPlus	243.7%	145.7%	79.8%
InceptionTimePlus	211.2%	198.7%	186.5%
gMLP	307.8%	149.8%	51.52%
OmniScaleCNN	118.0%	96.8%	77.5%
XCM	182.2%	160.5%	149.3%
LSTMPlus	415.8%	320.6%	241.5%
mWDNPlus	46.0%	9.7%	-18%
Lookahead (benchmark)	253072.0%	10825.6%	341.5%

Table 2: This table shows returns for tested models with different fee levels.

which works with pytorch backend and provides an implementation of state-of-the-art models for time series predictions.

Running the project

The main code is located in a notebook called Prediciton.ipynb , and results can be reproduced by running this notebook. It produces pickle files with back-testing results which are further evaluated in notebook Eval.ipynb. There are a few other notebooks containing data scraping and preprocessing code, but it is unnecessary as final data are included in the archive.

Conclusions and future work

In this project, different approaches for price prediction on the steam marketplace were evaluated with some interesting results. Some models achieved positive results in the backtest eaven with realistic trading fees, so they could be potentially profitable. When it comes to potential feature improvements it will

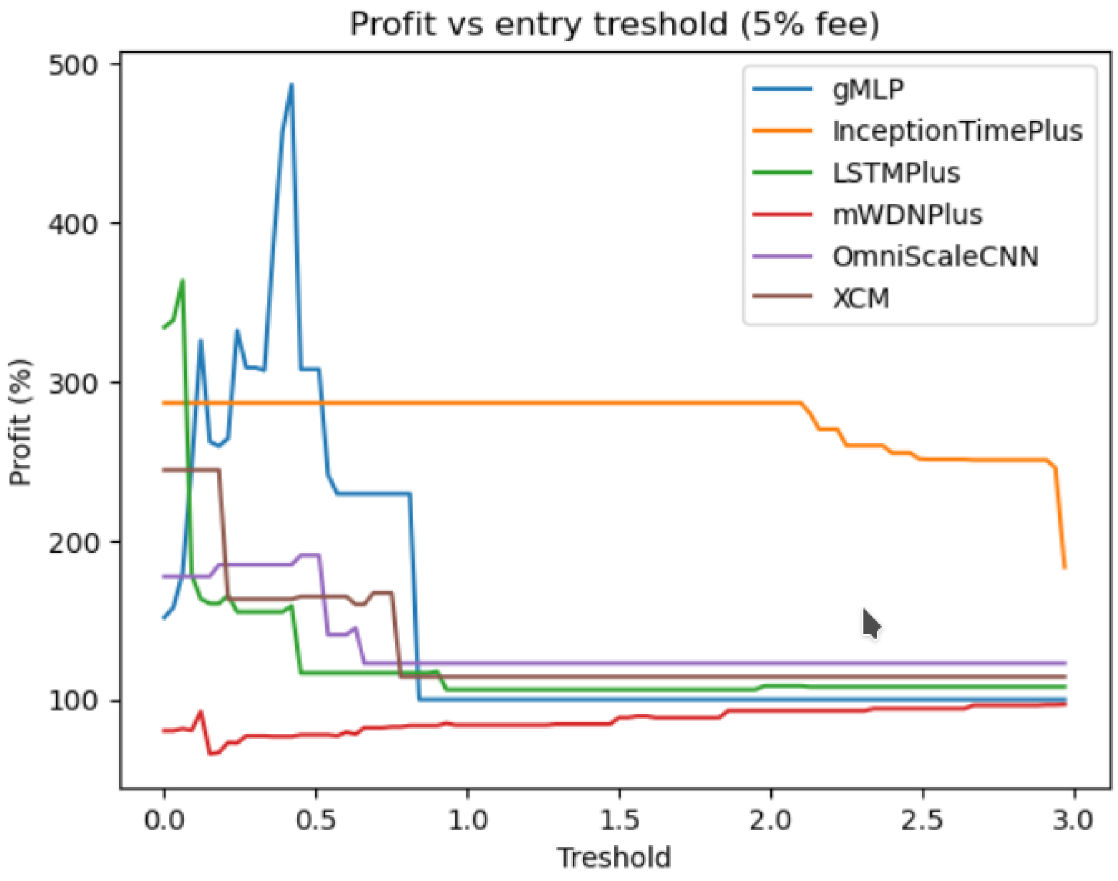


Figure 4: This plot shows the influence of the entry threshold on the resulting profit during the testing period in the case of 5% trading fees.

be interesting to test more complex entry logic as the one used now is pretty simple. Another improvement can be possibly made by finding more optimal hyperparameters.

References

[1] “Steam comunity marketplace.” `https://steamcommunity.com/market/`. Accessed: 2023-6-6.

[2] “The pushshift.io reddit api.” `https://github.com/pushshift/api/blob/beta/docs/index.rst`. Accessed: 2023-6-6.

[3] "TSAI : State-of-the-art Deep Learning library for Time Series and Sequences. ." `https://timeseriesai.github.io/tsai/`. Accessed: 2023-6-6.