The automated detection of fraudulent peer-to-peer transactions in massively multiplayer online economies

Hayley Davies - 1902055

[1] [2]

***Abstract*—Massively Multiplayer Online games are a hugely popular and successful subsection of the gaming industry. These games allow players to trade items within the game, but some players choose to buy and sell in-game items for real-world money, known as Real Money Trading. This leads to people preferring to buy from illicit sources rather than through the game itself, resulting in a loss of revenue for the developers. Additionally, this practice enables people to make money through methods such as botting, account theft, or cheating.**

***Index Terms*—Anomaly Detection, Massively Multiplayer Online, Real Money Trading**

I. Introduction

REAL-MONEY Trading (RMT) is the practice of buying and selling virtual items and currency within massively multiplayer online (MMO) video games. This practice often violates the games' Terms of Service or Code of Conduct[3][27], and as a result, players who engage in RMT are often at risk of being banned from the game. RMT has negative effects on the game's economy and community, and it is therefore discouraged by game developers and players alike.

II. Literature Review

A. Anomaly Detection for RMT

Anomaly detection is a technique used to help address the issue of RMT in MMOs[28][1]. By using computers to flag in-game transactions automatically for human review, it is possible to identify and prevent RMT activity in a way where humans don't have to analyse the all the data manually by hand. Preventing RMT helps to protect the game's economy and maintain a fair and balanced playing experience for all players. However, it is important to note that the effectiveness of this approach will depend on the specific details of the game and the methods used for detecting anomalies.

Fujita et al. propose a method for addressing the issue of RMT in MMO games, which involves identifying suspects, verifying their involvement in RMT activity, and banning their accounts[12]. They also classify RMT players into three categories:

- *Sellers* are those who sell the virtual property to players for real-world money.
- *Earners* acquire virtual property (currency and items) from non-player characters (NPCs) and real players.
- *Collectors* convey virtual property from earners to sellers.

Fujita et al. manually classified a set of players and then used an algorithm proposed by Newman and Girvan[13] to extract communities and further identify players. They ranked players based on the number of times they traded currency, the number of trades they made in total, and the total volume of currency traded.

Fujita et al. argue that RMT is harmful to both the game's economy and its players, as it is often associated with other illicit activities such as cheating, botting, and account theft. RMT can also drive away legitimate players who become frustrated with the problems it causes, and it can discourage new players from joining the game[23]. Overall, Fujita et al. believe that RMT is a serious issue that needs to be addressed to protect the integrity and health of MMO games. Han et al. and Sifa et al. back up Fujita's claims adding that "cheating in MMOs often reduces the shelf life of the game" by causing people to abandon it[14][23].

B. Types of Anomaly

Anomalous data can typically be categorized in three different ways[2][6]. These categories are:

- **Point Anomaly** where a data point is unusually out of range.
- **Contextual Anomaly** (or collective anomaly[26]) where sometimes, a data point which seems anomalous is actually within range depending on a varying factors.
- **Collective Anomaly** is where multiple data points are out of range, but when considered individually, are not out of range.

Understanding these categories can help researchers identify and address potential issues in their data. By detecting anomalies, it may be possible to uncover hidden patterns or trends and improve the accuracy and reliability of data-driven systems and models.

C. Machine Learning

Due to the nature of anomaly detection is often done with Machine Learning (ML) algorithms[19][16][33]. These algorithms are split into two categories, supervised and unsupervised[19]. Supervised methods "require a labeled training set

[1] Source code for all algorithms is availible on GitHub: https://github.com/cdgamedev/dissertation

[2] Source LaTeXis availble on Falmouth GitHub: https://github.falmouth.ac.uk/Games-Academy-Student-Work-22-23/1902055-comp3xx-dissertation

containing both normal and anomalous samples", whereas, unsupervised methods don't require training data as they assume a fraction of data points are anomalous[19].

Nassif et al. state that they "recommend that researchers conduct more research on ML studies of anomaly detection to gain more evidence on ML model performance and efficiency" following their own review of prior research. They mention that unsupervised datasets have a greater number of research papers than supervised datasets. They also identify 29 different machine learning models for detecting anomalies[17].

D. Nearest-neighbor Based Algorithms

Local Outlier Factor (LOF) works by getting each object in the dataset to "indicate its [own] degree of outlier-ness"[7].

k-Nearest Neighbor (kNN) is a supervised learning algorithm based on Nearest Neighbor and it's discussed frequently within anomaly detection and ties in with lazy learning algorithms. kNN functions by finding the K number of the nearest points and assigns the data point to a label that is most suited, this can be used for anomaly detection if the assignment is different to how the data is already labelled[9].

Lazy learning algorithms work by[30]:

- Storing all training data, and deferring processing until queries are given the required replies.
- Answering queries by combining the training data.
- After replying, the answer and any results are discarded.

Many improvements to the base kNN algorithm have been developed by other researchers. Muti-label kNN (ML-kNN) exists to provide lazy learning to problems such as text categorization or bioinformatics. This approach works by [32].

E. Graph-based Anomaly Detection

Graph-based anomaly detection algorithms "[detect] patterns (substructures) within graphs" where a "substructure is a connected subgraph in the overall graph"[18]. Noble et al. utilized their anomaly detection, Subdue, for intrusion detection, utilizing data which "contained 41 features describing the connection" and labelling them as "one of 37 different attack types" or "normal". This is a form of unsupverised learning[18]. Graph-based anomaly detection works well with large datasets and is often used for collective anomalies[11].

Davis et al. discuss the use of Yet Another Graph-based Anomaly Detection Algoritm (YAGADA). They find that in other methods of anomaly detection, single time events are detected easily, but anomalous patterns of data which are anomalous in context such as "an airport technician who regularly hangs around in the baggage handling area, or a clerk who is spending an unusually long time on their own in the cash room" would not normally be detected. They conclude that YAGADA works well for static graphs and suggest using it in "forensic analysis of graph transaction databases". They also conclude that LOF[7] is more suitable to numeric anomaly detection.

F. Autoencoders

Misra et al. focus on an autoencoder based model for detecting fraudulent transactions within the financial domain, specifically credit cards[16]. They propose a two stage method where "a lower dimension of features are extracted from the input" before "a model decides whether the transaction is fraud or not". The first stage utilizes an autoencoder and the final stage utilizes a classification algorithm. "Autoencoders are simple learning circuits which aim to transform inputs into outputs with the least possible amount of distortion"[4]. They state that having too many features can cause classification algorithms to "run poorly" and that the "data becomes very expensive [when] time complexity is concerned" and is resolved by reducing the number of features. Misra defined features or attributes as parts of a whole data point and that autoencoders can extract these features nicely on any dataset. For credit card fraud, some of these attributes are, time/amount/mode/location of transaction, a user's account number, a user's age.

Deep Autoencoding Gaussian Mixture Model (DAGMM) works by preserving "information of an input sample in a low-dimensional space" and then performs a "Gaussian Mixture Model over the learned low-dimensional space" before utilizing "a sub-network called estimation network that takes the low-dimensional input from the compression network and outputs mixture membership prediction for each sample"[33].

G. Standard Deviation

Yang et al. discuss the use of standard deviation (SD) within anomaly detection[31]. However, they note that simple datasets can cause false positives[20] and researchers misuse SD methods frequently[24]. The main issue with SD algorithms is that outliers influence the standard deviation and averages for the dataset.

To solve these issues Yang set out to develop a modified SD algorithm which could be relied on to give more accurate results. Two-stage thresholding (2T)[31] works similarly to Clever Standard Deviation (Clever SD)[8] by utilizing recursion. 2T works by recursively removing outliers one at a time and is the most accurate method of SD algorithms based on Yang's findings.

H. Anomaly Detection for Other Datasets

Bergman and Hoshen talk about anomaly detection for general data and classification of anomalies using AI. Examples, of where this is used, are for fraudulent credit transactions and detecting cyber attacks amongst others[5]. They state that "classification-based methods have dominated supervised anomaly detection", these are methods of anomaly detection which utilise a classifier trained by an ML model. They further discuss the use of the following semi-supervised methods; one-class classification and geometric-transformation classification. They make a comparison between SVMs[22], LOF[7] and DAGMM[33].

Similarly, Misra and Sadineni investigate the use of anomaly detection within Credit Card transactions[16][21].

III. Research Questions

The questions this paper sets out to answer are:

- Which anomaly detection algorithm is the most performant for peer-to-peer transactions within MMOs?
- Can most fraudulent transactions be found using anomaly detection algorithms?

By answering these, game developers working on MMO titles can easily identify and choose a method that suits the needs for their game. This will help reduce the revenue loss caused by RMT which could be millions[10].

IV. Hypotheses

A. Which anomaly detection algorithm is the most performant for peer-to-peer transactions within MMOs?

Following research conducted, for data specifically from Lost Ark, a simple method which doesn't require training data is most likely to be best. The 2T algorithm[31] could be beneficial for a dataset which has 4 features. As the data from Lost Ark requires only 4 features, a more complex algorithm likely isn't required.

B. Can most fraudulent transactions be found using anomaly detection algorithms?

Due to the fact that the dataset used cannot be labelled, it will be difficult to know for certain if the algorithms function to detect all fraudulent transactions. If a data point is out of range then an algorithm, assuming the correct settings, will be able to correctly identify all anomalous transactions. However, it could also incorrectly detect real transactions in the event that the price of a gem fluctuates for legitimate reasons. The main concern is that people will understand that spending 100,000 Gold on a Gem that shouldn't be worth that price will be flagged, and therefore, they sell several invaluable gems for fractions of that price. This will go unnoticed by any algorithm using the chosen features.

V. Computing Artefact

A. Game Background

For this research, the primary focus will be on Smilegate and Amazon Games' Fantasy MMORPG; Lost Ark[25] with a specific focus on its Gem system. Gems are collected by players during gameplay and can be sold to others using the in-game marketplace. Each Gem has different attributes, such as, Level, Name, Tier, Gem Effect, Sale Price and Sale Date.

- *Level* is a number between 1 and 10 which impact the effect of the Gem. Level n gems are created by merging 3 Level $n-1$ gems. A Level 10 gem should always cost more than a Level 1 gem as it takes 3^{10} Level 1 gems to make a single Level 10 gem.
- *Name* is a tag given to the gem and doesn't effect the gem. This means that name shouldn't affect the price.
- *Tier* is a number between 1 and 3 for all items in Lost Ark, however, gems only exist at the start of Tier 2 meaning all gems will either be tagged with Tier 2 or Tier 3. Tier 3 gems get unlocked at Item Level 1302 and offer higher effects than their Tier 2 counterparts. This means that a Level 1 Tier 2 gem should cost less than a Level 1 Tier 3 gem, however, a Level 10 Tier 2 gem is likely to cost more than a Level 1 Tier 3 gem.
- *Gem Effect* is the overall effect of the gem. This will either, increase the damage or decrease the cooldown of an ability in the game. Various gem effects will be more favourable based on the specific character build a player chooses. Favourable gem choices depend on the games meta and players often utilize a service like Maxroll[15] to get the best build for their characters class. Gem Effect can have an impact on the price of a gem, however, this data is near impossible to get without direct access to the internal marketplace database. However, gems can also be rerolled for silver which means that there shouldn't be a huge price disparity between two Level 1 or two Level 10 gems of the same tier.
- *Sale Price* is the price which a gem was sold for.
- *Sale Date* is the date which a gem sold on.

For this research, an anomaly detection algorithm would utilize 4 factors; Level, Tier, Sale Price and Sale Date. This should be enough for a basis to detect prices which are too high at any given time. It also allows us to investigate fluctuations in price of gems over time which could be due to various factors in the game, such as events which inflate the number of gems within the game through increased drop rates of gems, gem giveaways or a change in the number of players which leads to less supply/demand.

B. Proposed Algorithm

A simple algorithm to detect anomalies within a dataset range would work by calculating the average value and standard deviation of the data. These two values are used to ensure that the data is within a standard deviation threshold of the average value. In Python the algorithm could look like this:

```
import numpy as np

# detect anomalies in data
def detect_anomalies(d, t):
    # calculate mean
    m = np.mean(d)
    # calculate standard deviation
    s = np.std(d)
    # create threshold based on s
    st = s * t
    # store the anomalies
    a = []
    # for every value in data
    for i in d:
        # if point is in range
        if abs(i - m) > st:
            # add point to a
            a.append(i)
    # return all anomalies
    return a

# load a sample dataset
d = load_dataset('data.txt')
# find anomalies
a = detect_anomalies(d, 1)
# display the anomalies
print(a)
```

VI. Data Collection Methodology

A. Internal Data

The best method of gathering data, is to contact the developers directly. The developer's internal policy could impact the ability to get the raw data from them directly. They may log a lot more data than is publicly accessible via the marketplace, for example, the internal data could contain account identifying information.

B. In-game Data

Another method for data collection surrounds screenshotting pages from the Sale History of Gems from Lost Ark's Marketplace, as seen in Figure 1. Then, the screenshot is cropped to remove the Gem Name, Starting Bid and Quality fields and converted to a greyscale image along with other image manipulation techniques to ensure clear text, shown in Figure 2 (see Algorithm A-A).

Optical character recognition (OCR) is then used on the image and checked automatically for any errors (see Algorithm A-B). The data can then be randomly sampled and manually reviewed to ensure the accuracy of the data following this process. Due to the nature in which this algorithm works, unreadable data will be logged in the saved file as a "-" alongside its page and entry numbers, making the issue much quicker to rectify.

Removing the name is done as this has no bearing on the price of a gem. Removing the Starting Bid field is done because the Starting Bid is optional when adding a gem to the marketplace, it is also not indicative of RMT. Instead, for the RMT transaction to occur correctly, Gems have a Buy Now Price set to a specific value. Removing the Quality field is done since gems don't have a quality value; this is always "-".

A consideration which could also affect the price of Gems is the "Gem Effect". This is a specific ability that Gem does to impact a character's Damage or Cooldown Time on a specific move. However, the Gem Effect is optionally rerolled within the game player. Rerolling requires the use of Silver, a much easier resource to gather, so this doesn't have a huge impact on the price of the gem.

Using this data, humans can easily recognise when a sale price for a specific level/tier gem is too high compared to other gems being sold at around the same time.

Fig. 1. Auction house screenshot before image processing.

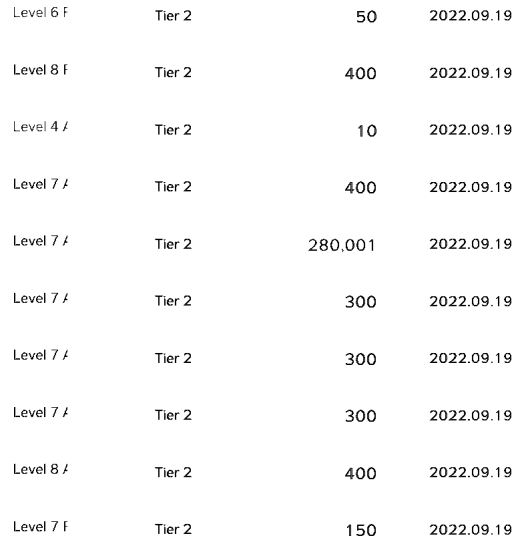

Fig. 2. Auction house screenshot after image processing.

VII. Validation and Verification

Without having a labelled dataset, it will be difficult to accurately verify anomalous transactions using an algorithm. However, by utilising different algorithms for the testing of data, it is possible to check overlapping anomalies between different algorithms. This will allow for the verification that a data point is anomalous.

A. Verifying Algorithms by Comparing Results of Different Algorithms

Comparing the results is a way in which the data can be validated. By assigning each gem with a unique value based on its location within the database we can check each gem for its occurrence as an anomaly across multiple algorithms. An example in Python would look like this:

```
# create a dictionary for anomalies
a = {}

# function to calculate results
def count_result(p):
    try:
        # if point already exists in anomalies
            increment it
        a[p] = a[p] + 1
    except:
        # create point in anomalies
        a[p] = 1

# function to check all results in the data
def check_results(d):
    # for all indexes in the data
    for i in d:
        # count the result of the index
        count_result(i)

# load the dataset
d = load_dataset('data.txt')

# check the results of the data
check_results(d)

# print the anomalies
print(a)
```

This method, however, suffers from a problem where anomalies which aren't detected by any algorithm won't be verified at all, leading to a reduced accuracy rate.

VIII. Research Methodolody

IX. Considerations

A. Legal

B. Ethical

This paper follows the Nuremberg Code[29]. The data being processed is gathered directly from the Lost Ark marketplace and doesn't contain any personally identifiable information which can be tied back to the user.

C. Professional

X. Results and Analysis

XI. Disccussion

XII. References Section

References

[1] Muhammad Aurangzeb Ahmad et al. "Mining for Gold Farmers: Automatic Detection of Deviant Players in MMOGs". In: *2009 International Conference on Computational Science and Engineering*. Vol. 4. Aug. 2009, pp. 340–345. DOI: 10.1109/CSE.2009.307.

[2] Mohiuddin Ahmed, Abdun Naser Mahmood, and Md. Rafiqul Islam. "A survey of anomaly detection techniques in financial domain". In: 55 (2016), pp. 278–288. ISSN: 0167-739X. DOI: 10.1016/j.future.2015.01.001.

[3] Amazon Games. *Amazon Games Code of Conduct*. URL. Accessed on November 11th 2022. URL: https://web.archive.org/web/20221110052745/https://www.amazon.com/gp/help/customer/display.html?nodeId=GK4QHHHAC82SQTS8.

[4] Pierre Baldi. "Autoencoders, Unsupervised Learning and Deep Architectures". In: *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*. UTLW'11. Washington, USA: JMLR.org, 2011, pp. 37–50.

[5] Liron Bergman and Yedid Hoshen. "Classification-Based Anomaly Detection for General Data". In: (May 2020). arXiv: 2005.02359 [cs.LG].

[6] Monowar H Bhuyan, Dhruba Kumar Bhattacharyya, and Jugal K Kalita. "Network anomaly detection: methods, systems and tools". In: *Ieee communications surveys & tutorials* 16.1 (2013), pp. 303–336.

[7] Markus M. Breunig et al. *LOF: identifying density-based local outliers. identifying density-based local outliers*. May 2000. DOI: 10.1145/342009.335388.

[8] Guido Buzzi-Ferraris and Flavio Manenti. "Outlier detection in large data sets". In: *Computers & chemical engineering* 35.2 (2011), pp. 388–390.

[9] T. Cover and P. Hart. "Nearest neighbor pattern classification". In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 21–27. DOI: 10.1109/TIT.1967.1053964.

[10] Julian Dibbell. "The Life of the Chinese Gold Farmer". In: *The New York Times Magazine* (June 2007).

[11] Hilmi E Egilmez and Antonio Ortega. "Spectral anomaly detection using graph-based filtering for wireless sensor networks". In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2014, pp. 1085–1089.

[12] Atsushi Fujita, Hiroshi Itsuki, and Hitoshi Matsubara. "Detecting Real Money Traders in MMORPG by Using Trading Network". In: *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2011, October 10-14, 2011, Stanford, California, USA*. Ed. by Vadim Bulitko and Mark O. Riedl. The AAAI Press, 2011. URL: http://www.aaai.org/ocs/index.php/AIIDE/AIIDE11/paper/view/4057.

[13] M. Girvan and M. E. J. Newman. "Community structure in social and biological networks". English. In: *Proceedings of the National Academy of Sciences of the United States of America* 99.12 (2002), pp. 7821–7826. ISSN: 0027-8424. DOI: 10.1073/pnas.122653799. URL: www.pnas.org/content/vol99/issue12/#APPLIED_MATHEMATICS.

[14] Mee Lan Han, Byung Il Kwak, and Huy Kang Kim. "Cheating and Detection Method in Massively Multiplayer Online Role-Playing Game: Systematic Literature Review". In: *IEEE Access* 10 (2022), pp. 49050–49063. DOI: 10.1109/ACCESS.2022.3172110.

[15] Maxroll GmbH. *Character Build Guides*. URL. Accessed on December 5th 2022. URL: https://web.archive.org/web/20221205003955/https://maxroll.gg/lost-ark/category/build-guides.

[16] Sumit Misra et al. "An Autoencoder Based Model for Detecting Fraudulent Credit Card Transaction". In: 167 (2020), pp. 254–262. ISSN: 1877-0509. DOI: 10.1016/j.procs.2020.03.219.

[17] Ali Bou Nassif et al. "Machine Learning for Anomaly Detection: A Systematic Review". In: *IEEE Access* 9 (2021), pp. 78658–78700. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3083060.

[18] Caleb C. Noble and Diane J. Cook. "Graph-Based Anomaly Detection". In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '03. Washington, D.C.: Association for Computing Machinery, 2003, pp. 631–636. ISBN: 1581137370. DOI: 10.1145/956750.956831. URL: https://doi.org/10.1145/956750.956831.

[19] Salima Omar, Asri Ngadi, and Hamid H. Jebur. "Machine Learning Techniques for Anomaly Detection: An Overview". In: 79 (), pp. 33–41. ISSN: 0975-8887. DOI: 10.5120/13715-1478.

[20] Thomas V Pollet and Leander Van Der Meij. "To remove or not to remove: the impact of outlier handling on

significance testing in testosterone data". In: *Adaptive Human Behavior and Physiology* 3.1 (2017), pp. 43–60.

[21] Praveen Kumar Sadineni. *Detection of Fraudulent Transactions in Credit Card using Machine Learning Algorithms*. 2020. DOI: 10.1109/i-smac49090.2020.9243545.

[22] Bernhard Schölkopf et al. "Support Vector Method for Novelty Detection". In: *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. Ed. by Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller. The MIT Press, 1999, pp. 582–588. URL: http://papers.nips.cc/paper/1723-support-vector-method-for-novelty-detection.

[23] Rafet Sifa et al. "Archetypal Analysis Based Anomaly Detection for Improved Storytelling in Multiplayer Online Battle Arena Games". In: *2021 Australasian Computer Science Week Multiconference*. ACSW '21. Dunedin, New Zealand: Association for Computing Machinery, 2021. ISBN: 9781450389563. DOI: 10.1145/3437378.3442690. URL: https://doi.org/10.1145/3437378.3442690.

[24] Joseph P Simmons, Leif D Nelson, and Uri Simonsohn. "False-positive psychology: undisclosed flexibility in data collection and analysis allows presenting anything as significant." In: (2016).

[25] Amazon Games Smilegate Amazon Game Studios. *Lost Ark - Free to Play MMO Action RPG*. Digital Game. 2019. URL: https://www.playlostark.com/en-us.

[26] Xiuyao Song et al. "Conditional anomaly detection". In: *IEEE Transactions on knowledge and Data Engineering* 19.5 (2007), pp. 631–645.

[27] Square Enix. *Square Enix Final Fantasy XI Online Real Money Trading (RMT)*. URL. Accessed on November 11th 2022. URL: https://web.archive.org/web/20221110062944/https://support.na.square-enix.com/faqarticle.php?id=20&kid=12802.

[28] Jianrong Tao et al. "MVAN: Multi-view Attention Networks for Real Money Trading Detection in Online Games". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, July 2019, pp. 2536–2546. ISBN: 9781450362016. DOI: 10.1145/3292500.3330687. URL: https://doi.org/10.1145/3292500.3330687.

[29] "The Nuremberg Code (1947)". In: *BMJ* 313.7070 (1996). Ed. by, p. 1448. DOI: 10.1136/bmj.313.7070.1448. eprint: https://www.bmj.com/content/313/7070/1448.1. URL: https://www.bmj.com/content/313/7070/1448.1.

[30] Dietrich Wettschereck, David W Aha, and Takao Mohri. "A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms". In: *Artificial Intelligence Review* 11.1 (1997), pp. 273–314.

[31] Jiawei Yang, Susanto Rahardja, and Pasi Fränti. "Outlier Detection: How to Threshold Outlier Scores?" In: *Proceedings of the International Conference on Artificial Intelligence, Information Processing and Cloud Computing*. AIIPCC '19. Sanya, China: Association for Computing Machinery, 2019. ISBN: 9781450376334. DOI: 10.1145/3371425.3371427. URL: https://doi.org/10.1145/3371425.3371427.

[32] Min-Ling Zhang and Zhi-Hua Zhou. "A k-nearest neighbor based algorithm for multi-label classification". In: *2005 IEEE international conference on granular computing*. Vol. 2. IEEE. 2005, pp. 718–721.

[33] Bo Zong et al. "Deep Autoencoding Gaussian Mixture Model for Unsupervised Anomaly Detection". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=BJJLHbb0-.

Appendix A
Data Collection Code Samples

A. image-formatter.py to turn auction house screenshots into OCR readable images.

```
import os
from PIL import Image, ImageEnhance, ImageOps, ImageFilter
import numpy as np

# input and output of the images
INPUT_LOCATION = "original-screenshots"
OUTPUT_LOCATION = "output-screenshots"

# the crop locations for the images (x pos, width)
IMAGE_CROPS = [(1405, 25), (700, 500), (222, 210), (0, 130)]
# output width of the images (retain original height)
IMAGE_OUTPUT_WIDTH = 800

# crop a column out of an image
def image_crop_column(img, crop):
    # get the starting pos
    crop_x, crop_width = crop

    # convert the image to an array
    img_arr = np.array(img)
    # move the data from the crop_width to the current x value
    img_arr[:, crop_x:img.width-crop_width] = img_arr[:, crop_x+crop_width:img.width]
    # convert the array back to an image and return the image
    crop = Image.fromarray(img_arr)
    return crop

# resize the image
def image_resize(img):
    # crop the image to be size [WIDTH, HEIGHT] and return
    resize = img.crop((0, 0, IMAGE_OUTPUT_WIDTH, img.height))
    return resize

# apply effects to the image
def apply_effect(img):
    # reduce the contrast
    enhancer = ImageEnhance.Contrast(img)
    output = enhancer.enhance(0.8)

    # reduce the bits per pixel to 1
```

```
    output = ImageOps.posterize(output, 1)

    # enhance the edges
    output = output.filter(ImageFilter.
        EDGE_ENHANCE_MORE)

    # invert the images' colours
    output = ImageOps.invert(output)

    # return the final image
    return output

# for all files in the directory
for file in os.listdir(INPUT_LOCATION):
    # escape clause if they aren't pngs, continue
        to next cycle
    if not file.endswith(".png"):
        continue
    # open the image and convert it to greyscale
    img = Image.open(r"{0}\\{1}".format(
        INPUT_LOCATION, file)).convert('L')
    # crop out the columns specified
    for i in IMAGE_CROPS:
        img = image_crop_column(img, i)
    # resize the image after cropping
    img = image_resize(img)
    # apply the desired effects to the image
    img = apply_effect(img)
    # save the image
    img.save(r"{0}\\{1}".format(OUTPUT_LOCATION,
        file))
    print(file)

print("\n\n!! COMPLETED !!")
```

B. image-processor.py to turn images into csv files based on the dataset date.

```
import pytesseract
import re

# important consts for program to know
IMAGE_COUNT = 165
DATASET_NAME = "Dataset 1 - 08.09.2022.csv"
INPUT_DATA_LOCATION = "output-screenshots"

# tesseract specific configuration
pytesseract.pytesseract.tesseract_cmd = r"C:\
    Program Files\Tesseract-OCR\tesseract.exe"
custom_oem_psm_config = r'''
-c tessedit_char_whitelist="01234567890TierLvl,.
    "
-c preserve_interword_spaces=1x1
--oem 3 --psm 6'''

# ensure user knows they are about to overwrite
    all data from previously
i = input("Continuing will clear existing data.
    Would you like to continue? [Y/N] ")
if (i == "Y"):
    # clear all data
    output = open(DATASET_NAME, "w+")
    output.write("Page,EntryNo,Level,Tier,Cost,
        Date\n")
    output.close()
else:
    # exit program
    log = "Input not recognised. Exiting program.
        "
    if (i == "N"):
        log = "Exiting program."

    print(log)
    exit()


# gem class to store info about gems
class Gem():

    # when creating a new class
    def __init__(self, page = "-", entry_no = "-"
        , level = "-", tier= "-", cost="-", date
        ="-"):
        self.page = page
        self.entry_no = entry_no
        self.level = level
        self.tier = tier
        self.cost = cost
        self.date = date

    # convert gem stucture to a string
    def __str__(self) -> str:
        return r"{0},{1},{2},{3},{4},{5}".format(
            self.id, self.page, self.entry_no,
            self.level, self.tier, self.cost,
            self.date)

# get the data in rows
def get_data_rows(ocr):
    # the minimum characters for the row to be
        counted
    min_row_characters = 10

    # split with regex of new line
    rows = re.split(r'\n+', ocr)

    # create a new array for cleaner rows
    filtered_rows = []

    # only get rows which are within the
        threshold
    for row in rows:
        #print(len(row))
        if (len(row) > min_row_characters):
            filtered_rows.append(row)

    # return the cleaned rows
    return filtered_rows

# function to get the gem data
# passes through the entry number
# passes through the row to get the data from
def get_gem_data(number, row):
    # get the row data by splitting entries with
        >=2 space characters
    row = re.split(r"\s{2,}", row)

    filter_level  = r'Level'
    filter_tier = r'er \d'
    filter_date = r'\d\d\d\d.\d\d.\d\d'
    filter_cost = '\d{1,3}[\,.]{1}\d{1,3}|\d{1,3}
        '

    # create a new gem using the page number and
        index number
    gem = Gem(page, number)

    # for all the cells in the row
    for cell in row:
        cell = cell.replace(',', '')

        # check through cell with each filter and
            process accordingly
        if re.search(filter_level, cell):
            gem.level = handle_level(cell)
        elif re.search(filter_tier, cell):
            gem.tier = handle_tier(cell)
        elif re.search(filter_date, cell):
            gem.date = handle_date(cell)
        elif re.search(filter_cost, cell):
            gem.cost = handle_cost(cell)
        else:
```

```
97                 print(r"Data issue: {0}".format(cell)
                       )
98
99         return str(gem)
100
101    def handle_level(level):
102        number = re.search('\d+', level).group(0)
103        if (int(number) > 10):
104            number = number[0]
105        level = "Level " + number
106
107        return level
108
109    def handle_tier(tier):
110        return tier
111
112    def handle_date(date):
113        return date
114
115    def handle_cost(cost):
116        cost = re.sub('\D', '', cost)
117        return cost
118
119    def parse_string_data(ocr):
120        data_rows = get_data_rows(ocr)
121        gem_list = []
122
123        for i in range(len(data_rows)):
124            row = data_rows[i]
125            gem_list.append(get_gem_data(i + 1, row))
126
127        output = open(DATASET_NAME, "a")
128
129        output.write('\n'.join(gem_list))
130        output.write('\n')
131
132        output.close()
133
134        return "[SUCCESS] {0}\n".format(page)
135
136    for page in range(1, IMAGE_COUNT + 1):
137        page_id = r"page_{0}.png".format(page)
138        ocr = pytesseract.image_to_string(r"{0}\{1}".
                  format(INPUT_DATA_LOCATION, page_id),
                  config=custom_oem_psm_config)
139
140        text_data = parse_string_data(ocr)
141        print(text_data)
142
143    output.close()
```