FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Value estimation of NFT collectibles
Student:	Daniel Bukač
Supervisor:	Ing. Martin Votruba
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

Non-Fungible Tokens (NFT) are crypto assets stored on the blockchain. Each NFT has its own unique properties, therefore they are not interchangeable. One of the most common use cases for NFTs are collections of tokens containing digital art. The goal of this work is to estimate the value of a given NFT.

- Describe NFT tokens on Ethereum blockchain and digital art market. Define keywords that are being used in the space.
- Research methods of value estimation of NFT.
- Propose a method using a ML technique that can estimate the value of a given NFT. This estimate should help identify promising opportunities for trading.
- Implement the proposed method using publicly available data.
- Evaluate estimations of the implemented method, discuss results with emphasis on possible use cases.

Electronically approved by Ing. Karel Klouda, Ph.D. on 25 January 2022 in Prague.

Bachelor's thesis

VALUE ESTIMATION OF NFT COLLECTIBLES

Daniel Bukač

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Ing. Martin Votruba
May 12, 2022

Contents

List of Figures

List of Tables

First of all, I would like to thank my supervisor Ing. Martin Votruba not only for his expertise but also for being friendly, positive, and tolerant. Many thanks also go to my family and friends for always supporting me during my studies. It was a great help to me.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 12, 2022 .....

Abstrakt

Práce se zaměřuje na odhad ceny nezaměnitelných tokenů, které představují záznam o vlastnictví digitálního sběratelského umění na Ethereum blockchainu. Nejprve jsou představeny metody strojového učení vhodné k řešení problému a následně jsou i aplikovány. Výstupem této práce je model strojového učení schopný predikovat hodnotu těchto tokenů. Napříč prací je zdůrazňováno využití tohoto modelu k efektivnímu obchodování těchto aktiv. Navržený model docílil průměrné absolutní procentuální chyby menší než 10% a zároveň byl schopný identifikovat potenciálně výdělečné obchody.

Klíčová slova NFT, Blockchain, Digitální umění, Hluboké učení, Neuronová síť

Abstract

This thesis focuses on estimating value of non-fungible tokens which represent the record of ownership of collectible digital art asset on the Ethereum blockchain. Suitable machine learning techniques are described and subsequently applied to solve this problem. The outcome of this work is a proposal for a machine learning model capable of predicting the value of these tokens. Throughout this thesis, the application of this estimator to effective trading of these assets is emphasized. The proposed model was capable of predicting the token value with absolute percentage error less than 10% and also identifying potentially profitable trades.

Keywords NFT, Blockchain, Digital art, Deep learning, Neural network

Acronyms

NFT	Non-fungible token
EVM	Ethereum virtual machine
EOA	Externally owned account
ML	Machine learning
ANN	Artificial neural network
MLP	Multilayer perceptron
MSE	Mean squared error
MAE	Mean absolute error
SGD	Stochastic gradient descent
API	Application programming interface
PCA	Principal component analysis
MAPE	Mean absolute percentage error

Introduction

Non-fungible tokens (NFTs) are a means of representing unique assets on a blockchain. Similarly to cryptocurrency tokens, the ownership and complete history of transactions is persistently stored on a distributed ledger; therefore, the owner can prove the ownership of the asset anytime and without any doubts. The key difference from other cryptocurrency tokens is the fact that each non-fungible token is unique, and thus it cannot be interchanged for another non-fungible token. On the contrary, cryptocurrency tokens should be indistinguishable from each other in order to trade them as a currency.

In recent months, we have witnessed a massive increase in attention to NFTs on the Internet. Digital artists, cryptocurrency enthusiasts, pop culture celebrities, or even international corporations have been trying to catch up with this phenomenon by either purchasing or creating their own NFT. The dynamics of this industry has attracted both investors and skilled workers in the hope of forming what the Internet will look like in the next years.

One of the first and most common uses of this technology is digital collectibles. These are collections of NFTs, where each token contains data of some image or video associated with it. Usually, each token contains slightly different visual content, but as a whole, it can be recognized as a collection of digital art.

These collections have gained popularity among long-term collectors and also short-term traders. Both these groups have been trying to speculate on tokens' values. Since each token in the collection is unique, its value is also different. This poses the question of whether it is possible to estimate the value algorithmically. If so, it would help all investors make better decisions and even automate trading. Furthermore, if the value of an NFT was estimated, decentralized finance platforms would be able to accept the NFT as a collateral against a loan.

The aim of this thesis is to propose, implement, and evaluate a machine learning model to estimate the value of a given NFT token with an emphasis on trading. The foundation of this work will consist of existing approaches to estimation problems in other domains. The model for the NFT domain will be based on the principles of these approaches.

In this thesis, we will focus only on NFTs on the Ethereum blockchain. In the practical part of this work, we will work with publicly available data from a selected collection. However, the core principles should be transferable to any other NFT collection.

Chapter 1

Theoretical foundation

The key concepts used in this work are introduced in this chapter. The blockchain and NFT related principles are explained in the beginning; then the machine learning concepts and models that can be used to tackle the issue of estimating the NFT value are described.

1.1 Ethereum

Ethereum was proposed in a paper known as the Ethereum whitepaper by Vitalik Buterin in 2014 [1]. In this work, he discussed existing concepts for decentralized digital currencies, such as Bitcoin. According to Buterin, Bitcoin has implemented two radical new concepts. The first was the concept of decentralized digital currency, which has been around since the 1980s [2], but these ideas were impracticable in the context of decentralization [1]. The second was the public consensus mechanism, implemented with the use of a proof-of-work algorithm, which makes attempts at malicious interventions almost impossible [3].

Bitcoin implemented a scripting language; however, this implementation had several limitations, such as the lack of Turing-completeness [1]. Ethereum aims to extend the concepts of Bitcoin with the ability to create consensus-based applications that can be run and replicated on the network. This was achieved by providing a Turing-complete programming language for the Ethereum blockchain [1, 4]. Ethereum can be seen as a ”very specialised version of a cryptographically secure, transaction-based state machine”[5].

On Ethereum, the performing entity is called account. The account is identified by its public key. If the account is operated by some user, the account's type is called Externally Owned Account (EOA) and is associated with cryptographic private key as well. The purpose of the private key is to sign transactions and messages to prove that the transaction was approved by the sender [6, 7]. Transactions between accounts represent the state change of the Ethereum blockchain. The transactions are then grouped into blocks and these blocks are chained together with a cryptographic hash function [1, 5].

1.1.1 Smart contracts

Smart contracts are the second type of Ethereum account in which the account is not controlled by the user, but its behavior is entirely controlled by ”immutable computer programs that run deterministically in the context of an Ethereum Virtual Machine (EVM)” [7]. In addition to that, smart contracts can interact with blockhain entities in the same manner as EOAs would. This feature provides developers with the ability to define a set of actions on the blockchain with which anyone can interact and which cannot be removed or modified on the blockchain. [8].

1.1.2 Non-fungible tokens

NFTs are special types of tokens on the Ethereum blockchain. Each token is unique and cannot be interchanged for another NFT. The purpose of this concept is to implement a way to represent the ownership of unique assets on the blockchain, where these assets can be further traded.

The standard for non-fungible tokens (NFT) was proposed in EIP-721 in 2018. This standard provides the ERC-721 smart contract interface, which essentially defines all NFTs smart contracts on Ethereum. Each token in this contract has its unique identifier. Together with the contract address, it should form a globally unique identifier throughout the Ethereum blockchain.

Another important concept implemented in the token standard is the extension to handle metadata. Each token can be associated with its metadata information. Name and symbol metadata are required, but the metadata field usually contains more information, e.g. data associated with digital art on this token. Storing a lot of data on the blockchain would be costful; therefore a mechanism to associate NFT with a web URL was provided [9].

1.2 Collectibles

The concept of collections of NFTs containing digital art was promoted for the first time by the CryptoPunks collection in 2017, but the principles described are relevant for most of the currently released collections. CryptoPunks contained 10,000 tokens containing recongizable pixel art images “inspired by the 1970s’ London punk scene and the dystopian grit of cyberpunk”[10]. Images were generated algorithmically and randomly assigned to specific assets. Each of the illustrations had its specific **traits** such as accessory or gender. These traits are also stored in the token metadata (CryptoPunks did not use the NFT standard 1.1.2, but a modified former fungible token standard).

The tokens are often ranked by **rarity** of their traits to measure how many tokens with similar traits there are in the collection. Rarer tokens are usually more expensive than common ones. Tokens that are not that interesting to investors are usually traded at a price close to **floor price**, which is the price of the cheapest token sale offer in the collection. This metric is provided to traders by various analytical platforms, but unfortunately there is no rigorous definition of rarity functions, so the ranks may differ [11].

Code listing 1.1 Example traits

```
{
  "attributes": [
    {
      "trait_type": "Type",
      "value": "Human"
    },
    {
      "trait_type": "Hair",
      "value": "Blue Spiky"
    },
    {
      "trait_type": "Clothing",
      "value": "Black Yukata"
    },
    {
      "trait_type": "Eyes",
      "value": "Suspicious"
    },
    {
      "trait_type": "Mouth",
      "value": "Grass"
    },
    {
      "trait_type": "Background",
      "value": "Off White D"
    }
  ]
}
```

1.3 Machine learning

Machine learning (ML) is a discipline of computer science that focuses on the ability of algorithms to perform in situations for which they were not explicitly programmed [12]. This behavior can be achieved by improving the performance of the algorithm by learning from its previous experience with the problem [13]. Therefore, ML algorithms are suitable for solving problems that cannot be effectively solved with traditional approaches because they are too complex, dynamically change, or the optimal algorithm is unknown [14], such as computer vision, speech recognition, or natural language processing.

Machine learning systems can be divided into three groups depending on the type of experience they learn from.

- **Supervised learning**: The training inputs for supervised ML systems are paired with its desired outputs. During the training period, the model deduces a generalized function to map these inputs to their labeled outputs, which can be later applied to previously unseen inputs.
- **Unsupervised learning**: Unsupervised models do not have the information about the desired output for their inputs, so they have to extract information and structure from unlabeled data.

- **Reinforcement learning**: The ML system interacts with a dynamic environment and learns from the reward or punishment for its actions.

1.3.1 Supervised learning

Supervised learning is a ML technique for learning a functional mapping from input vectors to output labels. This is achieved by learning from a portion of the data with the correct output values called **training data**. The mapping can then be applied to the rest of the inputs to predict the output values.

Depending on the type of output labels, supervised learning tasks can be divided into **classification** and **regression**. Classification output labels are discrete values, whereas regression labels are continuous.

1.3.2 Ensemble methods

Ensemble learning is an ML technique that takes multiple individual models and combines their capabilities. This may lead to better prediction accuracy than each model would have alone. Ensemble systems are convenient in situations where the problem is too complex for one learner, but can be solved by a suitable combination of learners [15].

The relevant ensemble algorithms for this work are the following:

- **Bootstrap aggregating**: The training dataset is split into n datasets of the same size by randomly selecting samples with repetition. The model provided for bootsrap aggregation is then trained on each of the datasets, and the final prediction is either a majority vote of the models in the case of classification or an arithmetical average of the models' predictions in the case of regression.
- **Gradient boosting**: A technique used in both regression and classification tasks. Gradient-boosting iteratively chooses a new estimator to correct residual errors made by the previous estimator [16]. The iterative process can be regarded as a gradient descent algorithm.
- **Ensemble averaging**: A process of combining predictions of multiple models that contribute equally to the final result.
- **Stacked generalization**: Improvement of ensemble averaging in which models do not contribute equally, but their weights are determined by a new combiner model. The final prediction is made by the chosen combiner model, and its inputs are assembled from the predictions of the underlying ensemble models. Stacking generalization usually outperforms each of the underlying models [17].

1.3.3 Imbalanced datasets

Most of the ML algorithms require an equal class distribution of data classes to perform well. Unfortunately, this requirement is not fulfilled for many real-world datasets. Unbalanced data in classification problems can be detected quite easily by checking the distribution of training

labels. In contrast, regression problems require some level of knowledge of the data domain, since the training data contain infinitely many labels to find the imbalance in the dataset.

Researchers explored methods for handling data imbalance in classification problems, whereas regression over imbalanced data is not examined that well [18]. Some approaches adapt the SMOTE algorithm modified for regression problems [19]. This algorithm decreases the number of more frequent data classes (**undersampling**) and synthetically increases the number of infrequent data classes (**oversampling**).

The infinite label space can be divided into a finite number of bins (classes), and then the methods used to handle the imbalance in the classification datasets can be applied. Reweighting of classes by their inverse frequency can be applied, but this approach usually does not perform well on real-world data [20].

Yet another method was proposed [21] to handle imbalances in classification neural networks by balancing mini-batches (described in 1.5.3 when training the network. With conventional mini-batch training, all training samples are used exactly once in a training epoch, whereas this balancing approach allows for overlapping selections of minority data class. The authors of the method were able to perform an experiment in which their approach achieved better results than undersampling and oversampling the data.

1.3.4 Principal component analysis

Principal component analysis (PCA) is a popular technique used to reduce dimensionality, explore, and visualize data. This method can simplify and extract the most important features from the observed data.

PCA computes linear combinations of the original variables called **principal components**. The first principal component should preserve the largest variance in the dataset, and thus probably lose less information from the dataset than other projections would [22]. The following principal components are computed under the constraint of being orthogonal to the previous ones and containing the most of the remaining inertia in the dataset [23].

1.4 Decision trees

Decision trees are a popular ML model that is suitable for both classification and regression tasks. They contain a single tree-like model, which recursively splits the data according to some testing conditions and stores the decision labels in its leaves.

At each node of the tree, the data are divided into branches and pushed downward in the tree. This step is repeated recursively until a stopping condition is satisfied. The construction of an optimal decision tree is an NP-complete problem [24], therefore, various different greedy algorithms are used to split the data. In each iteration, the most appropriate split is chosen. The quality of the partitioning is typically measured with one of the following metrics:

- **Information gain**: based on the concept of a random variable that represents the impurity of the data called **entropy** H. Variables $p_1, \ldots, p_k$ represent the percentual distribution of

k classes in the set T.

$$H(T) = -\sum_{i=1}^{n} p_i \log p_i \tag{1.1}$$

The information gain IG in the context of decision trees is computed as the entropy of the parent minus the weighted sum of its children's conditional entropies given the value of attribute a and it quantifies which attribute provides the maximum gain of information.

$$IG(T, a) = H(T) - H(T|a) \tag{1.2}$$

- **Gini index**: measures the probability that a randomly added attribute will be classified incorrectly. When all datapoints in the created data partition belong to the same category, the value of the Gini index is 0, indicating that the maximum information gain can be obtained. On the contrary, if the samples have a uniform distribution, the Gini index reaches its maximum value, signaling that the data do not contain useful information [25]. Gini index GI for a set T with k distinct values is calculated as:

$$GI(T) = 1 - \sum_{1}^{k} p_i^2 = \sum_{1}^{k} p_i(1 - p_i) \tag{1.3}$$

Regression problems can be handled in the same way as classification with the CART algorithm [26]. The only difference is in the metrics for partitioning the data, for which CART minimizes the sum of squared residuals at the nodes of the tree [27].

Decision trees combined with the bootstrap aggregation technique 1.3.2 are called random forests. Another popular ensemble method applied to decision trees is boosting, which usually outperforms random forests [28].

1.5 Neural networks

Artificial neural network (ANN) is a general-purpose machine learning model. It can solve both classification and regression problems. Its architecture was inspired by biological neurons and the way they are connected in human brains. This idea has been around for quite a long time. In 1943, the first model approximating the working of animal brains was published. In this study, Pitts and McCulloch proposed the idea of an artificial neuron with binary activation, multiple inputs, and a single output and showed that a model based on these artificial neurons can compute any logical proposition [29, 30].

1.5.1 Perceptron

Perceptron is the simplest implementation of the ANN architecture that can be used for binary classification tasks. It consists of a single artificial neuron with n numerical inputs $x_1, \ldots, x_n$ and a single output y. The neuron output is the result of applying **activation-function** f to the sum of each input multiplied by its corresponding weight w_i and the bias b added.

$$y = f\left(\sum_{1}^{n} w_i x_i + b\right) = w^t x + b \tag{1.4}$$

1.5.2 Multilayer perceptron

Multilayer perceptron (MLP) is a type of feedforward ANN. In feedforward ANNs connections between underlying neurons never form a cycle; thus, the information is transported in only one direction. This architecture compensates for the limitations of a single perceptron by grouping artificial neurons into layers. The output of neurons in one layer assembles the input of the next layer. MLP consists of at least three layers: the input layer, one or more hidden layers, and the output layer. The MLP layers are fully connected, so each pair of neurons in the adjacent layers is connected.

Cybenko proved in the universal approximation theorem [31] that any feedforward ANN with one hidden layer, nonlinear activations and a finite number of neurons is capable of approximating any continuous function on a compact subset of $\mathbb{R}$. Thus, we know that there exists a large MLP capable of representing any function we try to learn. However, the theorem does not guarantee that the MLP will be able to learn this function [32].

MLP use nonlinear activation functions, to introduce non-linearity to the model. If the activation functions were linear, the whole model would remain linear, because chaining of multiple linear transformations produces a linear transformation again. Therefore, w

Popular activation functions are:

- **tanh**: The hyperbolic tangent function is a sigmoid curve with the output range of $[-1, 1]$ and the layers tend to be centered around 0 at the beginning of training [33]. The hyperbolic tangent function is defined [34] as:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.5}$$

- **ReLU**: Recitified Linear Unit (ReLU) has become the most successful and for many frameworks the default activation function [35]. The function is defined as follows:

$$f(x) = max(0, x) \tag{1.6}$$

  ReLU is popular even though it cannot be differentiable for $x = 0$ and the gradient descent algorithm used for the training of ANNs requires the activation function to be differentiable.

ANN with more than 3 hidden layers is called **deep neural network**. In practice, ANNs with more hidden layers are preferred because they can exponentially reduce the number of neurons required to generalize the problem [36]. The deep architecture is suitable if we believe that the learning problem is composed of a variation of more simple factors [32].

1.5.3 Learning

The goal of learning ANNs is to minimize the value of **loss function**, which represents the error of the values predicted by the model. The selection of the right loss function for the problem is crucial. Typical loss functions used for regression problems are the following:

- **Mean squared error (MSE)** - measures the squared difference of the predicted values $\hat{Y}$ and observed values of the predicted variable Y from n samples. MSE is more sensitive to

larger errors than MAE because of squaring.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \widehat{Y}_i\right)^2 \tag{1.7}$$

- **Mean absolute error (MAE)** - measures the arithmetic average of absolute errors of predictions y and true values x from n samples.

$$MAE = \frac{\sum_{i=1}^{n}(x_i - y_i)}{n} \tag{1.8}$$

- **Mean absolute percentage error (MAPE)** - measures the average absolute percentage error of predictions y and the true values x from n samples. MAPE has several practical drawbacks [37], such as asymmetry [38], and it cannot be used when there are zero values in the real values. However, according to studies reviewed in [39], it is still one of the most widely used metrics for forecasting in business and organization, due to its good interpretability. MAPE is defined as:

$$MAPE = \frac{1}{n} * \sum_{i=1}^{n}\left|\frac{x_i - y_i}{y_i}\right| \tag{1.9}$$

- **Huber loss** - a robust regression loss function that is less sensitive to outliers, because the function is quadratic for values smaller than δ and linear for values larger than δ [40].

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta, \\ \delta \cdot \left(|a| - \frac{1}{2}\delta\right), & \text{otherwise.} \end{cases} \tag{1.10}$$

The development of ANNs had stalled for several years because it was not clear how to train MLPs. This changed in 1986 when the **backpropagation** algorithm was introduced. The idea behind this is that for each training instance, a prediction is made by passing the input data through all layers of the network. The prediction error is then measured with the selected loss function. The **gradient descent** algorithm is used to find a set of network weights that minimize the loss function. The steepest descent direction of the loss function is propagated from the output layer recursively towards the input layer. The descent direction is obtained by effectively computing the partial derivations of the error by applying the chain rule, which is essentially the application of the reverse automatic differentiation algorithm [41]. As a result, the activation function of neurons should be differentiable (or at least in most of its domain).

Computing the gradient for all training data would be impractical, hence a modification of the gradient descent algorithm was proposed. Stochastic gradient descent (SGD) is an approximation of gradient descent calculated only with a single sample or alternatively with n samples called SGD.

1.5.4 Overfitting

Overfitting is a common issue in ML that causes poor model performance. This phenomenon can be observed in situations where the model fits the training data very well but cannot generalize for unseen data, and thus performs poorly on validation or testing data. Let us examine how this problem can be addressed in the context of ANNs.

One strategy to prevent overfitting is the **early-stopping** technique. This is a straightforward method, which keeps track of the validation error during the training period, and when the validation error does not improve sufficiently or not at all, the training is stopped, and the best parameters are returned.

Another way to address this problem is called **dropout**. The key idea behind this method is to randomly choose some units of hidden layers with probability p and leave them and all their connections out of the network for the current training step. This also means that with each training step, a new ANN architecture is created. Effectively, this is a way to combine exponentially many ANNs [42].

Regularization techniques can also be used to avoid overfitting. "Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error" [43]. This is often achieved by adding a parameter norm penalty $\Omega(\theta)$ to the model cost function E. In the context of ANN, the modified cost function $\hat{E}$ can be expressed as:

$$\hat{E}(\theta,\ X,\ y)\ =\ E(\theta,\ X,\ y)\ +\ \alpha\Omega(\theta) \tag{1.11}$$

where α is a normalization hyperparameter and θ is a vector containing the ANN's weights and the unregularized parameters. **L_2 regularization**, also known as Ridge regression, is a method, that makes the model keep its values as small as possible [44] by adding a penalty:

$$\Omega(\theta)\ \ =\ \frac{1}{2}\ \|\theta\|_2^2 \tag{1.12}$$

where $\|\cdot\|_2$ denotes **Euclidean norm**. Applying the L_2 regularization makes the model "perceive the input X as having higher variance, which makes it shrink the weights on features whose covariance with the output target is low compared to this added variance" [45].

Furthermore, L_2 regularization can be constrained by a fixed constant c upper bound. This method is called **max-norm** regularization, because it limits the maximum norm of any weight to c [42].

1.5.5 Deep embedding

Deep embedding is a method of encoding categorical features into a continuous trainable dense vector representation of this feature. This type of encoding does not only produce low-dimensional embeddings, which is very often desired, but during model training, embeddings of similar categories should become proximate in the embedding space [46]. This approach can be successfully applied to natural language processing to create meaningful word embeddings [47].

1.5.6 Deep and wide learning

In 2016 a team of Google researchers introduced [48] a new concept of Wide & Deep learning, which was then implemented and evaluated in the recommendation system for the Google Play application store. The main concept was to combine the generalization abilities of deep neural networks with the memorization provided by wide linear models.

The goal of memorization in this framework is to learn the correlation between the historical occurrences of the features and the resulting value. Memorization is often used in recommendation systems because it can learn which combination of features users like.

On the other hand, generalization aims at learning the correlations between unseen or rare combinations of features. In the architecture of the proposed model 1.1 for the Google Play store, the component responsible for deep learning contained deep embedding layers 1.5.5. The model was able to generalize by matching items that are proximate in the embedding space, which improved the diversity of its recommendations.

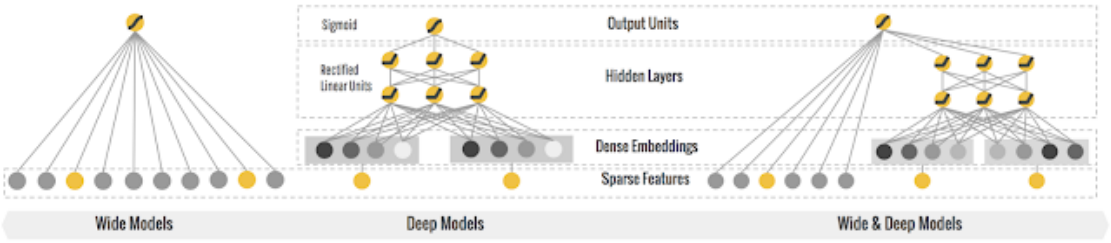

Figure 1.1 Deep & Wide architecture [49]

Chapter 2

Analysis

This chapter analyzes the objective of this thesis and the context of the NFT domain in more detail. Furthermore, existing methods of value estimation are discussed. The end of the chapter is dedicated to the methodology for the practical implementation of the proposed solution.

2.1 Objective and domain

The goal of this thesis is to propose a machine learning model capable of estimating the price for which the NFT will be sold. This could help traders identify underpriced assets or sell their tokens for a reasonable price.

Since the NFT space is fairly new, there is a lack of research, especially in the ML field. This thesis utilizes general machine learning concepts that are suitable for this type of problem. The target variable is continuous, so a regression ML algorithm should be used. In order to estimate the price, we can use historical transfers labeled with the actual price, and this leads to the process of supervised learning.

2.2 Existing approaches

The NFT area is new and has not been explored by researchers in detail yet; especially in the context of ML.

In an article [50] the concept of token valuation was introduced. It explained why conventional financial estimation methods are inappropriate and ML techniques perform better, which corresponds to observations made by another anonymous author [51].

The estimator proposed in the article is a gradient-boosted tree, combined with a non-linear transformation function implemented like a neural network, that creates dense embeddings of categorical traits of NFTs.

But the author of the article [50] had to keep some of the details of the implementation secret. The estimations were evaluated on the CryptoPunks collection, which has a very specific position in the NFT space, because it popularized NFT collectibles (as discussed in section 1.2).

Articles [50, 51] should be perceived more as an inspiration than as a solid foundation.

2.3 Methodology

Concepts introduced in this thesis will be tested and applied to a single selected collection. For these purposes, the Azuki collection was chosen because it was one of the most successful recently introduced NFT collections. Since its creation in January 2022, its floor price1.2 has risen approximately ten times. Although there are collections with a longer history, more trades, or higher prices, the Azuki collection is more representative of the current state of the NFT market.

The data used in this thesis are time-dependent; therefore, it is crucial to treat the data during training chronologically, because otherwise unknown information at that time could be leaked to the model.

Chapter 3

Implementation and design

This chapter analyzes the technical aspects of the proposed implementation. It starts with a detailed description of the models that were used for the final solution. The accuracy of each model is benchmarked. At the end of the chapter, the proposed solution combining all of the previously described models is described.

3.1 Datasets and preprocessing

A dataset containing a history of transfers with their prices and token traits is needed to train and evaluate a supervised ML model. Data retrieval and processing can be done separately for the history of transfers and tokens metadata. The final data set consists of these two data frames joined by the *asset_id* attribute.

3.1.1 Sales

Sale occurs when an NFT owner successfully sells his asset to someone else for an agreed price, usually via a specialized marketplace for NFTs. Sales data were retrieved from the OpenSea API, which is currently the largest NFT marketplace in terms of volume traded [52]. In the observed period from 2022-01-20 19:00, when the token traits were revealed (until then the traits were hidden and could not impact the trades), to 2022-05-08 there were more than 15,000 transfers of Azuki tokens.

The token with an attribute *assed_id* equal to 0 was excluded from this dataset because it was owned by the creators of the collection and was traded internally between Ethereum addresses of the same owner in a way that does not reflect the market.

The retrieved transaction data contain information on the date of the transfer, the price and the id of the transferred token. Most of the tokens were traded more times during their history; therefore, the dataset is enhanced by the number of historical trades for each specific token. This feature may contain information on the nature and liquidity of the token.

We assume that the predicted price will be strongly correlated with a floor price 1.2 of the collection at a given time. Since a floor price is based only on an offer and not on the actual sale, we have decided to use the lowest sale price instead. This attribute named *sales_floor* was calculated by finding the minimum price of the previous 5 sales.

3.1.2 Assets

Each of 10,000 Azuki tokens must be associated with its traits. Each token can have an arbitrary number of traits. Metadata containing information on traits can be obtained directly from the collection smart contract, respectively, from the metadata URL provided by the smart contract. In practice we encoutered limitations of this approach, such as restraint of the requests that can be made to the metadata server. Thus, we decided to use OpenSea as a source of asset metadata as well.

The types of traits that an asset can have and the number of possible values for each trait are shown in 3.1.

Trait type	Number of distinct values
Hair	124
Clothing	99
Background	8
Mouth	30
Offhand	54
Type	4
Eyes	27
Headgear	37
Ear	33
Special	10
Face	20
Neck	16

Table 3.1 Trait types and unique values counts

Assets are categorical attributes that cannot be handled by most ML models. Since we are interested in the rarity of each token, a naive approach to encoding categorical variables into numerical is just counting their occurrence (these features have a postfix "_occurence" after the trait name). Yet, we do not exclude the original categorical traits because they might be useful for embedding techniques.

There is no standard for calculating the rarity of assets. However, the basic concept of **rarity scoring** was described by the founder of the analytical platform Rarity.tools in an article [53]. The metrics he described first calculate the rarity of a single trait as follows:

$$p(a) = \frac{number\ of\ assets\ with\ \ trait\ a}{total\ number\ of\ assets} \tag{3.1}$$

then the rarity of the asset is computed as the sum of rarities of its n traits:

$$rarity\ = \sum_{i=1}^{n} \frac{1}{p(a_i)} \tag{3.2}$$

This method was then improved by reflecting the number of traits the asset has; sometimes called **trait count weight**. For example, token from the Azuki collection with the id. 2152 is the only asset in the collection that has only five traits, whereas all other tokens have at least six attributes, which makes it rare.

Furthermore, the distribution of the values for each trait type must be taken into account. This is commonly referred to as **trait normalization**. If the values of a trait type are equally distributed, the rarity of this trait type shall contribute less to the rarity score than the rarity of the trait type, which is distributed unequally. In this work, we compute the contribution weight as the mean of the probabilities of values of the trait type.

Although the exact algorithm for computing rarities is not standardized, the usage of trait normalization and trait count weight has become a standard among NFT analytical platforms.

3.2 Data exploration

The first step to better understand the data was to visualize the traits of the assets and their interaction with the sale price. This can be represented by the visualization of PCA with two principal components, each represented by an axis of the scatterplot and the points colored by the price relative to *sales_floor*.

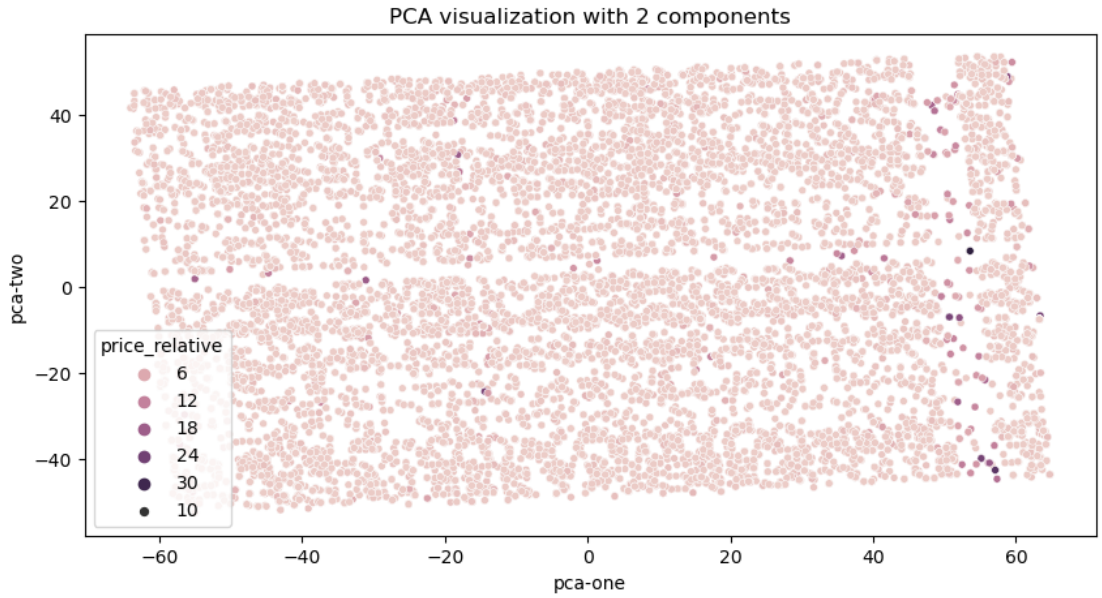

Figure 3.1 PCA visualization of traits and sales price

In this visualization 3.1, two narrow strips with lower asset density can be seen, which were traded at the highest prices relative to the cheapest sale at the time. This supports our assumption that less common combinations of traits have an impact on the sale price.

It is also worth examining the relationship of the price with the cheapest sale at a given time.

We can see from the figure 3.2 and table 3.2 that most of the assets are traded at prices close to

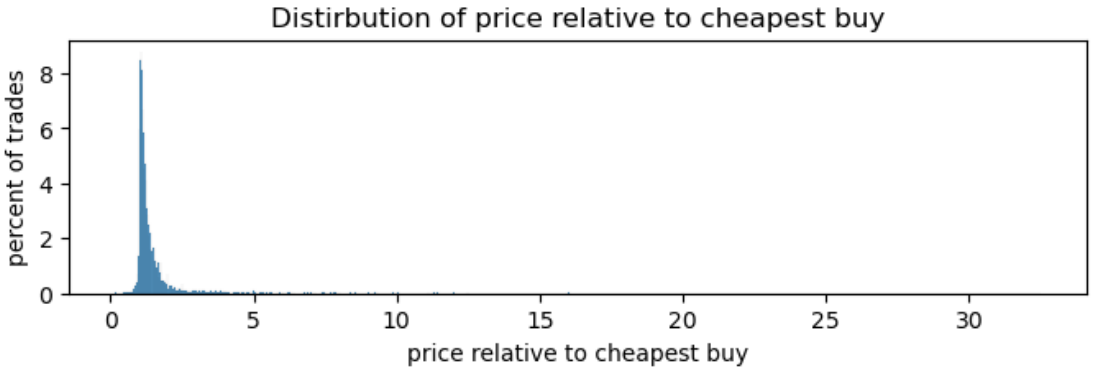

Figure 3.2 Histogram of price relative to the cheapest purchase

count	21698.0
mean	1.42
std	1.12
min	0.16
0.25 quantile	1.05
0.5 quantile	1.15
0.75 quantile	1.42
max	32.5

Table 3.2 Statistical analysis of floor price relative to the cheapest sale

the floor price of sales, and more than 75% of the assets are traded at a price less than 1.5 times the *sales_floor* price. This is not surprising because the vast majority of assets have common traits, whereas there are much fewer rare assets, and consequently, much fewer trades of these rare tokens. This may cause problems for the models because they can learn more about the common tokens than about the rare ones. The dataset is imbalanced1.3.3 in terms of the price distribution relative to the lowest sales price at a given time.

3.3 Models

In this section, we propose and measure the accuracy of multiple models that can be used to solve the problem. The models should be diverse. At the end of this section, these models are combined with the ensemble techniques.

3.3.1 Gradient boosted regression tree

The first proposed model is the gradient-boosted tree 1.3.2 implemented with the Scikit library.

This model accepts only numerical attributes; thus it was wrapped by pipeline, which excludes the categorical traits from the dataset and leaves only the occurence of traits and its rarity score. This pipeline also normalizes the features to improve the accuracy of the model.

The best hyperparameters of the tree were exhaustively searched and evaluated with cross-

validation. The hyperparameters found are shown in 3.3.

name	description	value
learning_rate	reduces the contribution of each tree	0.2
n_estimators	number of boosting stages	85

Table 3.3 Hyperparameters of gradient-boosted tree model

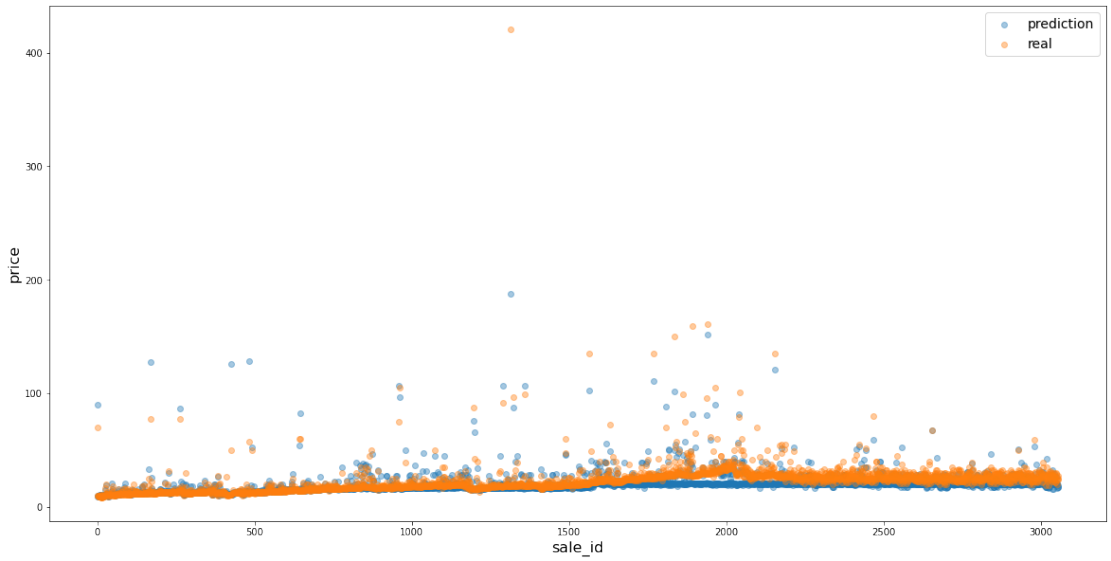

Figure 3.3 Predictions of gradient boosted regressor for testing data

In 3.3 we can see the predictions of the gradient-boosted tree model for previously unseen data. The problem is that it often overprices sales, which is undesirable for traders and does not follow the change of the floor price. Also, it uses the explicitly computed rarity instead of deciding which traits are important and which are not on its own. On the other hand, the regression tree was able to deal with the outliers with reasonably good accuracy since the training data were unbalanced and no special technique was implemented to handle that.

statistic	absolute error in ETH	percentage error
mean	4.271	15.371
std	6.573	12.42
min	0.0	0.003
quantile 0.25	0.89	5.409
median	2.607	12.266
quantile 0.75	6.296	23.563
max	233.364	151.687

Table 3.4 Descriptive statistics for absolute and percentage test error of gradient-boosted tree

The MAPE of the gradient-boosted tree model was 14.5% and almost 50% of all assets had an absolute percentage error lower than 10%

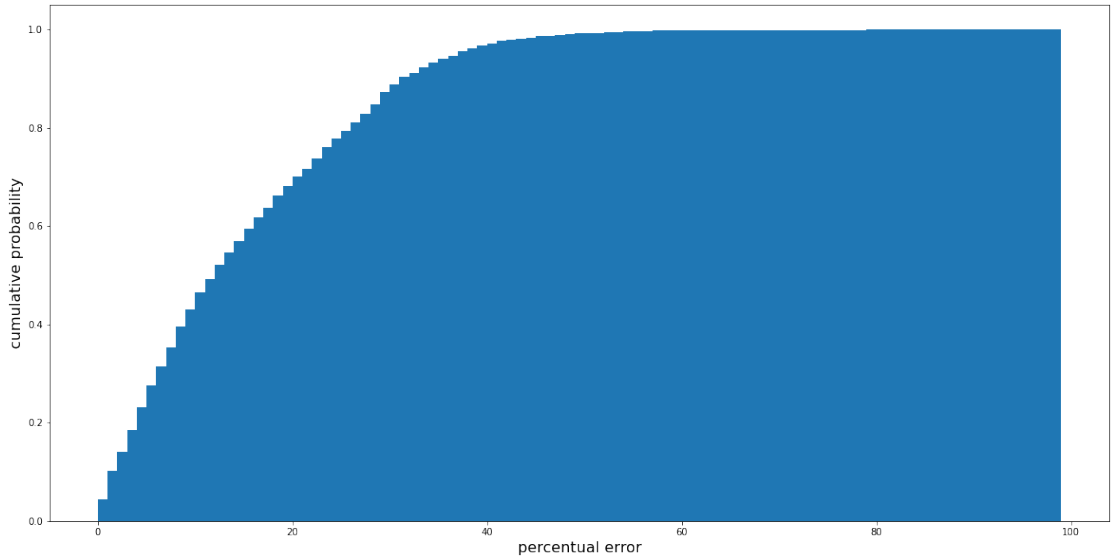

Figure 3.4 Empirical cumulative distribution function of test prediction errors of gradient-boosted tree model

3.3.2 Deep and wide ANN

The price of the tokens could not only be driven by the rarity of their traits, but some fairly common traits may also become popular among traders and consequent demand can raise the price of the asset. For this reason, we propose two ANN models which can determine how the traits impact the price without any explicit formula. However, we do not exclude information on the explicit rarity score and the occurrence of traits.

We want the models to be able not only to generalize the attributes but also to memorize the specific combinations of them regarding the sale price. The charachteristics of deep and wide learning seem to be suitable for this problem. In this section of the work, we will refer to the part of the models responsible for generalization as **deep branch** and to the other part responsible for memorization as **wide branch**.

For building the ANN models, the Keras framework will be used. Keras is a deep learning framework built on top of the Tensorflow2 library, which was developed by Google.

To allow the model to determine the relationship between traits and the sale price, we can leverage the deep embedding technique 1.5.5. Our neural network models have two inputs; one for the categorical attributes and the other for the numerical ones. First, categorical inputs that contain the traits of a given asset are encoded in a numerical representation by converting the categorical string to their index in the vocabulary search table. The vocabulary consists of all possible values that a trait can have and is adapted during the training period by the StringLookupLayer.

Then the numerical representation of the traits is fed to the embedding layer provided by Keras. This layer randomly initializes a matrix that contains a dense vector representation of traits [54], which will be further trained to contain meaningful values. The embedding layer is also responsible for looking up in this matrix.

The embedded vector representation of traits is pushed forward in the network together with the rest of numerical attributes into the deep branch of our neural network, which is a sequence of fully connected layers. Each of these branches uses a ReLU activation function. This branch needs to be deep enough to learn and generalize the relationships between the input features and the final price.

There are multiple approaches to avoid overfitting 1.5.4 in our network. The dropout technique is used and, as suggested in [55], dropout layers are placed before each hidden layer and low dropout probabilities are set for these layers. The max-norm method is applied together with a dropout technique, because it performed very well in [42]. And finally, the early-stopping technique is used to stop the training and restore the best weights after 6 epochs of not improving the validation loss.

The numerical input is also passed to the wide branch. The processings there are fairly simple — all numerical inputs are scaled to have mean equal to zero and standard deviation equal to one.

Finally, the outputs of the deep and wide branches are concatenated. This last layer is fully connected to the single output. The output has a linear activation function, which is typical for regression ANNs.

3.3.2.1 Conservative model

As we have mentioned, there will be two instances of the previously described architecture; each with different hyperparameters and targeting slightly different problems.

The first might be called the conservative one, because it aims to predict well the prices of assets, which were traded for **lower prices**. This model should not be distracted by outlying assets and neither should the estimated prices be higher than the real values.

This model uses the Huber loss function 1.5.3 with the parameter α set to the median of the ratio of the sale price to the $sales\_floor$ price in the training dataset. Therefore; the loss error grows quadratically for the common assets, but for the rare assets it grows linearly, so the outliers do not mislead the model. Also, the dataset is left unbalanced with the vast majority of common assets in it.

From the predictions visualized in figure 3.5 we can see that the conservative model is capable of modeling trend changes of the cheapest sales and rarely overprices an asset. It can signal that rarer NFTs are sold at higher prices, but the error for these assets is quite large.

statistic	absolute error in ETH	percentage error
mean	3.16	9.756
std	4.876	9.143
min	0.002	0.007
quantile 0.25	0.789	3.004
median	1.833	6.57
quantile 0.75	3.908	14.014
max	72.838	72.116

Table 3.5 Descriptive statistics for absolute and percentage test error of conservative ANN

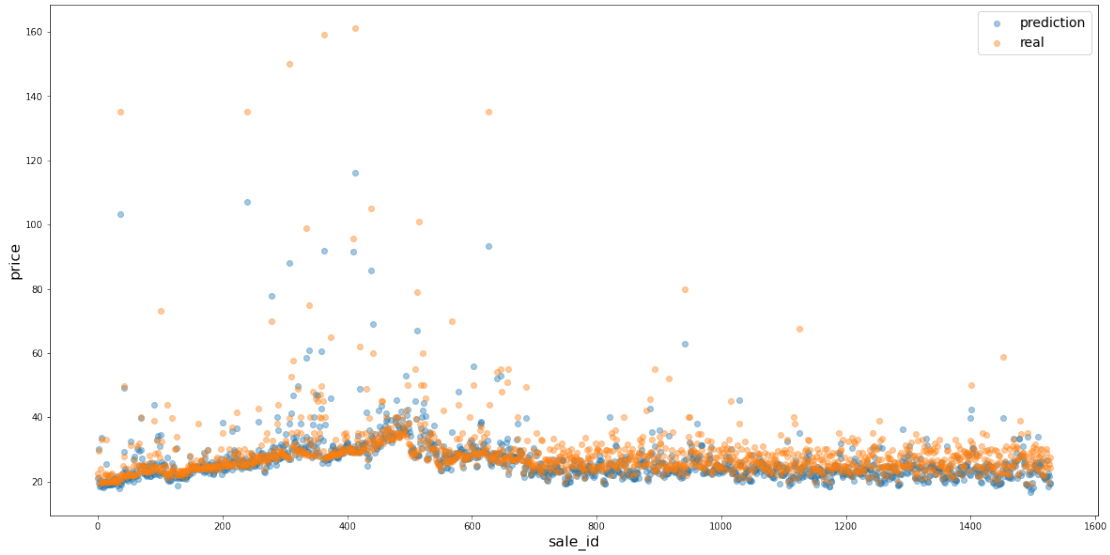

Figure 3.5 Predictions of conservative model for testing data

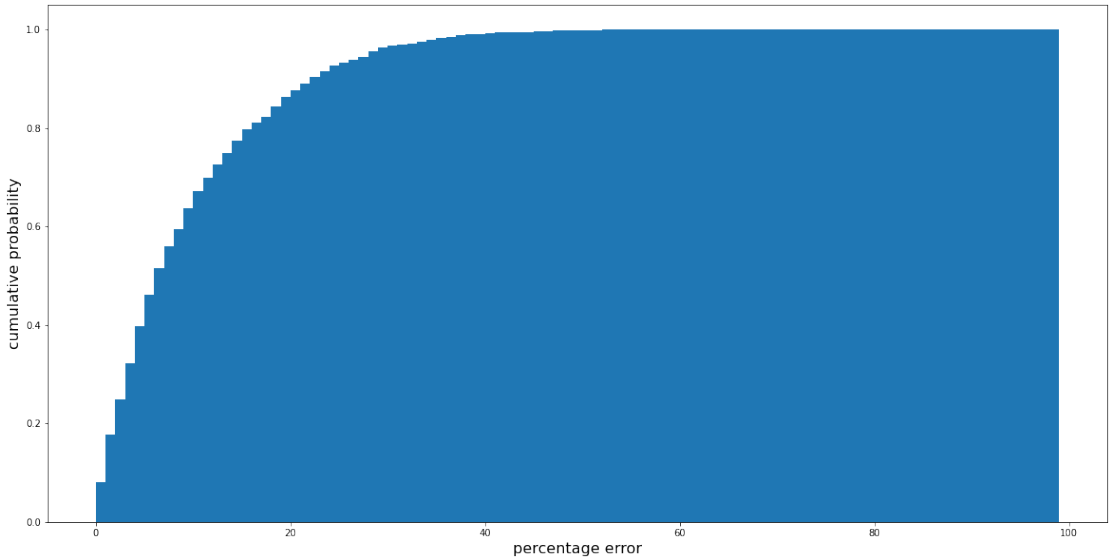

Figure 3.6 Empirical cumulative distribution function of testing error for conservative model

3.3.2.2 Progressive model

The second implementation of the deep and wide ANN architecture tries to improve the estimates of the rarer tokens.

This is achieved by **balancing** the mini-batches during the training process. Each of these batches should contain a fraction of sales data that has the price to the *sales_floor* ratio greater than the value of the parameter *balance_threshold*. This compensates for the fact that such sales are rarely seen during the training. These sales are randomly sampled for each mini-batch, and for this reason, the model may perform slightly differently after it is trained again. Also, the model uses the mean squared error metric, which is more sensitive to outliers.

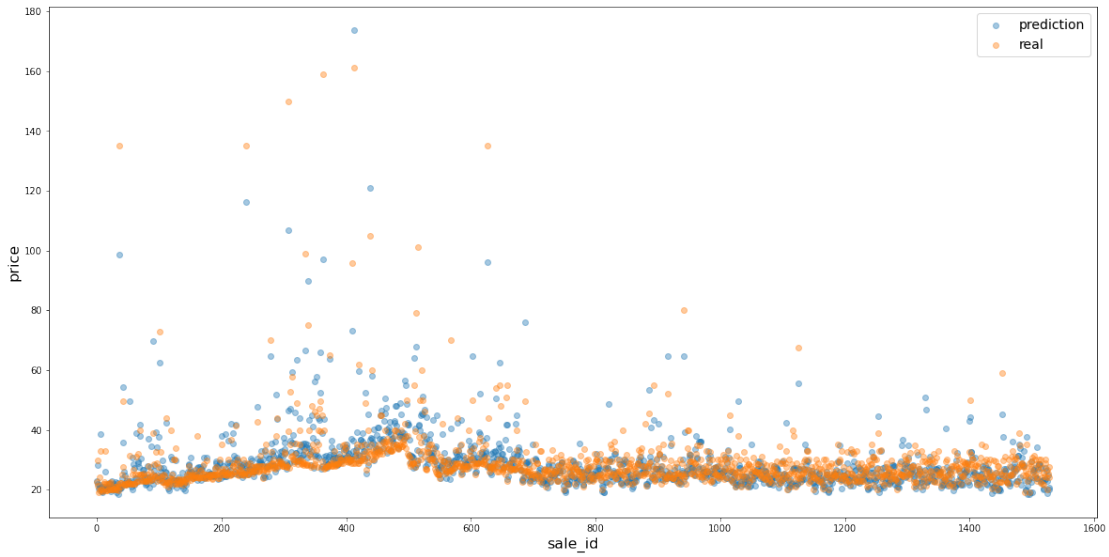

Figure 3.7 Predictions of progressive model for test data

In figure 3.7 we can see that the trend of the test predictions of the progressive model was quite similar to the predictions of the conservative model, but it was more optimistic toward the rarer assets; sometimes it even overvalued some NFTs.

statistic	absolute error in ETH	percentage error
mean	3.359	10.794
std	4.504	10.537
min	0.009	0.033
quantile 0.25	0.917	3.523
median	2.142	8.04
quantile 0.75	4.181	14.853
max	71.281	121.267

Table 3.6 Descriptive statistics for absolute and percentage test error of progressive ANN

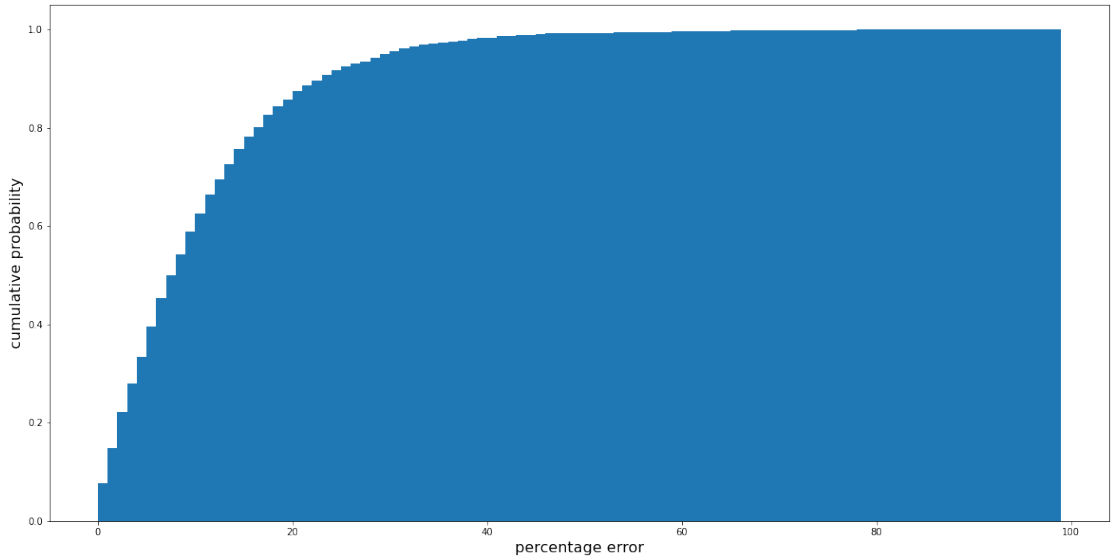

Figure 3.8 Empirical cumulative distribution function of testing error for progressive model

3.3.3 Ensemble

And finally, the capabilities and performance of the models described above must be properly combined to use them to their maximum potential. This can be achieved with ensemble techniques 1.3.2, such as **stacking** or **model averaging**, because these techniques allow the combination of diverse models.

We will compare the performance of a StackingRegressor and VotingRegressor provided by the Sklearn library. These techniques require the underlying models to follow the library's interface for estimators; thus, a wrapper above the ANNs implemented with a different framework is needed.

Let us start with the VotingRegressor. This model fits its underlying estimators and averages their predictions. Table 3.7 shows the precision achieved by this model.

statistic	absolute error in ETH	percentage error
mean	3.544	11.334
std	4.471	8.813
min	0.002	0.008
quantile 0.25	1.197	4.62
median	2.45	9.148
quantile 0.75	4.617	16.107
max	73.367	72.641

Table 3.7 Descriptive statistics for absolute and percentage test error of averaging ensemble model

The StackingRegressor combines the predictions of the underlying estimators, in our case, the gradient-boosted tree, conservative, and progressive ANNs, by applying the final regressor to their output. We use a Ridge regression for this purpose, which applies the L_2 penalization and is suitable for situations where independent variables are highly correlated [56]. We also tested

multiple values for the penalization hyperparameter α.

statistic	absolute error in ETH	percentage error
mean	3.004	9.246
std	4.757	8.638
min	0.003	0.012
quantile 0.25	0.741	2.828
median	1.743	6.501
quantile 0.75	3.86	13.256
max	75.737	71.998

Table 3.8 Descriptive statistics for absolute and percentage test error of stacking ensemble model

The metrics of the stacking model 3.8 show that this model performs better than the averaging one. Therefore; we propose this technique to combine the gradient-boosted tree, conservative ANN, and progressive ANN to tackle the problem of estimating the value of NFTs.

Chapter 4

Evaluation

In this chapter, the results and possible use cases of the proposed model are discussed. The estimations of this model are then put in the context of NFT trading.

Our stacking model achieved an MAPE 3.8 of 9.2%, which is better than the average of the results of the model proposed in [50]. However, each of the models was trained on different collections, so they cannot be compared appropriately.

The solution proposed in this paper is not suitable for use cases where exact predictions are crucial. An example of such a scenario might be a situation in which the NFT serves as a collateral for a loan.

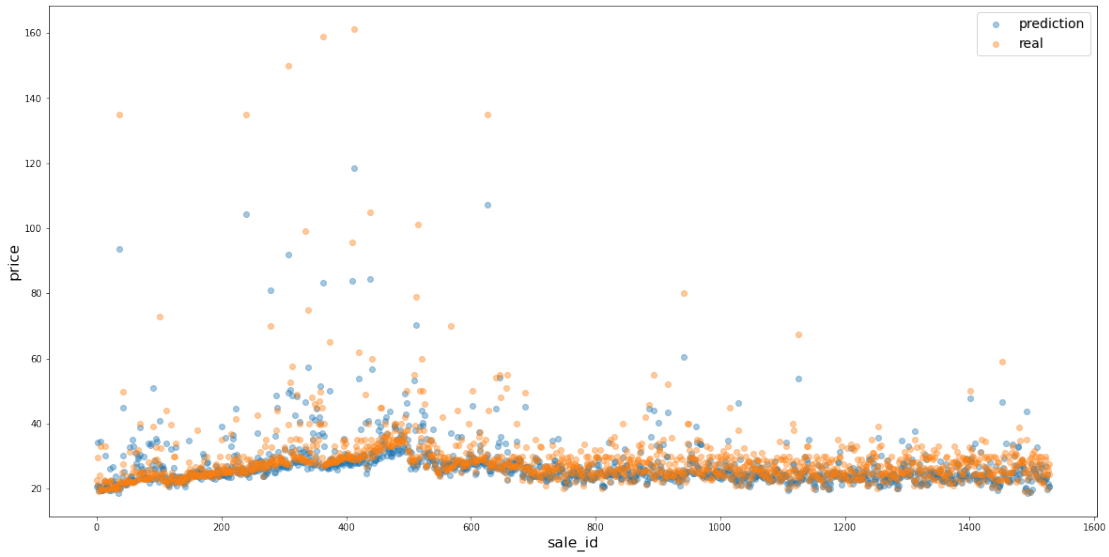

Figure 4.1 The predictions of the proposed model for previously unseen data

However, as shown in 4.1, the model introduced in this thesis is capable of capturing asset price trends with reasonable precision. This might be leveraged by NFT traders. Trading NFTs is

always extremely speculative, but the risk is balanced with the possibility of profit in the tens or hundreds percent. Therefore, the insight about valuing NFTs is valuable for traders, even though the estimation might have an offset of a couple of percent.

A typical scenario would be for a trader to let the trained model estimate the prices of assets that are currently listed for sale and look for emerging promising deals.

The model's predictions are rather conservative as it rarely overprices any asset. An opportunity for NFT trader would be to buy one of the assets which are undervalued according to the prediction in hope that the model estimated the price correctly and someone is willing to sell this asset cheaper.

Figure 4.2 shows the trades of the ten most **undervalued** tokens (the prices were estimated to be higher than they actually were) that were traded later in the testing dataset period. The sale prices were calculated relative to the $sales\_floor$ price at the time of sale. The relative price of almost all of these assets increased over time. This suggests that the model estimated the price higher correctly and that the assets offered for sale for a price lower than the price estimated by the model might be profitable deals. This also opens up new opportunities for **automation of NFT trading**.

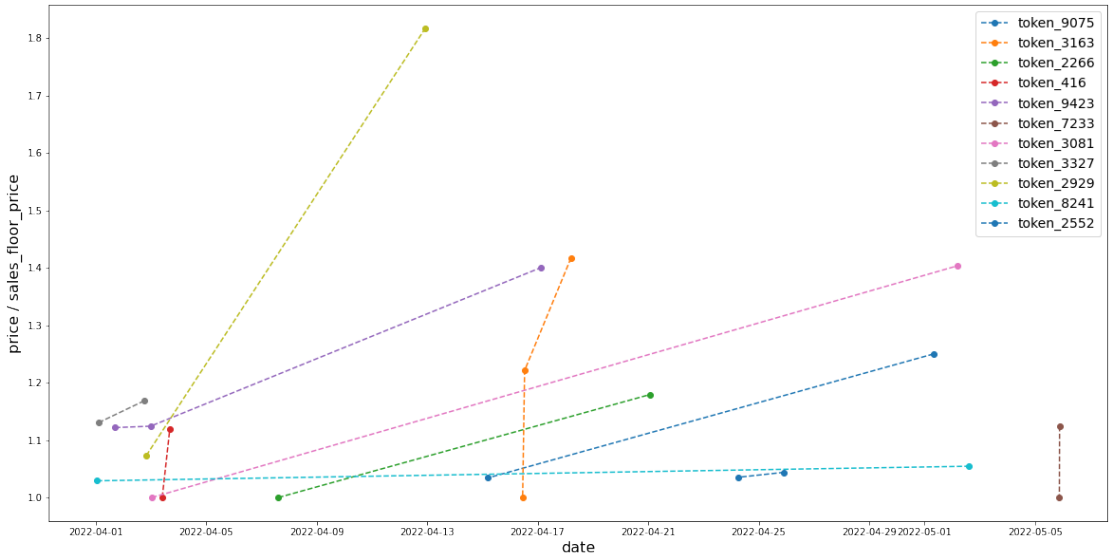

Figure 4.2 Trades of the ten most underpriced tokens in observed period

Investors also want to avoid buying items that will lose value. This is another use-case in which the proposed model might be helpful. If the model estimates the price of an asset significantly lower than it is offered to buy for, it could indicate that the asset may lose its value in the future; we say that the asset is **overpriced**.

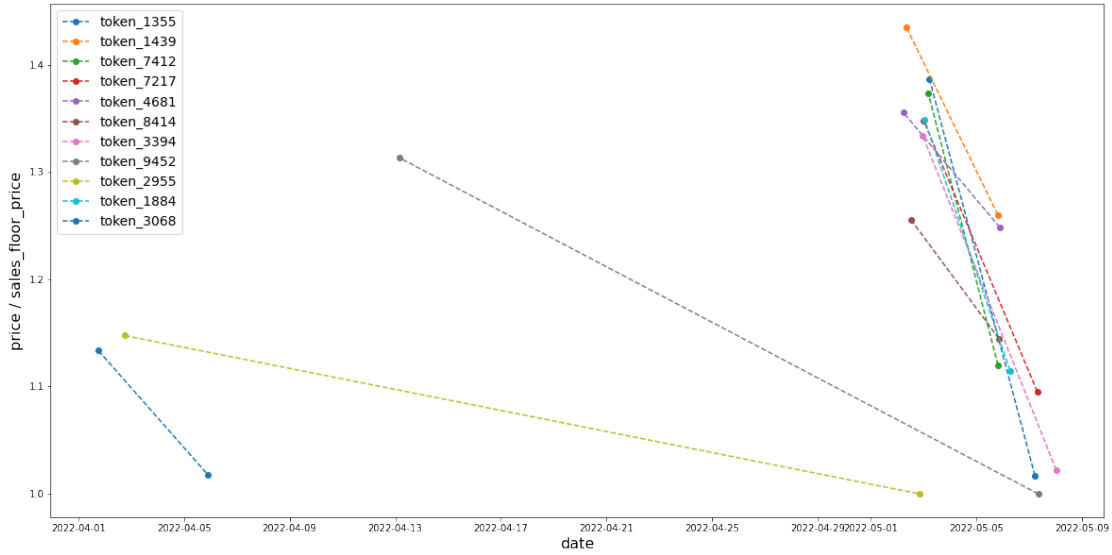

Figure 4.3 Trades of the ten most overpriced tokens in observed period

Figure 4.3 shows the trades of the ten most overpriced assets, which have been traded later in the period of the testing dataset. The price to $sales\_floor$ ratio was lower for the sales that occurred after the model estimated that the asset should be cheaper. Such trades should be avoided.

Conclusion

The goal of this thesis was to propose and implement a method to estimate the value of NFTs using ML techniques. This model was meant to be useful in practice for traders.

We started with the description of Ethereum ecosystem, NFTs, and digital art collectibles in the theoretical part 1.1.

Then methods for estimating NFT values were researched in the section 2.2. This research has shown that the area is very new and has not been thoroughly researched yet. As a consequence, this thesis is based on the application of general machine learning concepts to the specific problem of NFTs.

In the practical part, we tested multiple models to solve the problem of estimating the NFTs value. These models were shown to perform best when combined together, forming the final estimator.

The implementation of the final model was capable of estimating the NFT price with an absolute percentage error of less than 10%. These estimations may help identify underpriced assets. Furthermore; we demonstrated that some of the predictions reflected the value of the asset better than the price requested by their sellers, which creates opportunities for profitable trading.

The model for valuating NFTs proposed in this thesis may become the foundation for practical solutions and have an impact on how people trade NFTs.

Bibliography

1. BUTERIN, Vitalik. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform [online]. 2014. Available also from: https://ethereum.org/en/whitepaper/. [Cited 2022-03-21].
2. CHAUM, David. Blind signatures for untraceable payments [online]. 1983. Available also from: http://www.hit.bme.hu/~buttyan/courses/BMEVIHIM219/2009/Chaum.BlindSigForPayment.1982.PDF. [Cited 2022-03-28].
3. NAKAMOTO, Satoshi. *Bitcoin: A peer-to-peer electronic cash system [online]*. 2009. Available also from: http://www.bitcoin.org/bitcoin.pdf. [Cited 2022-03-28].
4. LEWIS, Antony. A gentle introduction to Ethereum [online]. 2016. Available also from: https://bitsonblocks.net/2016/10/02/gentle-introduction-ethereum/. [Cited 2022-03-30].
5. WOOD, Gavin. Ethereum: A secure decentralised generalised transaction ledger [online]. BERLIN VERSION 934279c. 2022. Available also from: https://ethereum.github.io/yellowpaper/paper.pdf. [Cited 2022-03-30].
6. FOUNDATION, Ethereum. Ethereum Account [online]. 2022. Available also from: https://ethereum.org/en/developers/docs/accounts. [Cited 2022-03-30].
7. ANDREAS M. ANTONOPOULOS, Gavin Wood [online]. Mastering Ethereum. In: O'Reilly Media, Inc., 2018, chap. 2. Available also from: https://github.com/ethereumbook/ethereumbook. [Cited 2022-04-02].
8. ETHEREUM FOUNDATION. Introduction to smart contracts [online]. 2022. Available also from: https://ethereum.org/en/developers/docs/smart-contracts/. [Cited 2022-04-05].
9. ENTRIKEN, William; SHIRLEY, Dieter; EVANS, Jacob; SACHS, Nastassia. EIP-721: Non-Fungible Token Standard [online]. *Ethereum Improvement Proposals, no. 721*. 2018. Available also from: https://eips.ethereum.org/EIPS/eip-721. [Cited 2022-04-05].
10. MARCOBELLO, Mason. CryptoPunks, CryptoCats and CryptoKitties: How They Started and How They're Going[online]. 2022. Available also from: https://www.coindesk.com/learn/cryptopunks-cryptocats-and-cryptokitties-how-they-started-and-how-theyre-going/. [cited on 02-04-2022].
11. MORALIS BLOG. *How to Develop an NFT Rarity Ranking dApp [online]*. 2021. Available also from: https://moralis.io/how-to-develop-an-nft-rarity-ranking-dapp/. [cited on 05-05-2022.

12. SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*. 2000, vol. 44, no. 1.2, pp. 206–226. Available from DOI: 10.1147/rd.441.0206.
13. MITCHELL, Tom M. *Machine learning, International Edition*. McGraw-Hill, 1997. McGraw-Hill Series in Computer Science. ISBN 978-0-07-042807-2. Available also from: https://www.worldcat.org/oclc/61321007.
14. GÉRON, Aurélien. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow". In: "Second". "O'Reilly Media, Inc.", 2019, pp. 1–6. ISBN: 9781492032649.
15. POLIKAR, R. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*. 2006, vol. 6, no. 3, pp. 21–45. Available from DOI: 10.1109/MCAS.2006.1688199.
16. GÉRON, Aurélien. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow". In: "Second". "O'Reilly Media, Inc.", 2019, pp. 199–208. ISBN: 9781492032649.
17. WOLPERT, David H. Stacked generalization. *Neural Networks*. 1992, vol. 5, no. 2, pp. 241–259. ISSN 0893-6080. Available from DOI: https://doi.org/10.1016/S0893-6080(05)80023-1.
18. YANG, Yuzhe; ZHA, Kaiwen; CHEN, Yingcong; WANG, Hao; KATABI, Dina. Delving into Deep Imbalanced Regression. In: MEILA, Marina; ZHANG, Tong (eds.). *Proceedings of the 38th International Conference on Machine Learning*. PMLR, 2021, vol. 139, pp. 11842–11851. Proceedings of Machine Learning Research. Available also from: https://proceedings.mlr.press/v139/yang21m.html.
19. TORGO, Luís; RIBEIRO, Rita; PFAHRINGER, Bernhard; BRANCO, Paula. SMOTE for Regression. In: 2013, vol. 8154, pp. 378–389. Available from DOI: 10.1007/978-3-642-40669-0_33.
20. CUI, Yin; JIA, Menglin; LIN, Tsung-Yi; SONG, Yang; BELONGIE, Serge. Class-balanced loss based on effective number of samples. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 9268–9277.
21. SHIMIZU, Ryota; ASAKO, Kosuke; OJIMA, Hiroki; MORINAGA, Shohei; HAMADA, Mototsugu; KURODA, Tadahiro. Balanced Mini-Batch Training for Imbalanced Image Data Classification with Neural Network. In: *2018 First International Conference on Artificial Intelligence for Industries (AI4I)*. 2018, pp. 27–30. Available from DOI: 10.1109/AI4I.2018.8665709.
22. GÉRON, Aurélien. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow". In: "Second". "O'Reilly Media, Inc.", 2019, pp. 219–221. ISBN: 9781492032649.
23. ABDI, Hervé; WILLIAMS, Lynne J. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*. 2010, vol. 2, no. 4, pp. 433–459.
24. HYAFIL, Laurent; RIVEST, Ronald L. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*. 1976, vol. 5, no. 1, pp. 15–17. ISSN 0020-0190. Available from DOI: https://doi.org/10.1016/0020-0190(76)90095-8.
25. MANEK, Asha S; SHENOY, P Deepa; MOHAN, M Chandra, et al. Aspect term extraction for sentiment analysis in large movie reviews using Gini Index feature selection method and SVM classifier. *World wide web*. 2017, vol. 20, no. 2, pp. 135–154.
26. BREIMAN, Leo; FRIEDMAN, Jerome H; OLSHEN, Richard A; STONE, Charles J. *Classification and regression trees*. Routledge, 2017.
27. LOH, Wei-Yin et al. Classification and regression tree methods. *Encyclopedia of statistics in quality and reliability*. 2008, vol. 1, pp. 315–323.

28. HASTIE, Trevor; TIBSHIRANI, Robert; FRIEDMAN, Jerome H; FRIEDMAN, Jerome H. The elements of statistical learning: data mining, inference, and prediction. In: Springer, 2009, vol. 2, pp. 337–384.
29. MCCULLOCH, Warren S; PITTS, Walter. A logical calculus of the ideas immanent in nervous activity[online]. *The bulletin of mathematical biophysics*. 1943, vol. 5, no. 4, pp. 115–133. Available also from: https://link.springer.com/article/10.1007/BF02478259. [Cited 2022-04-08].
30. GÉRON, Aurélien. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow". In: "Second". "O'Reilly Media, Inc.", 2019, pp. 280–284. ISBN: 9781492032649.
31. CYBENKO, George. Approximation by superpositions of a sigmoidal function[online]. *Mathematics of control, signals and systems*. 1989, vol. 2, no. 4, pp. 303–314. Available also from: https://link.springer.com/article/10.1007/BF02551274. [Cited 2022-04-10].
32. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. Deep Learning [online]. In: MIT Press, 2016, chap. 6. http://www.deeplearningbook.org.
33. GÉRON, Aurélien. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow". In: "Second". "O'Reilly Media, Inc.", 2019, pp. 290–291. ISBN: 9781492032649.
34. ZAMANLOOY, Babak; MIRHASSANI, Mitra. Efficient VLSI Implementation of Neural Networks With Hyperbolic Tangent Activation Function. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*. 2014, vol. 22, no. 1, pp. 39–48. Available from DOI: 10.1109/TVLSI.2012.2232321.
35. RAMACHANDRAN, Prajit; ZOPH, Barret; LE, Quoc V. Searching for Activation Functions. *CoRR*. 2017, vol. abs/1710.05941. Available from arXiv: 1710.05941.
36. LIANG, Shiyu; SRIKANT, Rayadurgam. Why deep neural networks for function approximation? *arXiv preprint arXiv:1610.04161*. 2016.
37. TOFALLIS, Chris. A better measure of relative prediction accuracy for model selection and model estimation. *Journal of the Operational Research Society*. 2015, vol. 66, no. 8, pp. 1352–1362.
38. GOODWIN, Paul; LAWTON, Richard. On the asymmetry of the symmetric MAPE. *International journal of forecasting*. 1999, vol. 15, no. 4, pp. 405–408.
39. GNEITING, Tilmann. Making and Evaluating Point Forecasts. *Journal of the American Statistical Association*. 2009, vol. 106. Available from DOI: 10.1198/jasa.2011.r10138.
40. HUBER, Peter J. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*. 1964, vol. 35, no. 1, pp. 73–101. Available from DOI: 10.1214/aoms/1177703732.
41. ELLIOTT, Conal. The Simple Essence of Automatic Differentiation. *Proc. ACM Program. Lang.* 2018, vol. 2, no. ICFP. Available from DOI: 10.1145/3236765.
42. SRIVASTAVA, Nitish; HINTON, Geoffrey; KRIZHEVSKY, Alex; SUTSKEVER, Ilya; SALAKHUTDINOV, Ruslan. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*. 2014, vol. 15, pp. 1929–1958.
43. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. Deep Learning [online]. In: MIT Press, 2016, pp. 116–117. http://www.deeplearningbook.org.
44. GÉRON, Aurélien. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow". In: "Second". "O'Reilly Media, Inc.", 2019, pp. 134–137. ISBN: 9781492032649.

45. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. Deep Learning [online]. In: MIT Press, 2016, chap. 7. http://www.deeplearningbook.org.

46. GÉRON, Aurélien. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. Second. 2019, pp. 433–437. ISBN: 9781492032649.

47. MIKOLOV, Tomas; CHEN, Kai; CORRADO, Greg; DEAN, Jeffrey. Efficient estimation of word representations in vector space[online]. *arXiv preprint arXiv:1301.3781*. 2013. Available also from: https://arxiv.org/abs/1301.3781. [Cited 2022-04-10].

48. CHENG, Heng-Tze; KOC, Levent; HARMSEN, Jeremiah; SHAKED, Tal; CHANDRA, Tushar; ARADHYE, Hrishi; ANDERSON, Glen; CORRADO, Greg; CHAI, Wei; ISPIR, Mustafa, et al. Wide & deep learning for recommender systems. In: *Proceedings of the 1st workshop on deep learning for recommender systems*. 2016, pp. 7–10.

49. CHENG, Heng-Tze. Wide & Deep Learning: Better Together with TensorFlow [online]. 2016. Available also from: [https://ai.googleblog.com/2016/06/wide-deep-learning-better-together-with.html]. [Cited on [2022-04-16].

50. RYKER. How to value items in NFT projects? — Part 1[online]. 2021. Available also from: https://medium.com/nftbank-ai/how-to-value-items-in-nft-projects-part-1-a87be8bcb21d. [cited on 19-04-2022].

51. *Taking NFT Pricing to the Next Level with Machine Learning*. 2021. Available also from: https://mirror.xyz/0x82FE4757D134a56BFC7968A0f0d1635345053104/dHNbc5bE8hnXB3xtjZInZ7TApKrE-S8QcexXrcj2QPo. [cited on 19-04-2021].

52. SERGEENKOV, Andrey. Top Six NFT Marketplaces [online]. 2022. Available also from: https://coinmarketcap.com/alexandria/article/top-six-nft-marketplaces. [Cited 2022-04-17].

53. ANONYMOUS. Ranking Rarity: Understanding Rarity Calculation Methods [online]. 2021. Available also from: https://raritytools.medium.com/ranking-rarity-understanding-rarity-calculation-methods-86ceaeb9b98c. [Cited on 2022-01-05].

54. GÉRON, Aurélien. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow". In: "Second". "O'Reilly Media, Inc.", 2019, pp. 433–436. ISBN: 9781492032649.

55. HINTON, Geoffrey E; SRIVASTAVA, Nitish; KRIZHEVSKY, Alex; SUTSKEVER, Ilya; SALAKHUTDINOV, Ruslan R. Improving neural networks by preventing co-adaptation of feature detectors [online]. *arXiv preprint arXiv:1207.0580*. 2012. Available also from: https://arxiv.org/pdf/1207.0580.pdf. [cited on 10-04-2022].

56. HILT, Donald E.; SEEGRIST, Donald W.; SERVICE., United States. Forest; NORTHEASTERN FOREST EXPERIMENT STATION (RADNOR, Pa.) Ridge, a computer program for calculating ridge regression estimates. In: Upper Darby, Pa, Dept. of Agriculture, Forest Service, Northeastern Forest Experiment Station, 1977, [n.d.], vol. no.236, p. 10. Available also from: https://www.biodiversitylibrary.org/item/137258. https://www.biodiversitylibrary.org/bibliography/68934.

Contents of enclosed medium

- `readme.txt`
- `src`
  - `data` — folder containing datasets
  - `notebooks` — Jupyter notebooks source
  - `thesis` — thesis source in LaTeX format
- `text` — thesis text
  - `thesis.pdf` — thesis text in PDF format