NATIONAL RESEARCH UNIVERSITY

HIGHER SCHOOL OF ECONOMICS

Faculty of Computer Science

Bachelor’s Programme 'HSE University and University of London Double Degree Programme in Data Science and Business Analytics'

UDC 004.62, 004.8

Research Project Report

on the topic Implementation of algorithms, mathematical models and neural networks in board games in the context of poker

Moscow 2021

Contents

1 Key terms and definitions

Machine Learning

That is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention

Neural network

It can be described as a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates.

Key concepts in ML problems

In general, there are two ways how an algorithm can be trained:

1) Supervised Machine Learning is used in order to train the algorithm to predict target values from the given labeled data. There are several approaches: *regression* (target values are continuous values) and *classification* (target values are discrete classes).

2) Unsupervised Machine Learning is needed in those cases when the given data is unlabeled or unstructured. This can be done by clustering (groups of similar instances in the data have to be found) or by finding unusual patterns (that is also called outlier detection).

Steps in ML problem-solving

The process consists of three main steps. First of all, the representation part is done. It should be figured out how to represent the learning problem in terms of something the computer can understand. The given data must be formulated in such way that it can be used as an input to an algorithm. This process involves 2 things. We need to convert each input object ("sample") into a set of features that describe the object. After that, a learning model is picked (the type of classifier that we want the system to learn). Secondly, the evaluation part is considered. Here the effectiveness of different classifiers is compared, after that an evaluation method is chosen (evaluation method provides the type of quality and the accuracy score). It is important to choose the right one, since a good classifier will have higher accuracy and will make a prediction that matches the correct true label most of the time. Finally, optimization is done. During this step the search for the optimal classifier (that gives the best evaluation outcome for the particular problem

Feature

It is an individual measurable property or characteristic of a phenomenon being observed. In general, there are 4 types of features in Machine Learning: *binary* (a feature that is consists of only 2 variables: it can be either 0 or 1 or "Yes" or "No"), *numerical* (a feature that reads real numbers), *categorical* (a feature that takes values $U_1, ..., U_n$ from some set U, the difficulty is consisted in the impossibility of these values‘ comparison between each other), *ordinal* (a feature that is rather similar to the categorical one, arithmetic operations with values of the feature are as well prohibited, however, one can put the values in order, therefore, it is possible to compare them).

Loss function

Loss functions are used to determine the error (or simply, “the loss”) between the output of the algorithms and the given target value. In other words, the loss function expresses how far from the correct answer the computed output is. There are several ways to calculate error. Two of the most popular loss functions in machine learning are the *0-1 loss function* and the *quadratic loss function.*

Metrics to evaluate the ML algorithm

1) Regression metrics:

Mean squared error – MSE is calculated by the sum of square of prediction error which is real output minus predicted output and then divide by the number of data points. It gives you an absolute number on how much your predicted results deviate from the actual number. Mean absolute error is similar to Mean Square Error (MSE). However, instead of the sum of square of error in MSE, MAE is taking the sum of absolute value of error.

2) Statistical metrics:

Correlation – it is a measure of relationship between two variables. There are several relations involving variables such as: linear, (in general) non-linear, and others. Also, variables can have differing quantities of correlation to each other.

Cross-validation is a re-sampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation.

Mean squared error is the average squared difference between the estimated values and the actual ones. It gives an absolute number on how much the predicted results deviate from the correct number.

Mean absolute error is a measure of error which can be described as the mean absolute values of the received predictions (result) on over all instances in the test.

Correlation is a measure of relationship (or connection of) between two variables and features. Variables can have differing quantities of correlation to each other

2 Introduction

2.1 Abstract

Nowadays because of rapidly increasing demand for big data analysis Machine Learning and the Artificial Intelligence have become one of the most essential and integral part of modern computer science and IT. To start with, extremely high accuracy can be obtained through constant learning and the different algorithms improvements. The more data is given to them, the more accurate output will be received. In addition, there are two ways to create a program code analyzing data: to write by hand many small pieces of a code and then bring everything together or to apply the neural networks in order to simplify the process and to spend much less time. Different neural networks try to find the correlation between events and provide us with better understanding of the fast-evolving world. Finally, neural network optimize various working processes and allow business to become much more profitable.

To conclude, the importance of the neural networks cannot be underestimated, as the key to the successful future is considered to be behind the technology. The demand for it will continue to raise and ease lives of usual people. That is why our project team has decided to choose exactly this task: we will be able to learn the basics of machine learning and to create our own neural network.

2.2 Purpose and Objectives

The goal of the project is to explore the properties, aspects and characteristics of neural network and the subject area. Create mathematical models and prototypes of neural networks based on various algorithms and operating principles that can determine the winning combination in the player's "hand". Analyze models‘ performance, efficiency, and correctness of the task.

Objectives of the project:

- Study the principle of operation and examples of already created mathematical models and neural networks for a board game.
- Dig into the study of Machine Learning.
- Learn the principles and stages of creating neural networks.
- Explore different methods of training mathematical models.
- Compare the results of the models trained by different methods.

- Summarize the results, evaluate the positive aspects of the models and their outcomes.

2.3 Methods, algorithms, models, and instruments for project implementation

To begin with, we need to be able to analyze and work with data. For this, we will use the following libraries in Python programming language:

Pandas. This library is needed for data manipulation and analysis. It offers a wide range of structures and operations for working with tabular, multidimensional and time series data. With pandas tools it is really easy to handle with missing information, convert various data types into each other.

Numpy. This library is useful for working with arrays and matrices. Numpy arrays are much faster than lists (that serve the purpose of arrays), as they are stored in one continuous place in memory, comparing to lists. Elements of such array have to be of the same type, as a result, each of them requires the same amount of memory.

Matplotlib is used for visualization in Python. This tool allows to see some correlations and tendencies in huge amounts of information and makes this data ready for a proper interpretation and analysis.

In order to practice the usage of this library, we have decided to plot a few graphs that took data from 2 different datasets: "Indian chess Grandmasters" and "Poker hand". The first dataset contains the list of all Indian Chess Grandmaster players till July 2020 and includes details like the year of becoming a Grandmaster, the year of birth, the birth place of all players, their classical, rapid and blitz ratings. On the graphs below (Figure 1 and Figure 2) the correlation between different types of chess ratings and the number of Indian Chess Grandmasters who are born in a particular year are illustrated.

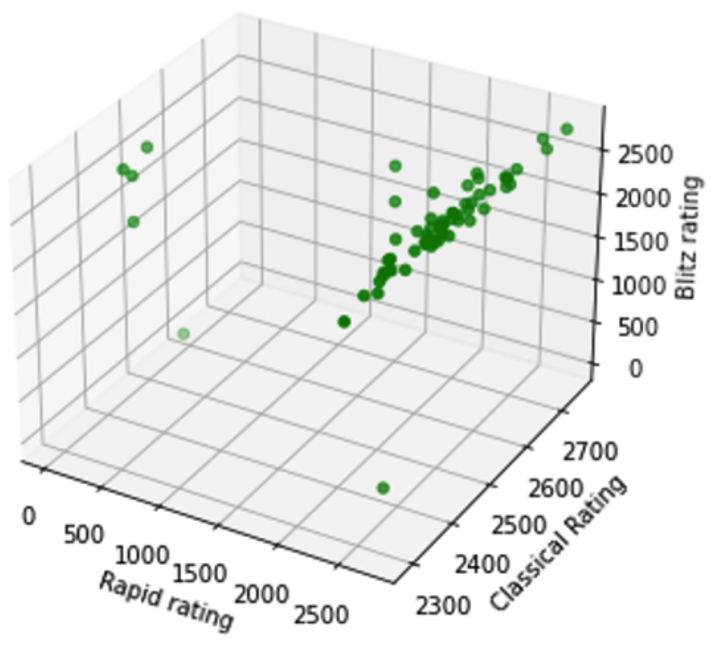

Figure 1: Correlation between Rapid, Classical and Blitz Ratings

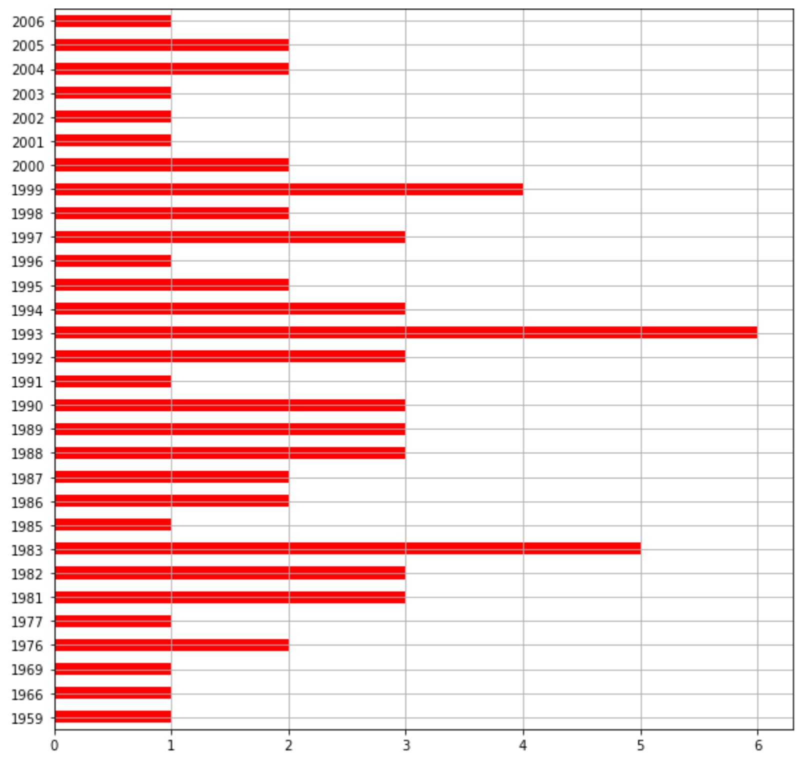

Figure 2: Figure 2: Number of chess Grandmasters born over the period of 1959 - 2006

The second dataset is used for classification algorithms. Each line in the set is an example of a hand consisting of five playing cards drawn from a standard deck of 52. Each card is described by two features: suit and rank. There is one Class attribute that describes the "Poker Hand". On the graph below (Figure 3) the distribution of poker combinations of a given dataset can be seen.

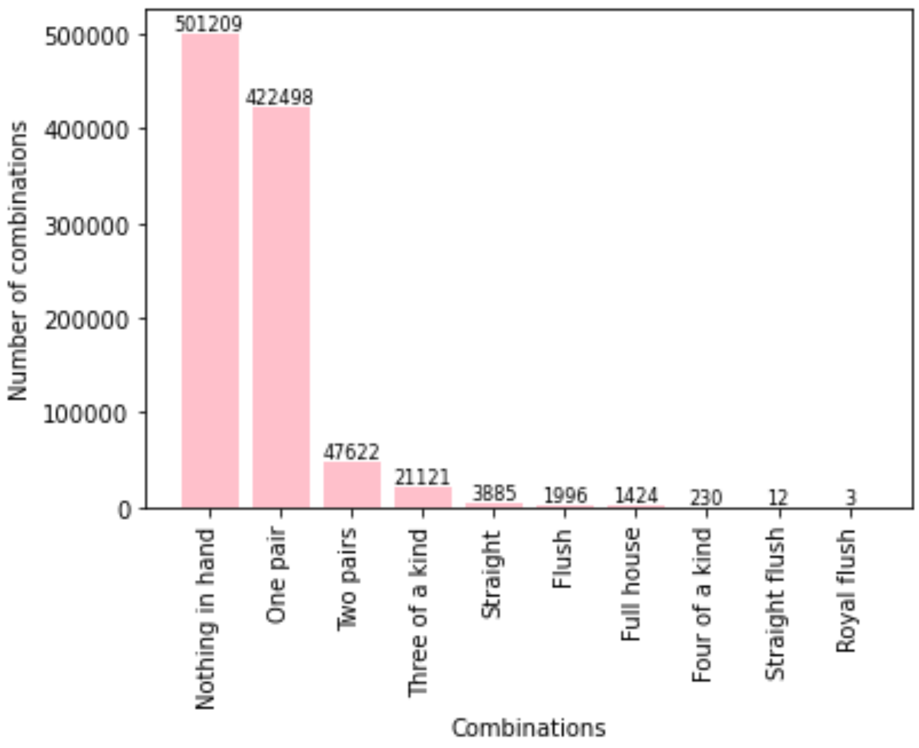

Figure 3: Number of possible combinations

Scikit-learn is a Python module for machine learning. It provides simple and efficient tools for data mining and data analysis. This library is built on NumPy, Scipy and Matplotlib. With its help a user can sort and process data and implement numerous algorithms for Machine Learning and solve problems regarding classification, regression and clusterization of data. In addition, using this library it is possible to create a MLP (Multi-layer perceptron) classifier. It is an algorithm of a network training with the use of a small number of layers, therefore, making the network run with high speeds.

Keras is a Python library that is widely used for the development and evaluation of deep learning models. It is based on efficient Theano and Tensorflow computational algorithms that allow to train user's neural networks by only a few lines of code. Keras sequential model API is useful to create simple neural network architectures without much hassle. The only disadvantage of using the Sequential API is that it doesn't allow us to build Keras models with multiple inputs or outputs. Instead, it is limited to just 1 input tensor and 1 output tensor.

In order to train our models, such methods and algorithms will be applied:

Linear Regression

In general, regression analysis is one of the most frequently-used statistical techniques. It focuses on modeling an explicit relationship between one dependent variable, often denoted as y, and several regressors (also called independent variables), usually denoted as $x_1, \ldots, x_p$. The goal of regression is to understand how y depends on $x_1, \ldots, x_p$. In such model it is assumed that this relationship is linear, so, the model takes the following form: $y_i = \beta_0 + \beta_1 x_{i_1} + \ldots + \beta_p x_{i_p} + \epsilon_i$,where β_j and ϵ_j are the parameters of the model that are learned. Linear models make predictions using a linear function (that was described before) of the input features.

Naive Bayes Classifier

This Classifier is similar to the Linear Regression models, sometimes this classifier performs even faster than the one that was discussed before. However, the accuracy may suffer, as Naive Bayes algorithm looks at each feature individually. This method is based on applying Bayes' probabilities theorem with the naïve assumption of conditional independence between every pair of features that define a certain class. In simple words, each feature is unrelated to the existence of any other feature of a particular class. Such type of classifier is widely used among various real-life problems solutions, since they require a small training data in order to estimate correctly all the needed parameters. Moreover, Naïve Bayes Classifiers are considered to be faster than many other training algorithms. This approach consists of calculating probabilities for each class having a particular list of features. The class which has the highest probability is the output of the algorithm. For classification problems, Gaussian Naïve Bayes model should be applied: it assumes that all the features of a particular class are normally distributed and stores the average value as well as the standard deviation of each feature for each class.

K-Nearest Neighbors Classification and Regression

Both of these algorithms assume that similar objects (in a sense of features) belong to the similar classes, in other words, similar objects are near to each other. KNN has the following principle of working: the algorithm stores all the features and the related classes. All the new input will be based on measuring similarity of a new case to the old ones, that is, the distances will be measured. The new object will be of the same class as the most common one among k of its neighbors. In general cases, huge k's should be preferable, as the overall noise will be decreased.

Classification KNN algorithms are applied in the cases where the target values are discrete, while the Regression KNN is needed for values that are real numbers (continuous variable).

Decision Tree Classification and Regression

A Decision Tree is a supervised ML model to predict the output by learning decision rules from features. In simple words, the key principle of this algorithm is connected to breaking down the training data by some particular features that were chosen. Classification Decision Tree algorithms are needed for discrete values prediction, while the Regression DT is applied for values that are real numbers (continuous variable). In Decision Tree algorithms leaves represent class labels, while branches represent combination of features that lead to those class labels.

Decision Tree algorithm is equipped with the following hyperparameters that will be changed in the following research: *criterion*, *splitter* and *max_ depth.* criterion is a function that measures the quality of a split and in Classification problems it is taken as entropy (which is applied for information gain). The entropy equals to 0 if all the nodes are of the same class, other way around, it is maximal in the cases when all the classes are equally likely (i.e., uniformly distributed). Splitter hyperparameter chooses the way nodes are split at each stage. best splitter uses evaluation before splitting each node, while random one does it just randomly at each stage. As for max_depth, in the most cases the depth of a decision tree is limited from the above in order not to get the overfitting.

Random forest algorithm

In simple words, Random Forest is a collection of decision trees – this helps to solve the problem of overfitting data. RF algorithm works in the following way: we assume that different decision trees (that are the components of RF algorithm) will overfit on some input data at some moment. But since we have many of them, each of them will be overfitting at different places. As a result, the results of prediction will be averaged, and the accuracy may increase.

MLP Classifier

A multilayer perceptron (MLP) is a type of feedforward artificial neural network (ANN). The simplest MLP consists of at least 3 layers: the input, hidden and output layers, which have nonlinear activation functions, except for the input layer. Actually, this structure make it possible to work with nonlinear data, which is the key difference between MLP and linear perceptron.

2.4 Main result

The main result of the project is to study Machine Learning, basic principles of neural networks operation and details of their implementation. Another point is to learn how to create different mathematical models that are able to determine a correct poker combination from the perspective of the given dataset with a set of 5 cards and initially known correct combination for each set. The main idea of the research is to compare the models' and networks' accuracy and analyze the reason for it.

2.5 The role of members of the research project

Each participant makes the same contribution to the preparation, development and creation of the final product on every step. However, some parts of the work were divided as follows: **Marianna Rybnikova** (Group 191) is responsible for data transformation and exploration of different mathematical models (i.e., Linear Regression, Naïve Bayes, K-Nearest Neighbors, Decision Tree and Random Forest). **Dmitry Kurbatov** (group 191) will focus on "Keras" Neural Network. He will study the correlation between various parameters and accuracy of the final NN. **Gleb Zotov** (group 191) will be working with "Scikit-Learn" Neural Network and on Multi-Layer Perceptron Classifier. He will conduct the analysis of the NN parameters.

3 Overview and Comparative Analysis of Sources

3.1 Chess

To learn how to play each game, an untrained neural network plays millions of games using "self-play" approach: through the process of trial and error called reinforcement learning. At first, it plays completely randomly, but over time the system learns from wins, losses, and draws to adjust the parameters of the neural network, making it more likely to choose advantageous moves in the future. As a result, the system is updating continually, taking approximately 9 hours for AlphaZero to be trained for a chess game. But anyway, it all depends on the complexity and the style of the game.

3.2 Checkers

Samuel's Checkers Player is the first machine learning system that received public recognition. He published some studies connected to machine learning using the game of checkers in 1967.

The method relied in part on the use of a polynomial evaluation function comprising a subset of weighted features chosen from a larger list of possibilities. One of the learning techniques is connected to remembering positions that it frequently encountered during play, while the polynomial was used to evaluate alternative board positions some number of moves into the future using a minimax strategy. Another one is based on self-learning procedure: one player is competing against others. The loser was replaced with a deterministic variant of the winner by altering the weights on the features that were used, or in some cases replacing features that had very low weight with other features.

3.3 Poker

Poker is not left without any neuron networks. A program called "DeepStack" was created in 2017 and practically immediately after creation won a game with professional players at heads-up poker. The main challenge in this game is the fact that a winning strategy depends also on intuition and a chance success, therefore, it is considered to be "imperfect" and for the AI it is rather difficult to find out the correct decision.

The system is based on the "continual re-solving" approach: it uses only those cards that are left in the game (and the ones that are in the hand), without remembering the full game, as it leads to lower overall exploitability. Moreover, the “intuitive” local search is applied: DeepStack does not provide any reasoning about the full remaining game. Instead of this, the system does a fast-approximate estimate and considers only a reduced number of actions allowing it to play at human speeds.

4 Project Implementation

4.1 Plan for Calculation Experiment

In our research we want to compare and analyze the accuracy of several mathematical models and algorithms in order to figure out the most effective one. We have the following steps to work on:

- Implementing mathematical models
- Converting the original dataset into a more convenient (binary) form for analysis
- Compare results of each model
- Determine why some algorithms are more precise than others

4.2 Database description

In our project we use dataset of 5 poker cards as an input and 1 combination as an answer. Each card presented as 2 numbers, one from 1 to 4 for each suit (e.g. 1 - for spade, 2 - for heart etc.) and one from 1 to 13 for each rank ("1" goes for 2, "2" goes for 3, ... "12" for "King", and "13" goes for Ace). The overall dataset consists of 1.000.000 strings with 11 number in each, first two numbers represent the first card, then the next two numbers are responsible for the second card representation, etc. The last number of each line is the best combination that can be received from the 5 cards that were described before. The data is split in 8-by-2 proportion, so that 800.000 strings of the csv-file are used to train data and 200.000 strings are needed for testing.

For some models, especially working with *Scikit-Learn* library, we introduce another kind of data representation. We wanted to transform our dataset in such way that each card is described by only one feature (and in the initial dataset it is formed from a "rank" and a "suit"). To do this, each card was represented by seventeen "0". After that the needed suit or rank was marked with "1". Therefore, each card was encrypted as a set of two "1" and fifteen "0", so, it was transformed into a binary format and written in a csv-file in a decimal notation. Apparently, a player‘s hand consisted of 5 numbers, each number had exactly two "1" in its binary form.

4.3 Implementation of models, algorithms and neural networks

Linear Classification and Naïve Bayes.

We decided to begin our exploration of various Machine Learning algorithms from the simplest ones. It was clear that the performance of these algorithms will not be perfect, since LC is based on a principle that there is a linear dependency between the features and the target class. Surely, it is not true in our case. Since the transformed data consists of numbers with exactly two "1" in their binary representation, we cannot simply find 6 numerical parameters that would lead us to the true result: the dependency here is much more complicated.

The same can be mentioned about Naïve Bayes algorithm. Since it is somehow similar to the previous one, it will not be correct to receive completely different results through the usage of this approach. The problem here is as following: Naïve Bayes approach assumes that all the features are independent between each other, but they are dependent actually. For instance, if the first card is Queen Hearts, then remaining ones cannot be of the same rank and suit. But Naïve Bayes says, they can.

K-Nearest Neighbors Classification and Regression

We have decided to try both of these algorithms in order to understand which of them is the most accurate on the given training data. Before the data was transformed, the accuracy was around 45% – and this is in fact a logical result, as the algorithms could not understand the relations of features (since “a feature” in our case is a combination of a rank and a suit). After the data was transformed and sorted in the ascending order (the first feature represents the weakest card, the fifth feature is related to the strongest card of a “hand”) the performance grew immediately and was equal to 92% for both algorithms.

The next step was to find the optimal number of neighbors for KNN Regression and Classification. We wrote a cycle that was running from 1 to 30 neighbors (with a step 3) and calculating the Mean Squared Error at each stage. As a result, both algorithms had the least MSE for $k = 4$ neighbors. The performance of KNN Classification and Regression can be seen on a line graph below (Figure 4).

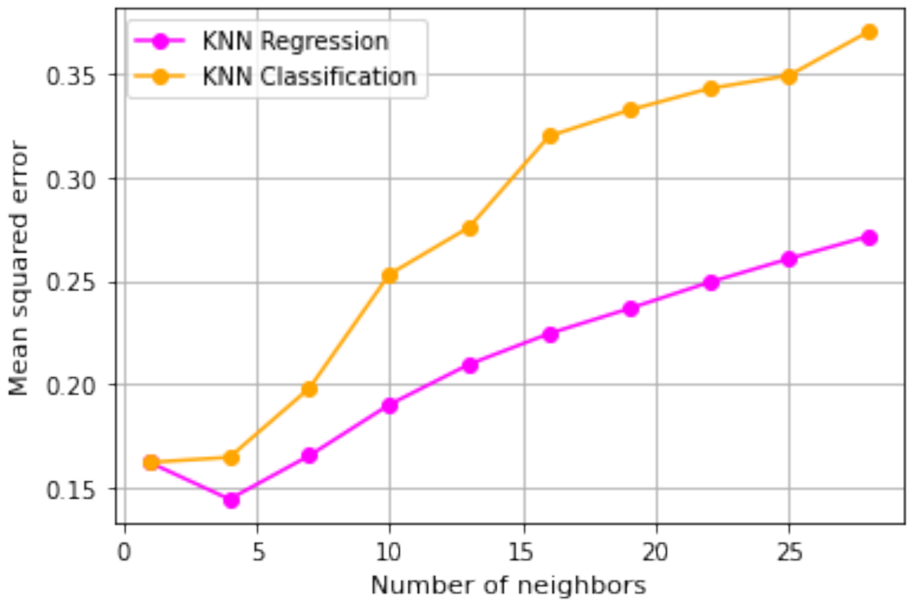

Figure 4: Correlation between MSE and number of neighbours in KNN algorithms

Decision Tree Classification and Regression

In order to check whether our Classification and Regression Decision Tree models are generalizable and to check what maximal depth will be optimal for our DTs, we fitted our models and plotted a graph that showed the dependency of Mean Squared Error and the depth of the Decision Tree. As a result, we received the following picture which shows that in general, Regression algorithm works more accurately for small values of the parameter, while the Classification algorithm shows better results for bigger parameter values (Figure 5).

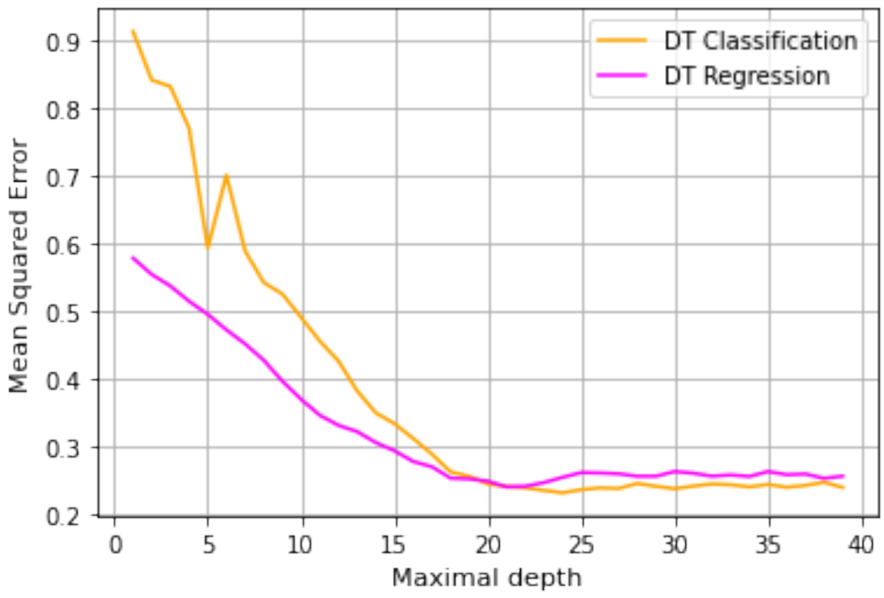

Figure 5: Correlation between MSE and maximal depth of the DT

Random Forest algorithm

At first, when Random Forest Classifier algorithm was applied with "raw" data, the accuracy was around 62%. The model was mistaken by a half of the class in general (Mean Absolute Error was equal to 0.52). After data transformation, we decided to add one more algorithm: the regression one and compare the results as in the previous models. So, as it can be seen from the graph below, the Regression RF is more accurate on smaller values of the parameter *max_ depth*, as for bigger values, both algorithms have generally the same result (Figure 6). Talking about accuracy, Classification Random forest showed a better target prediction.

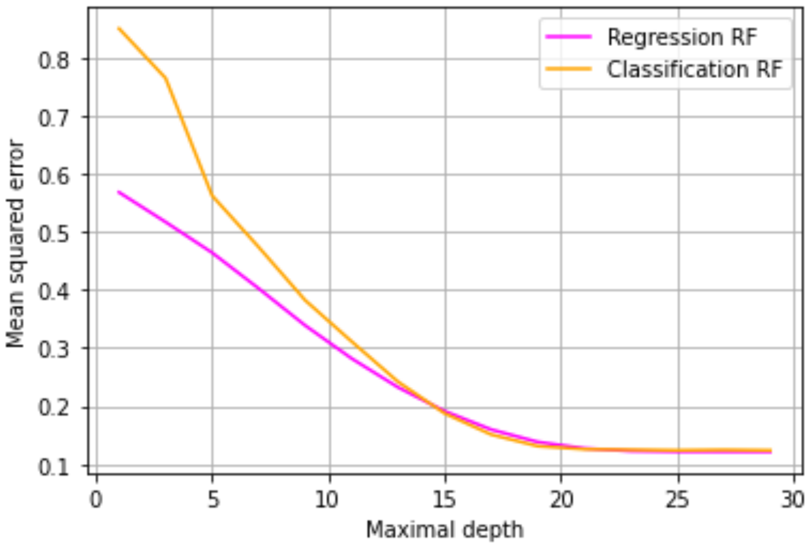

Figure 6: Correlation between MSE and maximal depth of RF

Keras neural network

We decided to create a Keras neural network initialized with *Sequential()* class that is created for feedforward neural networks (i.e. networks in which each layer has a direct input from another layer). In our case there are two layers. Such a choice is due to the moderate size of the sample. Keras neural networks are very powerful and are created to work with hundreds of millions of elements in a sample and a dataset of a million elements seems to be too small for more than two layers of neurons. In addition, a network has to include more layers in case the data we want to obtain is described by a very complex and abstract function, that is also not our case. First layer has density of 1000 neurons because sample size can be divided by 1000 exactly and therefore, each group of neurons will consist the same quantity of them. The second layer is fitted with 10 neurons as it needs to have the same size as the the number of elements in the vector of possible combinations (there are 10 of them). We tried to fit the model with different density of the first layer, however, the highest accuracy was obtained exactly with 1000 neurons. In the picture below (Figure 7) a correlation between the density of the first layer and the highest accuracy of the network.

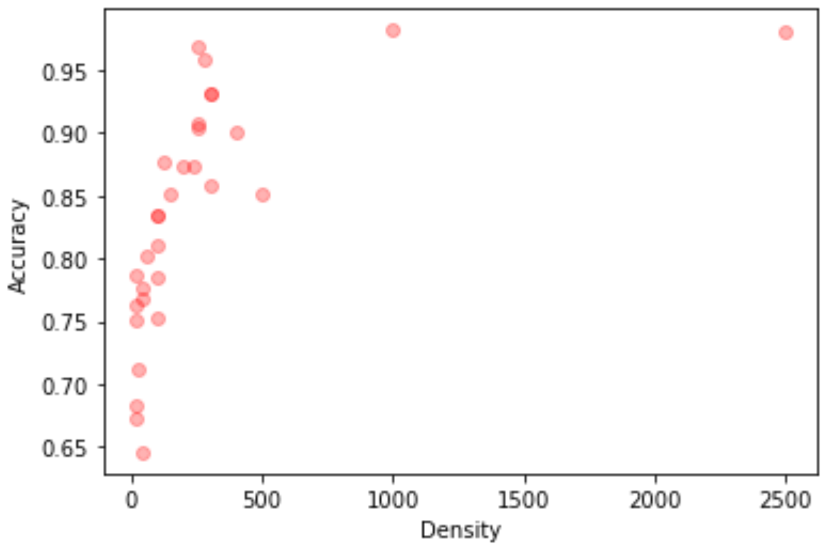

Figure 7: Correlation between density and accuracy of the NN, range 0 : 2500

We can see that the accuracy increases linearly with the density change until approximately 250 neurons, after which there is a decrease in the precision. The linear correlation can be seen in the picture below (Figure 8), where the range of the density value is limited above by 450.

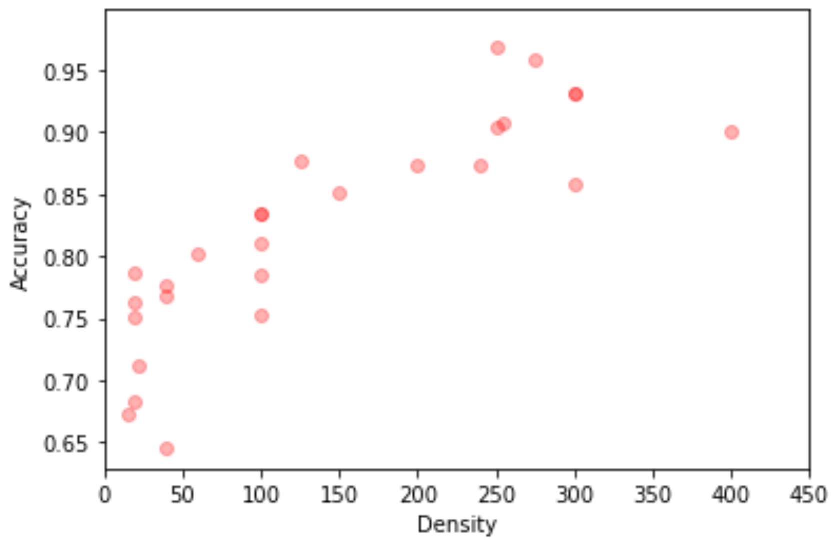

Figure 8: Correlation between density and accuracy of the NN, range 0 : 450

Besides layers‘ density there are three variable parameters left: activation function, optimizer and loss function. The last one is chosen to be a *categorical_ crossentropy* rather than *binary_ crossentropy* because in our case all poker combinations create a set of parameters that can be easily ordered and in addition, they do not make a binary classification.

By empiricism we obtained the maximum accuracy of 98.17% using *“sigmoid”* activation function for the first layer and *“softmax”* for the second one and *“adamax”* optimizer. In order to prove this is the best combination of an activation function and an optimizer we conducted a small research. In the first case the activation function was constant and the only variable parameter was optimizer. In total 7 optimizers were explored: *Stochastic Gradient Descent (SGD)* optimizer, *"RMSProp"* optimizer, *"adagrad"*, *"adadelta"*, *"adam"*, *"adamax"*, *"nadam"*. The best effectiveness was given by an *“adamax”* optimizer and the worst by *SGD*, this can be seen on the bar chart below (Figure 9).

The tendency can be explained by the origin and the nature of the algorithms. *SGD* is considered to be the roughest optimizer as it is the simplest one with the biggest error and therefore, in case of a complex function the network will not be able to find the correct solution. That is why, *"adamax”* that is a deep improvement of the *SGD* algorithm has a possibility to deal with challenges occurring while training.

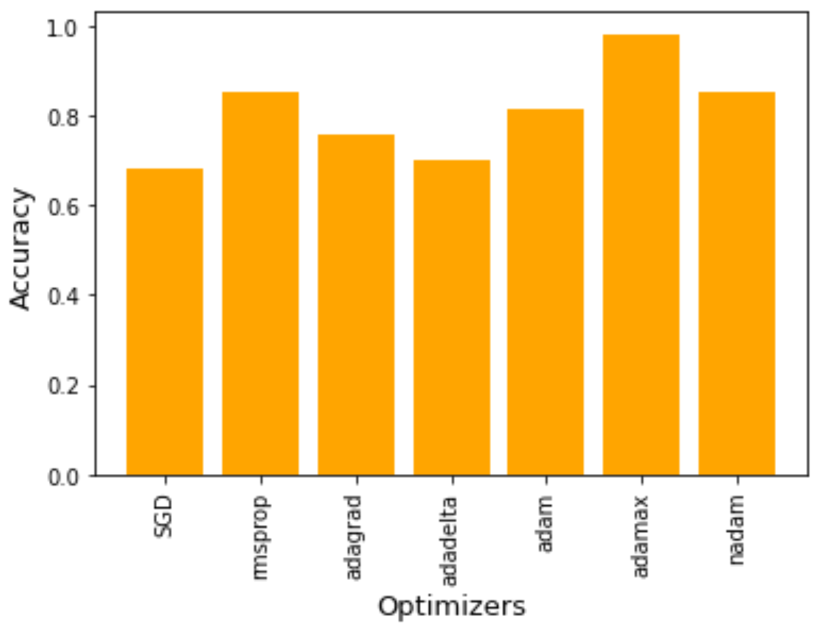

Figure 9: Accuracy of optimizers

"RMSProp" optimizer also obtains rather good results (approximately 88%), however, accuracy an losses fluctuate severely during training (Figure 10) and therefore, entire results may be less precise. This happens because of the working principle of the optimizer. Its updating function that influences the obtained result relates on the square of gradients, consequently, small changes in the gradient may cause high bias of the accuracy of the optimizer. A similar tendency is captured while using the *"Adadelta"* optimizer that has the same nature as the *"RMSProp"*.

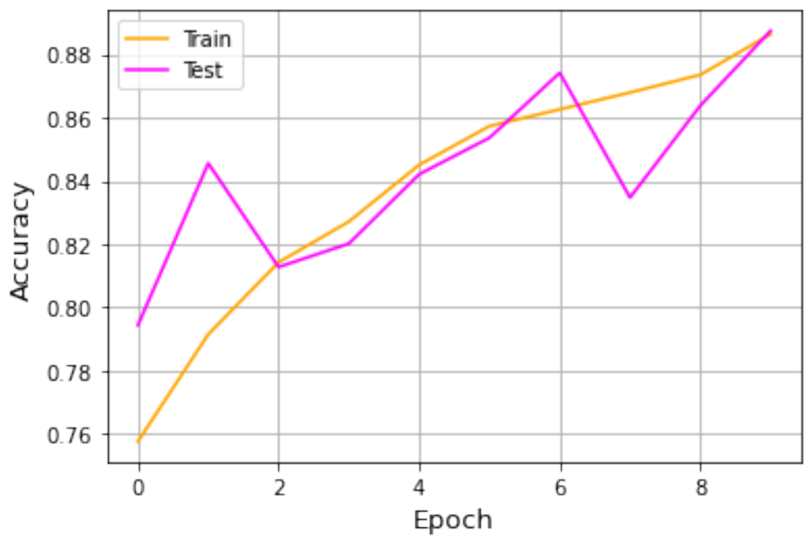

Figure 10: Accuracy of the NN with "RMSProp" optimizer

The next step for us was to compare different activation functions and ensure that *"sigmoid"* is indeed the most effective function. In total we observed 8 activation functions with the only one optimizer - *"adamax"*. Functions are: *"selu"*, *"elu"*, *"relu"*, *"tanh"*, *"sigmoid"*, *"hard-sigmoid"*, *"linear"*, *"exponential"*.

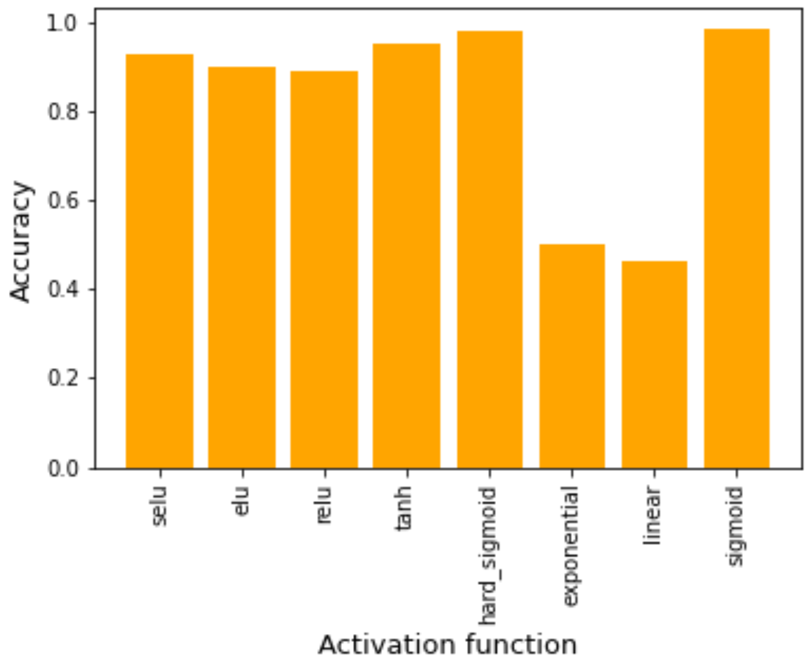

Figure 11: Accuracy of functions

As can be seen from the bar chart (Figure 11), the least precise activation functions are *"linear"* and *"exponential"* with the use of which the neural network does not reach even 50% of accuracy, apparently, randomizing is more precise. However, other activation functions affected positively on the training of the neural network and practically every network using one of these 6 activation functions succeeded in guessing the combination with total accuracy of more than 90%. However, the most precise activation function is *"sigmoid"*. Sigmoid is a function of form:

$$\phi(\zeta) = \frac{1}{1 + e^{-\zeta}}$$

,

where $\phi()$ is an activation function and ζ is an input value. Sigmoid function resembles a logistic distribution as while $\zeta \longrightarrow -\infty \quad \phi(\zeta) \longrightarrow 0$ and $\zeta \longrightarrow +\infty \quad \phi(\zeta) \longrightarrow 1$. Thus, for our case *"Sigmoid"* and *"Hard-sigmoid"* activation functions that are both based on the sigmoid function is apparently the most suitable type of an activation function.

To sum up, the Keras neural network consisting of 2 layers can guess a correct combination of 99.09% of poker hands from the given dataset (Figure 12). Simultaneously, losses obtained by the work of the network are also very appreciable and reach 0.0225 points that is considered to be a great result (Figure 13).

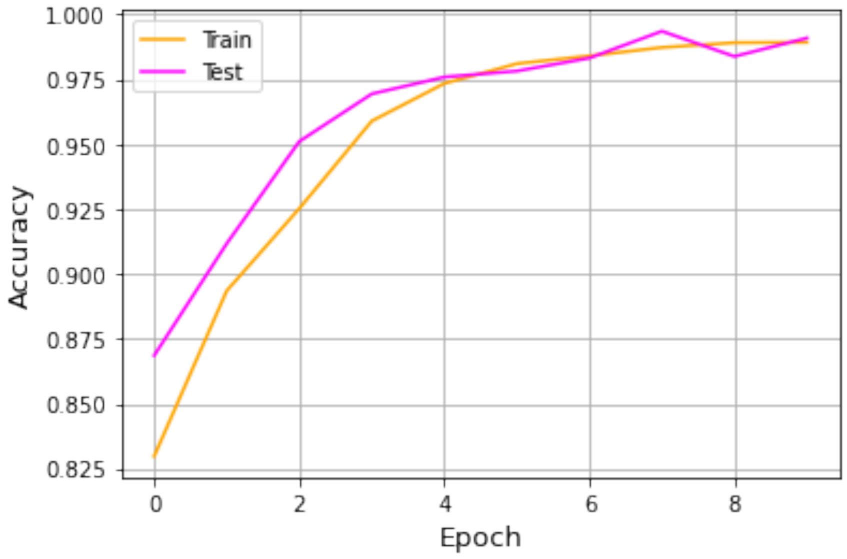

Figure 12: Keras model accuracy

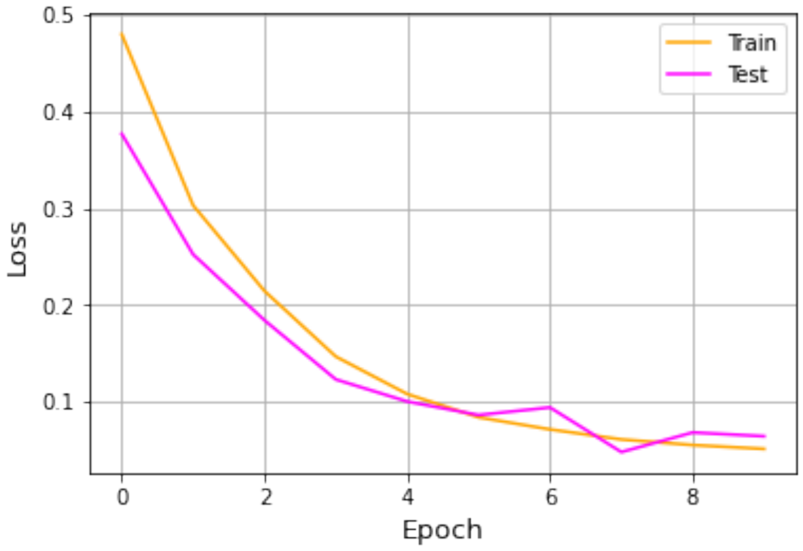

Figure 13: Keras model losses

MLP Classifier neural network

The process of learning is based on the backpropagation technique, the idea assumes that the first half of the iteration (epoch) the input data passes through the hidden layers and outputs the value according to the set weight. Then, the answer is checked and the error is estimated, according to this error the second half of the epoch goes from the output layer to the input and changes weights, in our case the alpha and solver parameters represents the penalty for the weights and weight optimization algorithm respectively. Therefore, if we improve weights we will obtain better results.

By empiricism we have determined the best bundle of parameters that gives the highest accuracy. Parameters are as following:

Parameter	**Type**
Activation function	*Tanh*
Solver	*Adam*
Hidden layer sizes‘	3 x 100
Learning rate	Adaptive
Max. iter	300
Penalty for weights (α)	0.0001

According to Jeff Heaton a single hidden layer allows the neural network to approximate any function involving a "continuous transformation from one finite space to another". So, the MLP with only 1 hidden layer is already a strong instrument for classification, however approximately 3 hidden layers is recommended as the standard. That is why our neural network has 3 hidden layers. The graph below (Figure 14) shows the accuracy according to the size of the hidden layers.

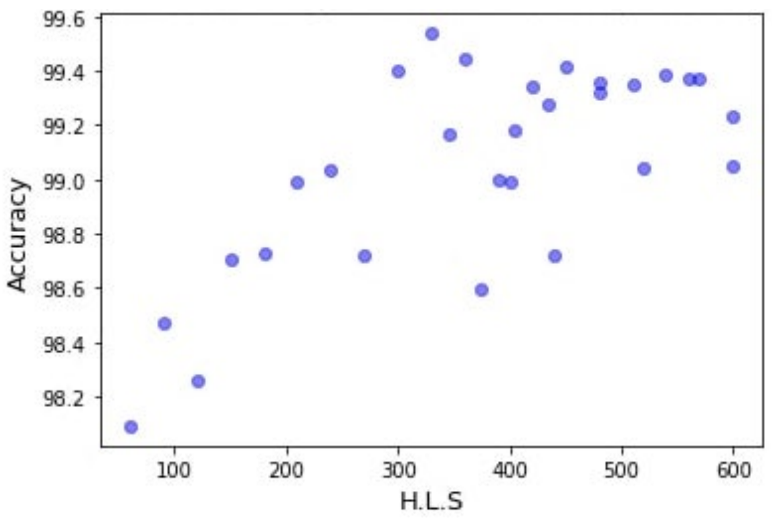

Figure 14: Correlation between hidden layers‘ sizes (HLS) and accuracy of the NN

The results are similar and approximately equal 99% of accuracy, however the best results were shown at size of 110 (330/3) with 99.5% of accuracy. The reason for this is the usage of 3 hidden layers, which provide complicated structure of neurons and optimize weights precisely. Now, let us compare the work time and losses for each hidden layer size (Figure 15, Figure 16).

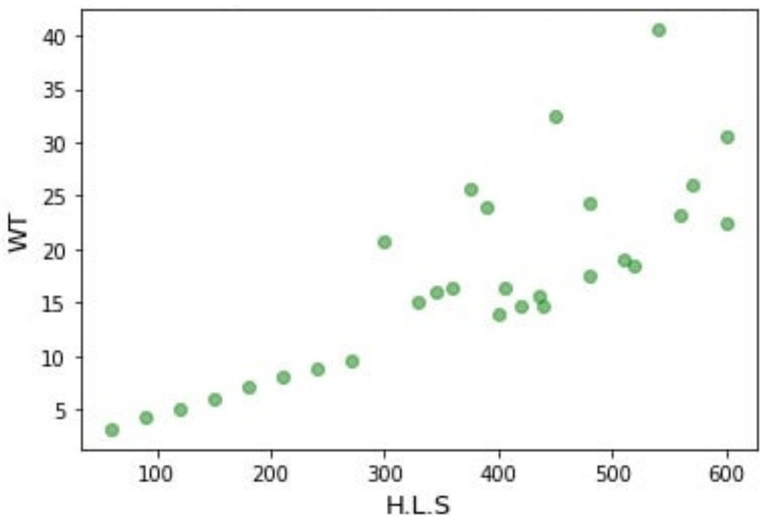

Figure 15: Correlation between hidden layer sizes‘ and work time of the NN, range 10:600

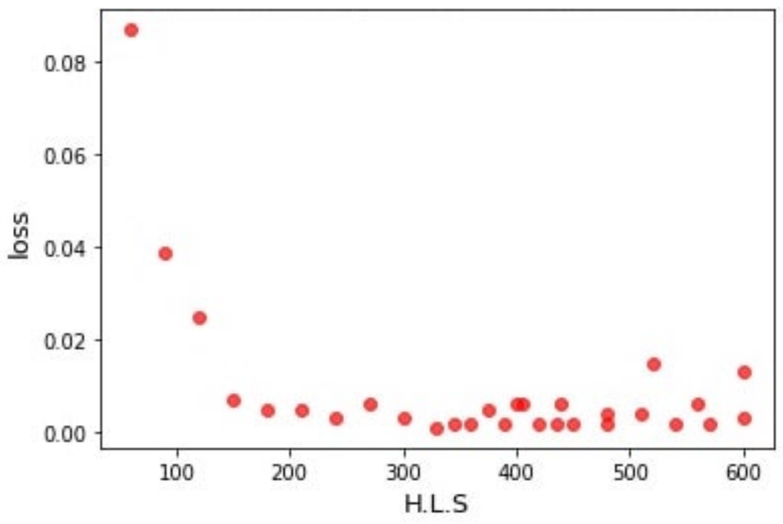

Figure 16: Correlation between hidden layers sizes' and losses of the NN, range 10:600

The activation function for our NN was chosen also by the series of the experiment, as 'logistic' and 'identity' show unsatisfied results, the choice was between *ReLu* and *Tanh* functions. In fact, *ReLu* is stated as the best and universal activation function, however, for each task it is better to test different combination to determine the most efficient. So, by taking key points with size 10, 50, 100, 130 and 150 the diagram below (Figure 17) proves the advantage of *Tanh* activation function.

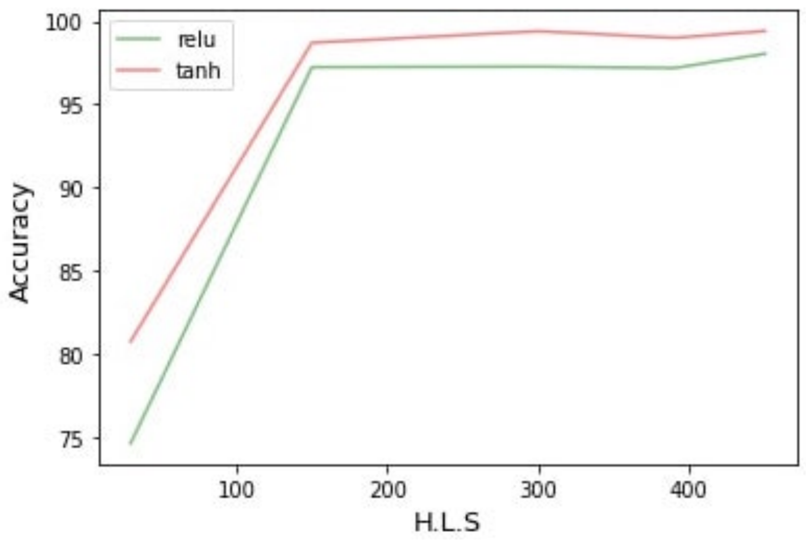

Figure 17: Difference between *ReLu* and *Tanh* activation functions

The reason to choose *Adam* solver is similar to the process of choosing activation function. Here are the results of the experiment (Figure 18).

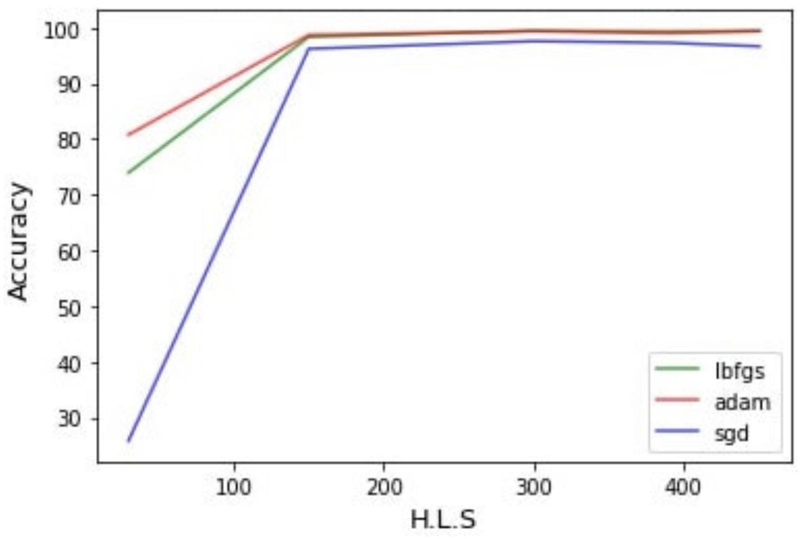

Figure 18: Difference in accuracy among solvers

Jeff Heaton in his research, states that *Adam* solver works precisely for huge datasets, while the *lbfgs* (optimizer in the family of quasi-Newton methods) is better for small subsets. Since the data set consists of 1.000.000 samples the *Adam* solver is more efficient.

All in all, the NN provides the best result of 99.5% accuracy using 3 hidden layers with size of 110, *Tanh* activation function, *Adam* solver and $\alpha = 0.0001$.

4.4 Key conclusion

As a result of our project we have studied, constructed and improved several algorithms that are able to solve the same task: guess the best poker combination out of 5 cards. We compare all the received output and develop the following accuracy:

Algorithms	**Accuracy, %**
Linear Regression	51%
Naïve Bayes	55.7%
K-Nearest Neighbor Classification	93.38%
K-Nearest Neighbor Regression	92.4%
Classification Decision Tree	95.11 %
Regression Decision Tree	95.19%
Random Forest Classification	77.69%
Random Forest Regression	80.33%
Keras neural network	99.09%
MLP Classifier neural network	99.5%

Overall, the best results are shown by neural networks with 99,09% and 99,5% of accuracy. Talking about the performance of mathematical models, Regression Decision Tree and KNN Classification algorithms were the most accurate ones with the accuracy of 95.19% and 93.38% respectively (Figure 19).

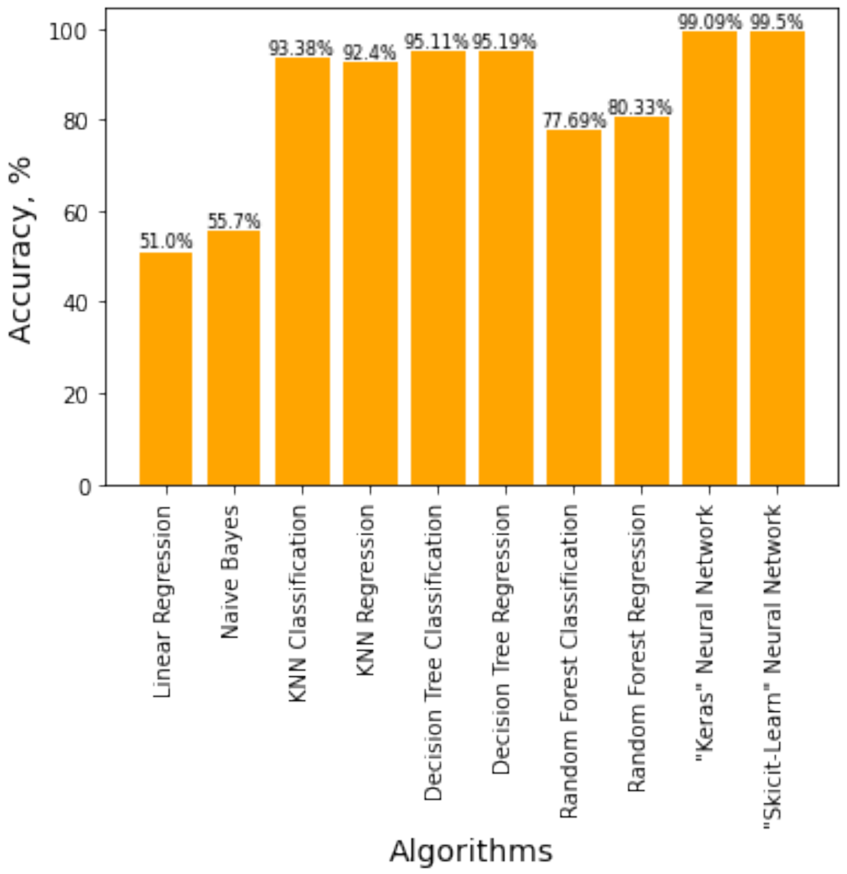

Figure 19: Accuracy of the applied algorithms

5 Bibliography

1. Andreas, C. Introduction to Machine Learning with Python / C. Andreas Müller, Sarah Guido – USA: O'Reilly Media, Inc., 2017 – 394 p.

2. Моя шпаргалка по Pandas [Electronic Source] / Habr. Available at: https://habr.com. (Accessed: 09.01.21)

3. Scikit-Learn: Machine Learning in Python [Electronic Source] / Scikit-Learn. Available at: http://scikit-learn.org. (Accessed: 20.01.21)

4. Cross-validation: evaluating estimator performance [Electronic Source] / Scikit learn. Available at: https://scikit-learn.org/stable/modules/cross_validation.html. (Accessed: 29.01.21)

5. Loss and Loss Functions for Training Deep Learning Neural Networks [Electronic Source] / Machine Learning Mastery. Available at: https://machinelearningmastery.com. (Accessed: 06.02.21)

6. Pyplot tutorial [Electronic Source] / Matplotlib. Available at: https://matplotlib.org. (Accessed: 08.02.2021)

7. Как устроен шахматный мозг AlphaZero? [Electronic Source] / Chess.com. Available at: https://www.chess.com. (Accessed: 10.02.21)

8. Expert-Level Artificial Intelligence in Heads-up No-Limit poker [Electronic Source] / DeepStack. Available at: https://www.deepstack.ai. (Accessed: 10.02.21)

9. Samuel's Checkers Player [Electronic Source] / Encyclopedia of Machine Learning. Available at: https://link.springer.com. (Accessed: 14.02.21)

10. NumPy [Electronic Source] / Python 3 для начинающих. Available at: https://pythonworld.ru. (Accessed: 19.02.21)

11. Hello, TensorFlow. Библиотека машинного обучения от Google [Electronic Source] / Habr. Available at: https://habr.com. (Accessed: 23.02.21)

12. XGBoost documentation [Electronic Source] / XGBoost. Available at: https://xgboost.readthedocs.io. (Accessed: 24.02.21)

13. When to use MLP, CNN, and RNN neural networks [Electronic Source] / Machine Learning Mastery.
Available at: https://machinelearningmastery.com. (Accessed: 09.03.2021)

14. Developer guides [Electronic Source] / Keras. Available at: https://keras.io/guides/. (Accessed: 15.03.2021)

15. A Gentle Introduction to Scikit-Learn: A Python Machine Learning Library [Electronic Source] / Machine Learning Mastery.
Available at: https://machinelearningmastery.com/sk-learn. (Accessed: 20.03.2021)

16. Методы последовательной модели [Electronic Source] / Keras Документация.
Available at: https://ru-keras.com/sequential/. (Accessed: 27.03.2021)

17. A Gentle Introduction to Keras [Electronic Source] / Kaggle.
Available at: https://www.kaggle.com/keras. (Accessed: 01.04.2021)

18. sklearn.neural_network.MLPClassifier [Electronic Source] / ProgramTalk.
Available at: https://programtalk.com/MLPClassifier/. (Accessed: 03.04.2021)

19. sklearn.linear_model.LogisticRegression [Electronic Source] / Scikit-Learn. Available at: https://scikit-learn.org/LogisticRegression/. (Accessed: 08.04.2021)

20. Support-vector machine / Wikipedia.
Available at: https://en.wikipedia.org/wiki/Support-vector_machine. (Accessed: 11.04.2021)

21. Краткий обзор алгоритма машинного обучения: Метод Опорных Векторов (SVM) / Habr. Available at: https://habr.com. (Accessed: 18.04.2021)

22. The Number of Hidden Layers [Electronic Source] / Heaton Research.
Available at: https://www.heatonresearch.com. (Accessed: 14.05.2021)

23. Scikit-Learn - Neural Network / CoderzColumn.
Available at: https://coderzcolumn.com. (Accessed: 23.05.2021)

24. Методы оптимизации нейронных сетей [Electronic Source] / Habr.
Available at: https://habr.com/en/post/318970/. (Accessed: 05.05.2021)

6 Appendix

6.1 List of figures

6.2 Project Work’s Calendar Plan

November 2020:

- Selection of the project and the topic
- Choice of a mentor and negotiation with him
- Submission of the application form

December 2020:

- Analysis of the thematic materials and learning the basics of the topic
- Attending the Coursera courses on Machine Learning

January 2021:

- Graph plotting and working with data: pandas, matplotlib, numpy.
- Based on the received knowledge, plotting graphs according to the data given by the mentor.
- Learning python libraries for machine learning (Scikit-Learn)
- First attempts to train the model

February 2021:

- Preparation of the materials for CP1
- First attempt to create a "Scikit-Learn" neural network
- Development of mathematical models and algorithms for card combination prediction

March 2021:

- Exploring "Keras" library for Python
- Development of "Keras" neural network

April 2021:

- Creation and development of several "Scikit-Learn" neural networks
- Accuracy comparison in different mathematical models and neural networks in the context of combination prediction
- Analysis of obtained information
- Preparation of the materials and documents for the CP2

May 2021:

- Analyzing the results: comparison of the received accuracy and graphs plotting
- Preparation for the final CP