University of Glasgow | School of Computing Science

Predicting Horse Race Outcomes

Avinab Shah

School of Computing Science

Sir Alwyn Williams Building

University of Glasgow

G12 8RZ

A dissertation presented in part fulfillment of the requirements of the Degree of Master of Science at the University of Glasgow

21st January 2021

Abstract

Horse racing is one of the most archaic sports all around the world with the start dating back to the colonial era. The technological developments have impacted the industries across the globe. Horse Race Betting is an emerging investment-based industry which became popular post saturation in the stock exchange market. It accounts for a substantial amount of sports betting markets across the globe, for instance in 2017, Kentucky Derby (annual horse race event) recorded betting worth $139.2 million approximately [1]. With such huge investments, betting industry is growing every single day. We proposed a novel idea to predict the horse race outcomes by using technical and fundamental data simultaneously. Datasets used in this project was provided by roiLab, a Glasgow-based quantitative investment firm, also involved in several sports betting markets. We used two different types of datasets namely Fundamental (macro) data and Technical (micro) data. Fundamental data includes individual attributes like horse's age, weight, sex etc., past performances like races won, best finish time, race gap, jockey's information, trainer's information, and owner's information. While Technical Data is a time-series data comprising of market odds, traded volume in given time interval. For example, traded volume of the horses participating in a race for the past thirty minutes with one minute time step. We explored and investigate several deep learning algorithms such as MLP, CNN and LSTM(for technical data since it is a time-series based data). The naïve ensembled hybrid model achieved an accuracy of approximately 90% on validation and test set.While CNN models using only Fundamental data achieved f1 score, accuracy close to 0.90 on the test set.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic form.

Name: Avinab Shah	Signature: Avinab Shah

Acknowledgements

I would take this opportunity to express my gratitude and thank my supervisor, Dr. Joemon Jose, for his precious mentorship, continuous support, and time throughout the dissertation. I am grateful to him for motivating me in difficult situation, when my laptop crashed, and I suffered from Covid. I am also thankful to School of Computing Science for giving me the opportunity to work on this project.

Contents

Chapter 1 Introduction

1.1 Motivation

Recent developments in machine learning and deep learning have brought revolutions in almost every industry and domain, betting is no exception and one such domain is Horse Racing. It accounts for a substantial amount of sports betting markets across the globe, for instance in 2017, Kentucky Derby (annual horse race event) recorded betting worth $139.2 million approximately [1]. Traditionally, bookies, handicappers did manual analysis to come up with the market odds for horses participating in each race. Their analysis included studying about the past performance of the horses, gap between races, jockey's performance, associated trainer, owners and so on. It is noteworthy that any analysis though thorough does not always lead to correct winner prediction. Gradually, there has been shifts in era from manual analysis to computer analysis to machine/deep learning (AI era) specially the last decade. Though the breakthrough came in the year 1997, when IBM's Deep Blue [2], a deep learning-based neural network model defeated Gary Kasparov, then world chess champion. Since, then deep learning has grown further with its horizon spreading supported by successful stories like AlphaGo [3], Texas hold'em poker [4].

Technically, deep neural networks have a proven track record in solving both simple and complex problems such as classification, regression, and time series forecasting etc. [5]. Time series forecasting can be applied to a range of problems [6], few applications of time series forecasting are stock market predictions, weather forecasting, seasonal sales revenue prediction etc. Stock market prediction relies upon several quantitative trading strategies, which involves mathematical computations and number crunching over mammoth amount of data to make informed trading decision [1] . The underlying algorithms of the machine/deep learning models designed to process macro (fundamental) and micro(technical) data to identify profit opportunities [7] did exist decades ago. However, computational power and its availability were limited then. As explained in Moore's law [8], technology evolution has superseded these constraints overtime paving the way for implementation of more complex and computationally expensive models leading to the AI era. It's wide use in the stock trading led to the saturation of profit making which resulted in application of machine learning approaches in the unconventional investment markets such as sports betting [7]. Similar algorithms and AI techniques can be applied to horse race betting because popular sports betting exchanges like Ladbrokes, Betfair have their fundamental theories of pricing and trading, comparable to stock exchanges [9]. We propose the novel idea to exploit deep learning using both fundamental (individual attributes like horse's age, weight, sex etc. and past performances like races won, best finish time, race gap) and technical data (traded volume, market odds time series) to design a neural network model for horse race winner prediction which can subsequently be used to devise betting strategies yielding profits.

[1] https://www.investopedia.com/terms/q/quantitative-trading.asp

1.2 Aims and Objectives

Horse race forecasting is an existing problem and thus research has been done to create models using typical machine learning algorithms for forecasting the winners [10, 11]. These models were trained on fundamental data and features engineered manually which requires domain expertise. Thus, it is time-consuming and costly. The classifiers based on hand-crafted features often fail to generalize well [11]. Further, they also did not capitalize on the technical data which could have been quite vital in enhancing its performance to predict horse race winners.

The aim of this project is to fill these gaps by developing a novel horse race prediction classifier which factors both fundamental and technical data simultaneously to forecast winners. This forecasting could then be used to optimize strategies to improve gains from the betting. In this capacity, the objectives of the project are as follows:

- i) To design and analyze different horse race prediction models – MLP (Multi-layered Perceptron) and CNN (Convolutional Neural Network) using fundamental data.
- ii) To design and analyze LSTM (Long-Short Term Memory) based neural network using time-series technical data such as traded volume and last traded price.
- iii) To study the effect of multi-headed self-attention on the accuracy of the proposed model variants with an aim to improve its performance.
- iv) To develop a novel, robust deep learning neural network-based classifier by ensembling the embeddings of the fundamental and technical data to predict horse race winners.

1.3 Achievements

We successfully conducted all the experiments to test the hypothesis and subsequently fulfilled all the objectives of the project. The key highlights are as follows:

- i) We implemented a naive ensembled hybrid model to predict horse race outcomes which use both technical and fundamental data. The model achieved an accuracy of approximately 90% on validation and test set.
- ii) We also explored LSTM exclusively for technical data. We developed different model variants using LSTM – multivariate and univariate. The models performed reasonably well with accuracy and precision approximately .32.
- iii) We also used CNN to build models only using Fundamental data. They did surprising well too. On the test set, f1 score, accuracy was close to 0.90.

Chapter 2 Survey

In this chapter, we shall lay down the foundation for the upcoming concepts and its actual implementation. We begin with the literature survey of the associated work done in the past, followed by the critical analysis and finally explanation of the algorithms or models to be used and the research objectives.

2.1 Literature Survey

Horse racing is one of the most archaic sports all around the world with the start dating back to the colonial era [12]. From the start it was viewed as a sport that presented an opportunity of showcasing the strength of an individual by competing in a racing competition of one of the fastest tamable animals in the world. Horse race betting was not too late to follow and betting on horses soon became a popular leisure activity. With the technological advancements, the conventional manual analysis of the factors affecting a horse race were replaced by artificially intelligent prediction models that could predict the winner.

N.M. Allinson and his colleague were the first to apply neural network to predict horse race outcomes in the year 1991. They implemented a fully connected feed forward Multi-layered perceptron (MLP) to predict the outcome of a race, and accordingly bets were placed [13]. They were able to achieve results such as the bets were only laid if the prediction was greater than 0.9, which was notable, being a start in this domain. However, neural network's ability to predict optimal outcomes was unexplored to the fullest due to data size and availability constraint. After around two decadeds in 2016, Edelman D [14] adapted the methodology of Support Vector Machines to predict the odds of horserace. He used various fundamental features such as "finishing position of previous race', bookmaker's odds of previous race, total prizemoney in previous race, final call position of previous race indicator, no call position of previous race, distance of previous race odds * prizemoney of previous race, weight carried in previous race, days since previous race, total prizemoney in current race, weight carried in current race, distance of current race". They achieved an accuracy of 48% for the training set, and 45% for the test set which was higher than applying linear models on the same data. However, they were unable to find the key feature set affecting the race outcome.

In the year 2008, Williams J. A applied four Neural Network algorithms namely, Back-propagation, Quasi Newton, Levenberg-Marquardt, and Conjugate Gradient Descent, to the data collected from 143 races held in Jamaica in 2007. The main achievement of this study was that it achieved an increased average accuracy of 74% but fell short in terms of training time and parameter selections [15]. Later Elnaz Davoodi applied the same neural network algorithms as Williams, with the addition of Backpropagation along with momentum to the data collected from AQUEDUCT Racetrack in New York that included the race data from 100 races [16]. This study was able to demonstrate that Neural Networks have the potential to forecast horse race outcomes with an accuracy of around 77% as compared to Williams J. A [15]. Additionally, the experimental results described in Figure 1 indicate that back propagation (BP) algorithm is comparatively better than other neural network algorithms and the predictions made were precisely close to the actual outcomes which was welcomed by the betters then. However, the training time of the best performing model (BP) was still an open issue that needed to be addressed.

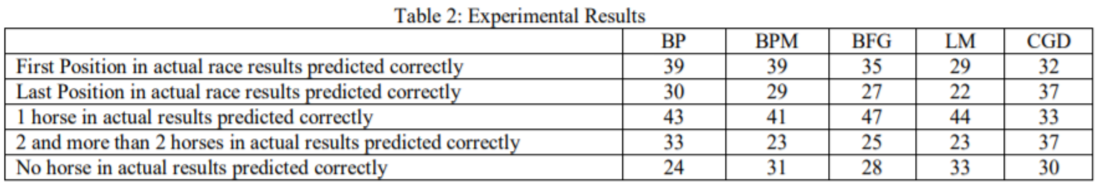


Table 2: Experimental Results

	BP	BPM	BFG	LM	CGD
First Position in actual race results predicted correctly	39	39	35	29	32
Last Position in actual race results predicted correctly	30	29	27	22	37
1 horse in actual results predicted correctly	43	41	47	44	33
2 and more than 2 horses in actual results predicted correctly	33	23	25	23	37
No horse in actual results predicted correctly	24	31	28	33	30

Figure 1: Experimental Results [16]

In 2013, Sameerchand Pudaruth developed a software that used a weighted probabilistic approach to predict the outcome of races at Champ De Mars, Mauritius [17]. They used the results of 240 races from the year 2010 as the data for the software and outperformed the best professional tipsters by achieving an accuracy of 58%. It enlightened some influential features such as jockey of a horse, worth of each horse, type of horse etc. that could be used for a model to learn. Although, as the evaluation was done on just 24 races, the evaluation results obtained was not substantial enough to be applicable everywhere. Later he extended his work and applied MLP (an Artificial Neural Network) composed of two hidden layers to predict the horse racing winner. Using the fundamental data of a horse race namely, "weight, draw, odds, jockey, previous performance and distance", with the output being the margin i.e., distance between the considered horse and the winner. Although the model presented an unconventional design and output, the accuracy was just limited to 25% [18]. Wai-Chung Chung in 2017, used the race results of 2691 race records to train a SVM based classifier for the prediction of horse race [19]. They used 19 features obtained from fundamental data to train several SVM models independently for the prediction of the winner. Subsequently, in the end result the individual SVM predictions were combined with the help of a committee machine. This method produced an accuracy of 35% which was better than the previous works.

2.2 Critical Analysis

Based on the machine learning and deep learning models used in the previous research works, we can infer that there is scope to use more sophisticated algorithms and neural networks. To harness their capability, it is important to realize their shortcomings and carry out research to make more suitable decisions regarding the type of network and the parameters that must be used to obtain better results [20]. In addition to this, we observed the datasets used were small consisting of some hundred races only [14,16,17]. Due to this reason, we cannot completely rely on the credibility of these results and require testing on larger datasets. As we established earlier that there is similarity between stock market and horse race betting fundamental theories. By researching about the state-of-the-art deep learning models for stock price prediction and multi-variate time series data classification, we found that majority of the methods are based on CNN and RNN's which denotes they are effective for multi-variate time series data [21,22]. Feature selection also known as feature subset selection, variable selection looks towards choosing input variables by removing immaterial features or of no foreseen information. The feature selection has few disadvantages. When the number of observations is inadequate or deficient, the overfitting possibility increases. With so many observations around thousands fundamental data set and lakhs of rows makes it difficult to interact and fit the chances [17].

2.3 Proposed Key Models

There are other models implemented and explained in the next chapter, below are the key ones:

MLP/CNN with Self-Attention

The success of any predictive model lies on the features over which it has been trained, though deep learning models have the capability to learn representation on its own, but it is a good practice to identify the important features and use them to train the model. Fundamental data is a tabular data having more than 400 features, so we propose using self-attention mechanism along with MLP or CNN to learn the important features. Recently, Gowthami et al. implemented a hybrid deep learning approach to deal with tabular data efficiently, which includes an enhanced embedding method and performs attention mechanism over both rows and columns [23]. This was introduced to solve the problem of deep learning models that obtained certain degree of hinderance in performance with traditional techniques. When tested over 16 tabular datasets, this method improved performance over conventional deep learning methods whilst also achieving better results over other gradient boosting algorithms, which would enhance the results of a domain of tabular data such as horse racing.

Ensembled Hybrid Model

We propose a novel model to predict horse race outcomes. It is an innovative thought to use information from both fundamental and technical data together in the form of concatenated embedding. Thus, not losing any valuable information. The architecture and implementation are explained in detail Chapter 3.

2.4 Research Objectives

The primary objective of this project is to evaluate the following hypothesis:

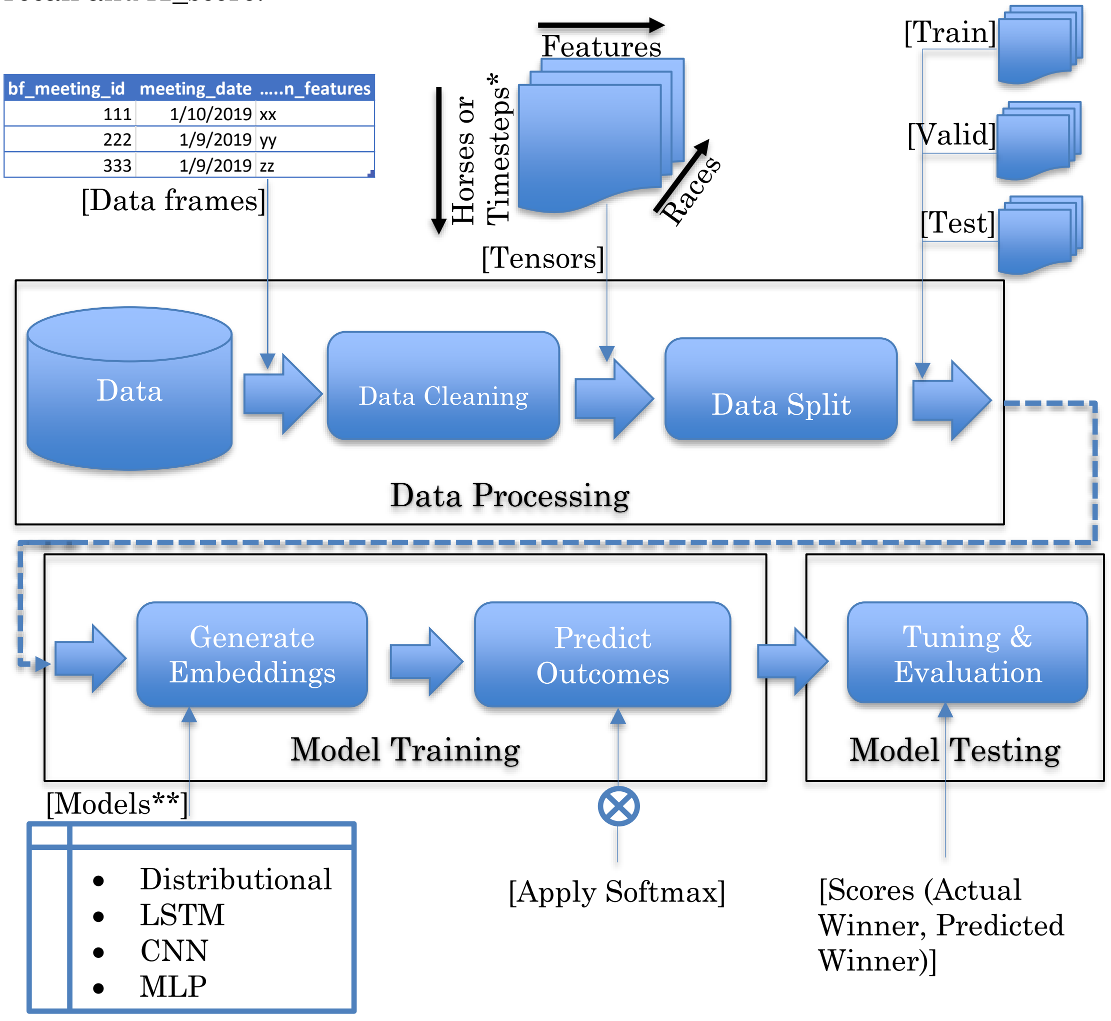
- Hypothesis 1 – Multi-layered Perceptron (MLP), though a simple and easy model to implement yet, it is quite efficient in learning the representation from both complex and simple datasets because of its adaptive learning capability. Therefore, MLP based neural network should be effective in predicting horse race outcome.
- Hypothesis 2 – Self-attention mechanism has been so far successful in natural language processing and computer vision task. The hypothesis is it can be equally effective with tabular datasets too. The idea is to evaluate self-attention for fundamental data in horse race outcome winner prediction.
- Hypothesis 3 – Long Short-Term Memory (LSTM) are capable of processing time-series data due to its capability to store previous time-steps in the form of hidden state. The objective is to evaluate LSTM for micro(technical) data having market odds of the horses participating in the race.
- Hypothesis 4 – Macro and mini data if used together can be more reliable and effective in forecasting horse race outcomes

Chapter 3 Design and Implementation Details

This chapter is in continuation to the fundamental theories laid down in the previous chapter. It illustrates the model architectures and their implementation details along with data processing. We begin with a high-level overview of the pipeline which is common to all the models implemented as part of this project. We then outline the libraries and infrastructure used. In line with the pipeline flow, we start with datasets description and their processing to form multi-dimensional tensors, followed by model's architecture – MLP, CNN, LSTM, and their variants. We end this chapter by listing down the evaluation metrics used.

3.1 Pipeline, Libraries, and Infrastructure

Figure 1 outlines the generic pipeline starting from data processing, model training, testing and evaluation. Data Processing comprises primarily of cleaning, tensor creation and splitting. The processed training data is then fed into the model for training which is hyper-tuned using the validation set. We use testing set to evaluate the model's performance based on scores – accuracy, precision, recall and f1_score.



bf_meeting_id	meeting_date	.....n_features
111	1/10/2019	xx
222	1/9/2019	yy
333	1/9/2019	zz

Figure 1: Model Pipeline

* - Rows represent horses for fundamental dataset while timesteps in case of technical data set.
** - The embeddings vary depending upon the model variant.

We employed below data science libraries to process the data and develop the models:

- pandas 1.1.5 – To load and process the data,
- numpy 1.19.5 – To read and form multi-dimensional arrays from the raw data,
- pyTorch 1.10.0+cu111 – To design and develop the models,
- scikit-learn 1.0.2 – To evaluate the model and cross validate
- matplotlib 3.2.2 – For visualization

Infrastructure

The datafiles (.csv format) used to train and test our models were quite large, ranging from 5GB to 10 GB approximately with hundreds of columns(features). Therefore, it was not feasible to perform pre-processing and training on a local workstation with 16GB RAM and 6 core-processor. We used Google's cloud platform Colab pro.The environment configuration was Python 3 with 25 GB GPU, which played a vital role in the model development life cycle starting from data processing to training. For ease of access, we used Google Drive as the data repository and mounted it at run time on Colab.

3.2 Dataset Description

Datasets used in this project was provided by roiLab, a Glasgow-based quantitative investment firm, also involved in several sports betting markets.

We used two different types of datasets namely Fundamental (macro) data and Technical (micro) data. Fundamental data includes individual attributes like horse's age, weight, sex etc., past performances like races won, best finish time, race gap, jockey's information, trainer's information, and owner's information. While Technical Data is a time-series data comprising of market odds, traded volume in given time interval. For example, traded volume of the horses participating in a race for the past thirty minutes with one minute time step.

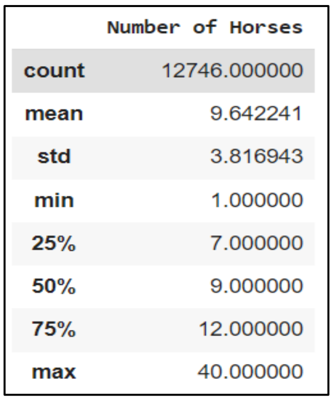
3.2.1 Dataset 1 – Fundamental Data

Exploratory Data Analysis

Fundamental Dataset provided by roiLAB covers races held in the year 2019 and 2020. We use 2019 dataset for training (65%), validation (15%) and testing (20%) the model based upon fundamental data. While 2021 dataset is entirely used in testing for fair evaluation of the model. We used only 2019 dataset for data exploration and analysis to avoid gaining knowledge of 2021 data distribution beforehand, thus setting up an environment equivalent to real-world scenario.

In this chapter if dataset year is not explicitly mentioned for fundamental, it should be deemed as 2019 data.

2019 Fundamental dataset consisted of a total of 1,22,903 rows and 421 columns where each row can be attributed as the characteristic of a runner for a given race. It comprises of 12,746 races held in the Great Britain and Ireland wherein 25,257 horses participated.



	Number of Horses
count	12746.000000
mean	9.642241
std	3.816943
min	1.000000
25%	7.000000
50%	9.000000
75%	12.000000
max	40.000000

Figure 2: Horse Race Distribution

Figure 2 describes the horse distribution across the races held in the year 2019. The average number of horses taking part in each race is around 10 with deviation close to 4 while 75% of the races have 12 races. We also observed outliers with only 1 race having one horse participating and a single race with 40 participating horses.

As mentioned earlier, there are 421 features such as horse_age, horse_code, overweight, horse_gender etc. characterizing each participant in a race. We delved further into them to establish the correlation between the winners and the respective attribute. Figure 3 demonstrates feature correlation examples.

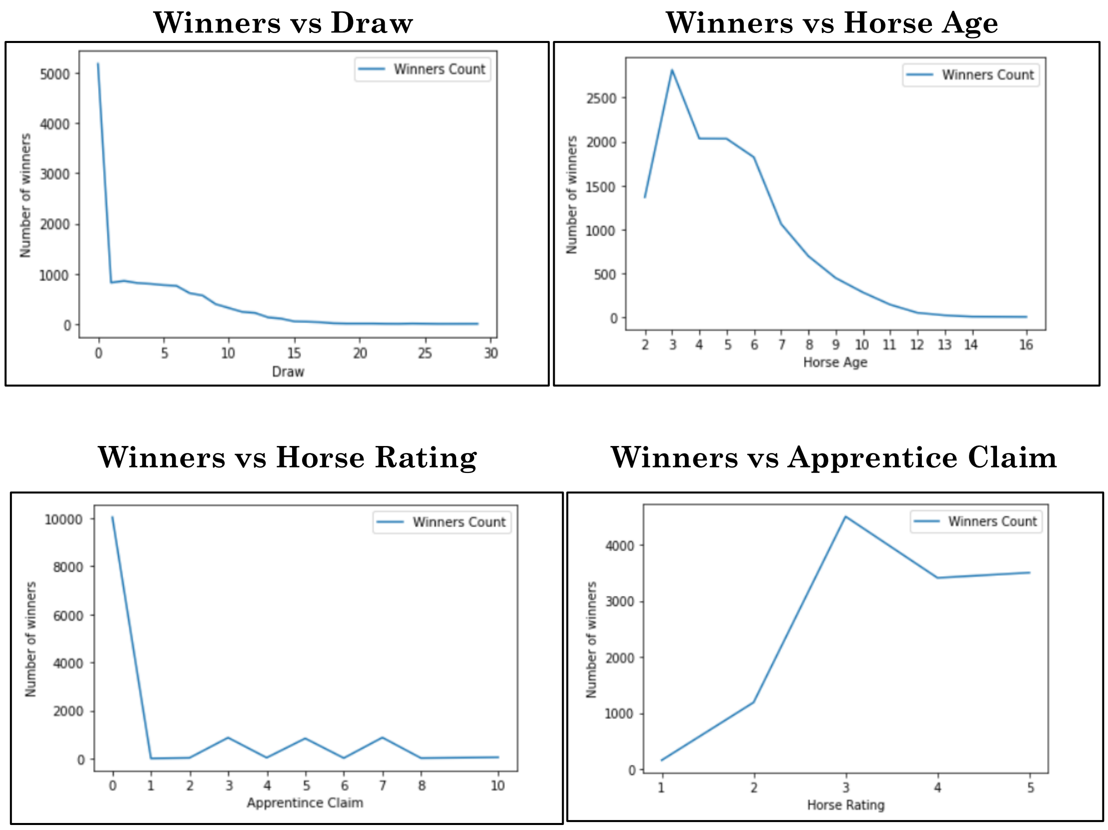


Figure 3: Features Correlation

Draw is the starting position of the horse in a race, which we found to be negatively correlated with the winner, lower the drawing higher is the winning probability and vice-versa. Horse's Age (experience of horse in racing) and Apprentice Claim (weight to be carried in proportion to the number of races jockey had won earlier) follow similar trend. Unlike, others horse's rating (bookies belief) is positively correlated to winning probability. However, it is not always the case odds play an important role in horse racing.

On further analysis, we found the data to be noisy with missing values, duplicate rows, features with substantial percentage of null values which were cleaned and processed. We used 344 features to train and test the model.

Dataset Cleaning and Processing

We load the raw data from both the csv files (2019 and 2021 Fundamental Dataset) into respective dataframes using Pandas library. In the initial processing, we observed 42 features missing in 2019 dataset while 60 in 2021 set. On further investigation, we identified the gap in naming conventions. Post fixing, we were left with 394 common features among both the datasets. The columns in the dataframes were reordered to have the same order in both. Dataset has both continuous and categorical features. We identify the feature category in respective datasets and ensure that both have same continuous and categorical set of features. We follow the best practice to clean the data. We first drop all the duplicate rows; duplicate here includes i) identical rows and ii) multiple rows with same race id and runner id i.e., each horse in a race can have only one record. We also drop rows with missing race identifier or runner identifier because they together represent a meaningful race entity. Further, we exclude textual features and only include feature (continuous & categorical) having less than 25% missing values because theoretically columns with more than 25% missing values noisy even after imputation with exceptions which requires extensive domain knowledge. Missing values for Continuous features were imputed with respective mean value while missing categorical feature were tagged as 'MissingValue' which were later encoded using sk-learn label encoder[2]. Finally, we are left with 307 continuous, 37 categorical features summing to 344 features in total.

Finally, we group all the rows by race identifier ('bf_market_id') to organize the data in the form of individual races and ensure each group (race) has only one winner. Races having no winners, or more than one winner are eliminated. Based on the statistics (Figure 2) and earlier studies, we filter the races having 5 to 20 participating horses. It is a primary step in data processing because it serves many purposes. Firstly, it helps to eliminate the outliers in the dataset which could have skewed the model's prediction. Generally, good betting strategies refrain from betting on races with very few numbers of participants due to inconsistency in return value while races with high number of participants suffer from high variance [14,15]. Secondly, all the models accept tensors of a fixed dimension, thus selecting races with a given range of participating horses helps in achieving a uniform tensor shape.

[2] sklearn.preprocessing.LabelEncoder — scikit-learn 1.0.2 documentation

Tensor Modeling

Data is fed into the model variants in the form of multidimensional tensor and as explained in the previous section, the perquisite is they must have uniform shape so that it can be processed by the model during training and testing. Each dataset post cleaning and feature selection is grouped by race identifier using Pandas dataframes which is subsequently modeled in the form of a multi-dimensional array of the shape – (r, h, f) using NumPy library. This NumPy array is converted into input tensor using PyTorch keeping the shape intact (Figure 4.1). Parameter 'r' refers to the number of races (unique 'bf_market_id' count) in the cleaned and processed dataset, 'h' denotes the number of horses participating in a race, and 'f' is the number of features finally selected. The tensor thus created is the input tensor which is used for training and testing model variants based upon fundamental data and hybrid model (explained later).

After data cleaning and processing we had 11897 and 11245 races in the 2019 and 2021 dataset respectively, while the number of features selected is 344 (continuous (307) + categorical (37)) and the maximum number of horses participating in a race was set to 20. Consequently, shape of the tensor modeled was 11897 * 20 * 344 for 2019 dataset while 11245 * 20 * 344. We also design an output tensor (Figure 4.2) for each dataset of the shape – (r, h). Again, 'r' represents the number of races while 'h' denotes the maximum number of horses participating in each race which is defined as 20. The output tensor stores binary values - '0' for losers and '1' representing the winner for each race in the dataset. Since, a race can have only one winner, a single two-dimensional array in the tensor is equivalent to a race and hence should have only single entry of '1', rest should be '0'.

As explained earlier, the number of horses in each race varies while we have a technical constraint to have uniform tensor shape i.e., same number of participating horses in each race. Thus, input and output tensors were zero-padded for races with less than 20 participants (maximum number of participants in a race) to preserve uniformity in shape of the tensor.

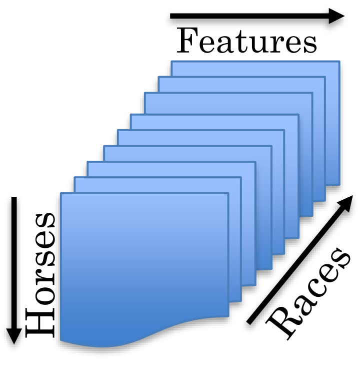


Figure 4.1: Input Tensor

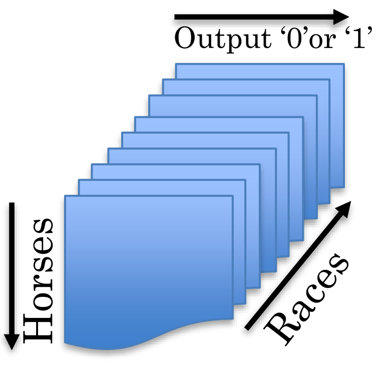


Figure 4.2: Output Tensor

3.2.2 Dataset 2 – Technical Data

Exploratory Data Analysis

roiLab also provided the Technical Dataset consisting of races held in the year 2019 only. We split the dataset like 2019 fundamental dataset i.e., 65%, 15% and 20% into training, validation and test set respectively. It consisted of market odds time series along with race identification attributes like ‘market_id’, ‘race_id’, ‘meeting_date’, ‘pt’, ‘market_type’, and horse identification features such as ‘runner_id’, ‘runner_name’ spread over 70 columns and 45,54,867 rows corresponding to 12,685 races participated by 25,590 horses from the Great Britain and Ireland. Each row is equivalent to a time step, i.e., it contains market odds (‘spf’, ‘ltp’. ‘wmb’, etc.) recorded for each participating horse in a race at a given instance. The market odds were recorded for each horse taking part in a race over a period at an interval of 1 minute. We considered odds for the last 30 minutes only, prior to the race start time to train and evaluate the model.

Further, we did a comparative study of trend followed by the key market odds attribute for horses winning a given race and horse losing the same race. Since, the number of races were quite high we sampled few races randomly and ploted the market odd trend for both the winner and losers corresponding to a race.

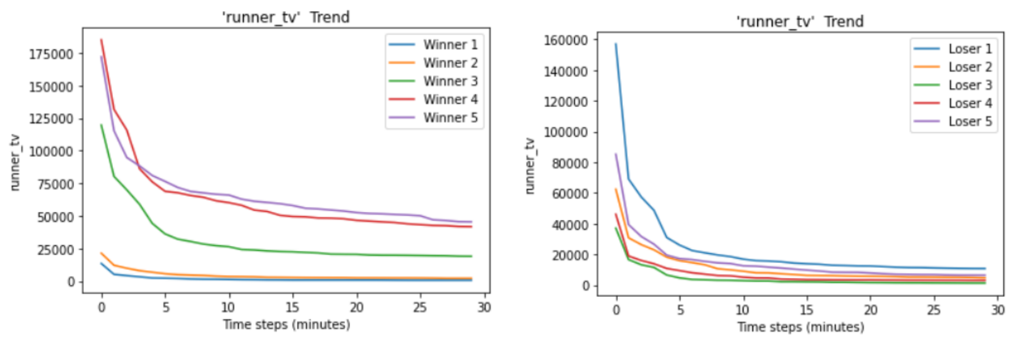


Figure 5.1: ‘runner_tv’ Trend

In Figure 5.1, on the left we have ‘runner_tv’ (total volume matched on a horse) over 30 minutes at an interval of 1 minute ('0' on the x- axis means 1 minute prior to race starting while 29 is 30 minutes prior to race.) corresponding to winners from five different races while on the right we have it for the losers. They seem to follow the same trend however, for the same race winner has higher ‘runner_tv’ value throughout 30 minutes while losers have a little lower ‘runner_tv’ value, though both increase gradually as the race approaches. Hence, it is a potential feature to discriminate between race winner and losers.

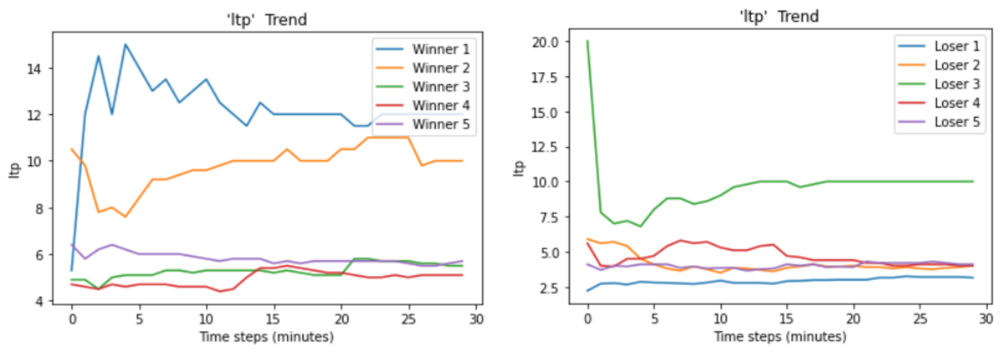


Figure 5.2: ‘ltp’ Trend

Similarly in figure 5.2, ‘ltp’ defined as the last price the runner was traded at in any given race, does not seem to follow any specific trend. However, there are few sharp changes close to the start of the race and in opposite direction for the winner and the losers. Unlike ‘runner_tv’ and ‘ltp’, ‘spf’ – starting price far (Figure 5.3) has high variance and does not have any uniform pattern at all.

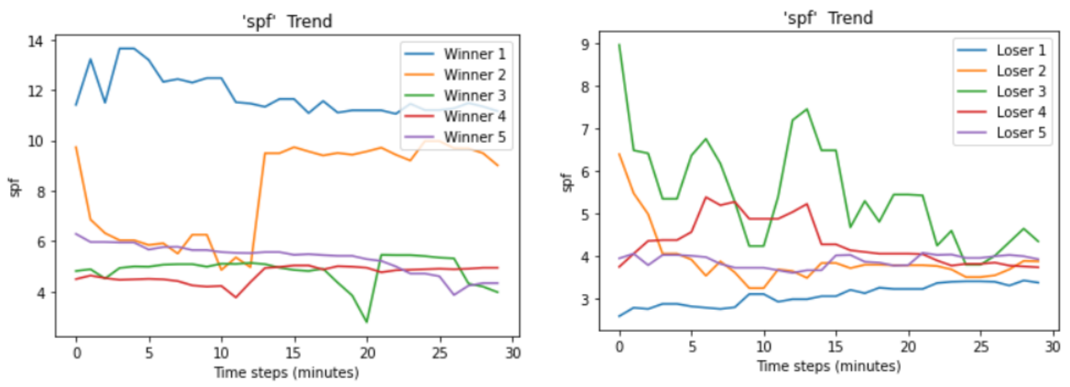


Figure 5.3: 'spf' Trend

'spn' – starting price near (Figure 5.4) is constant and equal to zero for all the horses in all the races. Therefore, it is not significant at all.

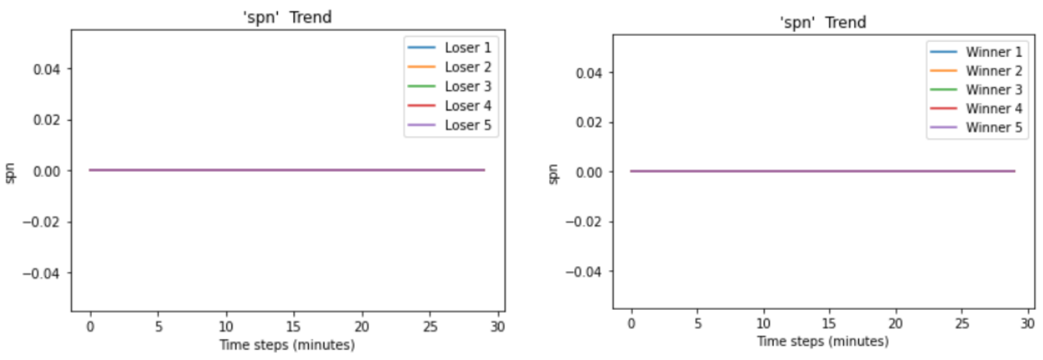


Figure 5.4: 'spn' Trend

We analysed few more market odds parameters like 'vwap', 'wmb', 'wml' , 'high traded price', 'low traded price' shown in Appendix A.

Dataset Cleaning and Processing

We use Pandas to load the technical data also and perform the cleaning operations. 'runner_id', 'market_id', 'event_id' are some of the features which uniquely identifies the participating horse and the race. We drop records with null or empty identifiers ('runner_id', 'market_id', 'event_id). Further, we followed similar data processing pipeline i.e., drop duplicate rows or duplicate 'runner_id' in the same race. Since the technical data did not have the race outcomes, we joined the technical data with fundamental data using the identifier attributes – 'bf_selection_id', 'bf_market_id' (fundamental data) and 'runner_id', 'market_id' (technical data). For the hybrid model we also filtered the fundamental data for the common races in technical and fundamental data. We further processed data for few more checks like number of winners in a race, races with more than one winner were dropped (144 races were dropped). We also performed few basic but important checks like the number of horses participating in each race are same in fundamental and technical data, for a given race winner is same in both fundamental and technical data; mismatches found were dropped. Additionally, we imputed the missing market odd values with its previous timestep value. For, races with substantial amount of missing information were dropped. After, completely cleaning and processing the data we end up with 3613 races.

Tensor Modeling

Likewise, fundamental data we also model the technical data into multi-dimensional tensor. There is slight variation in the dimensions of the tensor, it's shape is (r,t,f), where 'r' is the number of races, 't' is the timestep and 'f' denotes number of features. Unlike the fundamental data tensor, the rows here are the timesteps in the chronological order – first row having the market odd values 30 minutes prior to start of the race while last row has the values 1 minute before the race starts. Consequently, the input tensor has 30 rows for each race. The columns here are the market odd values, and they are ordered in a fashion such that different market odd values M1H1, M2H1 of Horse1 are placed right next to each other followed by odd values corresponding to Horse2 M1H2, M2H2 and so on. The number of columns is dependent upon the number of market odd features considered to train the model. The output tensor (Figure 4.2) is identical to the one created for fundamental data.

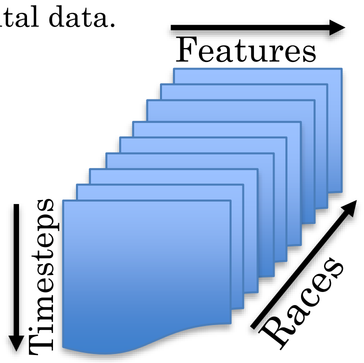


Figure 6: Input Tensor (Technical Data)

3.3 Model Description

We designed and developed deep learning neural network-based classifier to predict horse race outcomes. We investigated and experimented different model variants – MLP, CNN (1D convolution), LSTM using fundamental and technical data. We also came up with a novel approach to train the classifier using embeddings from both fundamental and technical data simultaneously. All the model variants followed a similar pipeline with different underlying neural network architecture. In the process, we also explored the potential of multi-headed self-attention mechanism for tabular data (non-sequential data) i.e., fundamental data in our case.

As mentioned earlier, we used PyTorch library to implement all the model variants because of its unique yet simple dataloader feature to feed data into the network for training and easy to use enriched pre-implemented abstract classes for different purposes like Linear Modules, CNN etc. which meets our project needs.

3.3.1 MLP – Fundamental Data

Model Architecture

Figure 7.1 outlines the underlying architecture of the model which can be decomposed into two major sections excluding raw data processing function to create the input tensors and output evaluation. The first section is composed of 37 embedding layers one for each categorical feature followed by dropout layer in parallel with a batch normalization layer for continuous features and finally concatenation of continuous and categorical feature embeddings as one single embedding. It projects the feature in a latent space and generates distributional embeddings for each feature in a n-dimensional vector space. These embeddings are learned during model training. The second section is primarily a simple Multi-layered perceptron with 2 hidden layers, each followed by a batch normalization and dropout layer to avoid overfitting and a linear output layer with softmax function to generate the winning probability of each participating horse in a race. Softmax is used in the output layer because the nature of the problem is multi-class classification. As shown in Figure 7.1, hidden layers are linear layers having Relu as the non-linear activation. Relu is chosen as the activation function because it does not suffer with gradient descent vanishing problem during model training, while other potential activation functions such as sigmoid and tanh do suffer from it.

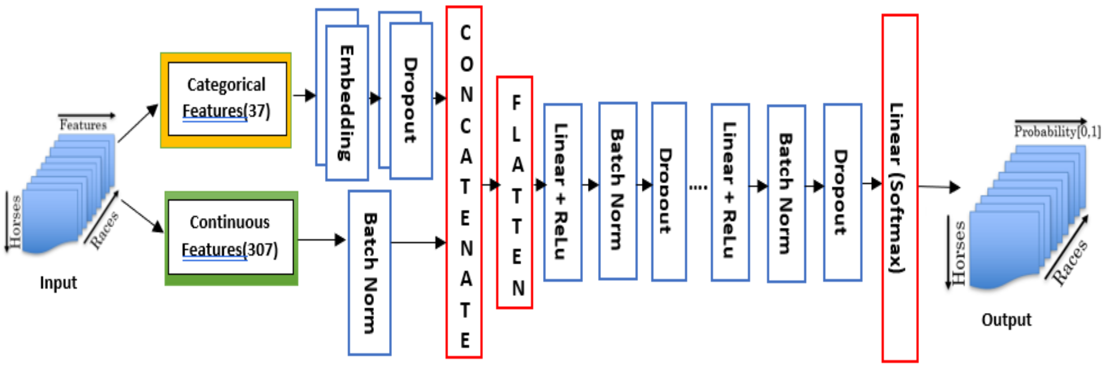


Figure 7.1: MLP (Fundamental Data) – Model Architecture

Model Implementation

Input tensor modeling process has been explained in the Dataset description section and is common to all the model variants, it differs slightly depending upon dataset type i.e., fundamental, and technical. We use nn.Module class to implement the model. The layers in the model are initialized within the __init__ method of the class and default forward method is overridden to define the structure of the model as explained in the model architecture (Figure 7.1). Model initialization requires nine parameters in total. 'emb_dims' is a tuple comprising of each categorical feature class size and its respective embedding dimension. nn.Embedding generates the distributional embeddings for the categorical features which is followed by the embedding dropout rate set to 0.5. 'emb_dropout' is the parameter to set its value. 'no_of_cont' is the number of continuous features passed in the input. nn.BatchNorm1d, the inbuilt batch normalization class is instantiated for normalization layer. The inbuilt function torch.cat() is used to concatenate the categorical and continuous embeddings, torch.view flattens the concatenated vector. For simplicity, we use two lists - 'lin_layer_sizes' and 'lin_layer_dropouts' to define the input dimension of the hidden layers and their corresponding dropout in the network. We used two layers with dimensions 60 and 0.1,0.01 dropout rate for each layer respectively. The length of the list is equal to the number of hidden layers in the network. Since we have 20 horses participating in each race, 'output_size' is always set to 20 thus defining the dimension of the output layer. PyTorch default Relu and softmax functions are used in the hidden layer and output layer respectively. There are additional parameters 'attention' and 'num_heads' (explained in the next model). 'attention' is a boolean parameter which should be always set to False for this model.

3.3.2 MLP with Self-Attention – Fundamental Data

Model Architecture

This model is an enhanced version of the MLP model explained in the previous section supported with self-attention mechanism. As shown in Figure 7.2, the underlying neural network architecture and principle is identical to the last model except for an additional multi-headed self-attention layer just before the MLP layer. The input and output tensor also remains unchanged.

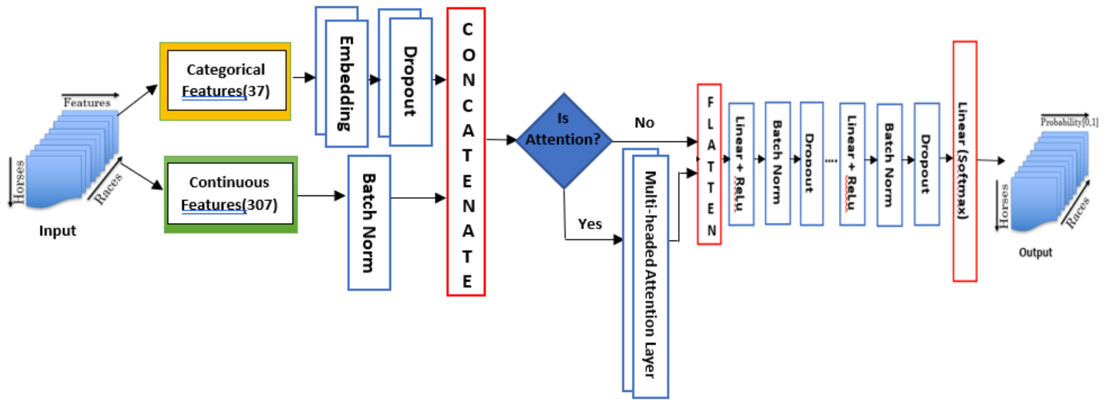


Figure 7.2: MLP with Self-Attention (Fundamental Data) – Model Architecture

Multi-headed self-attention layer is added just in between the two sections described earlier – i) embedding generating layer and ii) MLP. The concatenated embedding of the categorical and continuous features is fed into the attention layer. The attention layer can be broken down into two parts – i) Attention and ii) Layer normalization. It returns two outputs namely attn_output (it is the dot product of the input vector and attn_weights determined in the training process) and attn_output_weights. The idea behind using self-attention mechanism to determine and give high weightage to the more relevant features which is learned during training [23]. Attn_output is of the same dimension of the concatenated vector fed into it which is then passed into layer normalization followed by the MLP section and finally the output layer with softmax function like the last model to generate the win probability distribution for the input batch of races.

Model Implementation

We implement this model using the same class with two additional parameters – 'attention' (boolean) and num_heads (integer). The rest of the implementation remains the same as explained in the model implementation section of the last model. Whether the neural network model should exploit attention mechanism or not is controlled by the 'attention' parameter which when set to True, activates the multi-head attention in the network else it remains dormant. Self-attention mechanism is employed by instantiating torch.nn.MultiheadAttention class, which has the potential to process in parallel, the number of parallel processes is determined by the parameter num_heads. It is recommended to set num_head such that embedding dimension of the input to this layer is completely divisible. We set num_heads to 2 because embedding dimension i.e., dimension of the concatenated vector (categorical + continuous embedding) was 1046. Factors of 1046 are 2 and 523. It is infeasible and computationally expensive to have 523 parallel attention layers. Ideally, multi-attention layer expects three different input vectors in the form of query, key, and value but since the data here is neither sequential nor does it have time step based hidden states. Therefore, we pass the concatenated vector as the query, key and value thus making it a typical self-attention layer. The network after attention layer to the output layer is same as the last model and hence implementation was also same.

3.3.3 CNN with or without Self-Attention - Fundamental Data

Model Architecture

This is the last variant of the models we implemented purely using fundamental data. It has two flavors – i) CNN without self-attention ii) CNN with self-attention. Figure 7.3 gives an overview of the model architecture. Again, the input tensor is the same as the ones used in the models explained earlier having same shape and dimension. Convolutional Neural Networks are known to be widely used in image recognition, video processing and other computer vision tasks, however its application to tabular dataset like the one we have is limited because the underlying principle of CNN is based upon the assumption that there exists spatial correlation between the neighboring features which rarely holds true in case of tabular datasets. However, the tabular dataset we have can be transformed into a structure such that all the features of a horse participating in a race are in a sequence followed by the features of another horse and so on.

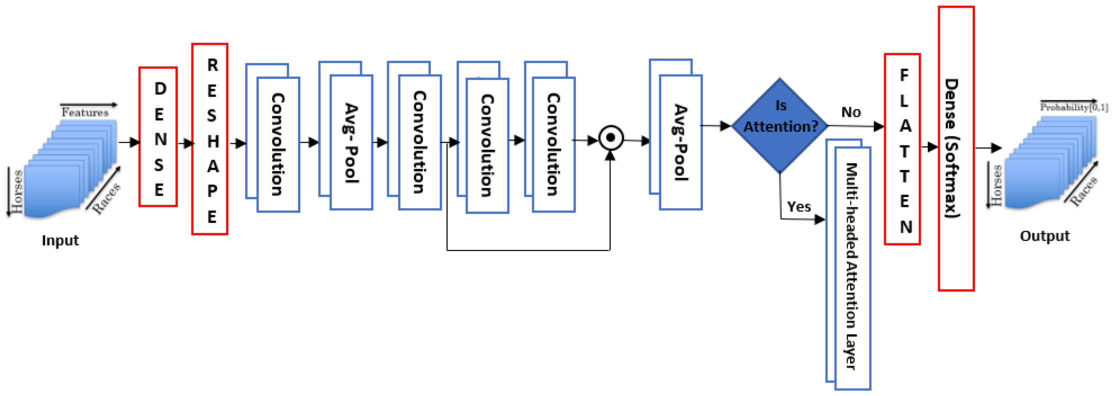


Figure 7.3: CNN with or without Self-Attention (Fundamental Data) – Model Architecture

In this capacity, we designed a neural network with four convolution layers with a dense layer as the input layer. For the reason, explained above we reshape output of the dense layer and also increase its dimension so that the following CNN layers will have substantial number of features to learn the representation. Each CNN layer consists of a 1-dimesnional convolution i.e., kernel moves in only one direction, which is preceded by a 1d batch normalization layer and a dropout layer, followed by a non-linear activation function – Relu. The output of the first convolution layer is average(adaptive) pooled and then fed into the second convolution layer. While the output of the second convolution layer is not only passed on to the next layer (third convolution layer) but also added to the output of the fourth convolution layer to mitigate information loss over several layers. The final output of the convolution layer (fourth layer) after simply adding with the output of the second convolution layer is average pooled. The embeddings thus generated is passed to the multi-attention attention layer, if 'attention' parameter is set to True else it is flattened into a one-dimensional vector before passing it to the output layer. Output layer is same as the earlier model variant, a linear layer followed by softmax function to compute the winning probability distribution of the horses participating in a race.

Model Implementation

The model is implemented using nn.Module class. The layers in the model are initialized in the __init__ method and forward method defines the layer structure of the model as explained in the model architecture (Figure 7.3). The model requires nine parameters to initialize. 'input_dim' is the dimension of the input layer which is equal to the number of features in the input tensor, 'output_dim' is the dimension of the output layer which is equal to 20, number of horses participating in race. Other parameters include channel size of the first convolution layer and hidden convolution layers (third and fourth) which is set to 64. The output channel of the first convolution layer is 128 which is the input of the second convolution layer while output of the second convolution layer is 64 only. The stride of all the CNN layers is 1 while kernel size is 5 for the first and last CNN layer, 3 for second and third. Similarly, padding is 2 for the first and last CNN layer while 1 for second and third. The remaining part of the network was implemented similar to the previous models having batchnorm layer, dropout layer, linear layers and Relu activation function, with softmax function applied in the output layer.

3.3.4 LSTM multi-variate – Technical Data

Model Architecture

As explained earlier, technical data contains information about the market odds of horses participating in a race for a period of 30 minutes at a time interval of 1 minutes till the race starts. Therefore, this data distribution is a potential candidate for time-series based forecasting using LSTM (Long Short-Term Memory), an enhanced version of Recurrent Neural Network, described in Chapter 2. LSTM layer empowers the model to store information over a time-period in the form of hidden states. Moreover, unlike the basic feed forward neural networks it can process the entire sequence of data as well as a single data point. It is noteworthy that we modeled technical data input tensor slightly different from the fundamental data input tensor. The rows here are the timesteps in the chronological order – first row having the market odd values 30 minutes prior to start of the race while last row has the values 1 minute before the race starts. Consequently, the input tensor has 30 rows for each race. The columns here are the market odd values, and they are ordered in a fashion such that different market odd values M1H1, M2H1 of Horse1 are placed right next to each other followed by odd values corresponding to Horse2 M1H2, M2H2 and so on. The number of columns is dependent upon the number of market odd features considered to train the model.

The model shown in Figure 7.4, is a multi-variate LSTM model wherein the input tensor fed into the network contains information about two market odd variables namely 'runner_tv' and 'ltp' for all the 20 horses participating in a race organized in an ordered sequence explained earlier. LSTM layer is the input layer of the network which generates the embeddings, perhaps it is the last hidden state of this layer. The hidden state embedding is then attended by the output layer which is a fully connected dense layer with softmax function that produces finally the winning probability for all the horses participating in each race in the input batch.

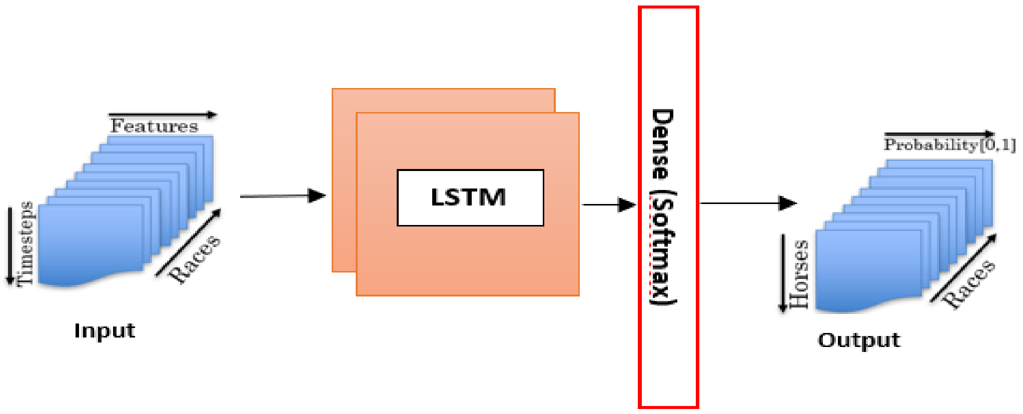


Figure 7.4: LSTM multi-variate (Technical Data) – Model Architecture

Model Implementation

We used the inbuilt nn.LSTM class that inherits the base class - nn.Module to implement the LSTM layer. The three important parameters to instantiate the LSTM layer object are i.) 'input_dim' – it is the input size which is equal to 40 (20 (number of horse) * 2 (features – 'runner_tv and 'ltp") = 40), 'hidden_dim' – it defines the dimension of the hidden state which is set as 512 after doing an empirical analysis with different values and iii) 'layer_dim' – it represents the number of LSTM layers to stack. We arrived at three LSTM layers in this model because a higher number is computationally expensive and requires more training time. 'batch_first' is another parameter which is always set to True so that model is aware that shape of the input and output (LSTM layer) tensor is in the order - batch, sequence, feature. Finally, the output layer is a dense layer having 20 neurons one each for the 20 participating horse, instantiated using nn.Linear(). The default softmax function is applied to output of this layer along dimension 1 to predict the outcome of the races in the batch.

3.3.5 LSTM multi-variate + MLP – Technical Data

Model Architecture

The model depicted in Figure 7.5 is also a LSTM multi-variate model but with additional fully connected layers (Multi-Layered Perceptron). The LSTM layer used in this model is identical to the one used in the last model i.e., same input tensor, same input size, same number of layers and same dimension of the hidden states. The following section in the neural network is the Multi-Layered Perceptron with two hidden layers and an output layer. As explained earlier in the first MLP model, Relu function is applied as the activation function to the output of each hidden layer for non-linearity. MLP layer is stacker over the LSTM layer because dense layers are more expressive and can help learn the underlying latent feature representation more efficiently. Since, it is a multi-class classification problem, softmax function is applied to the output layer like other models to predict the outcomes of the horse races in the input batch.

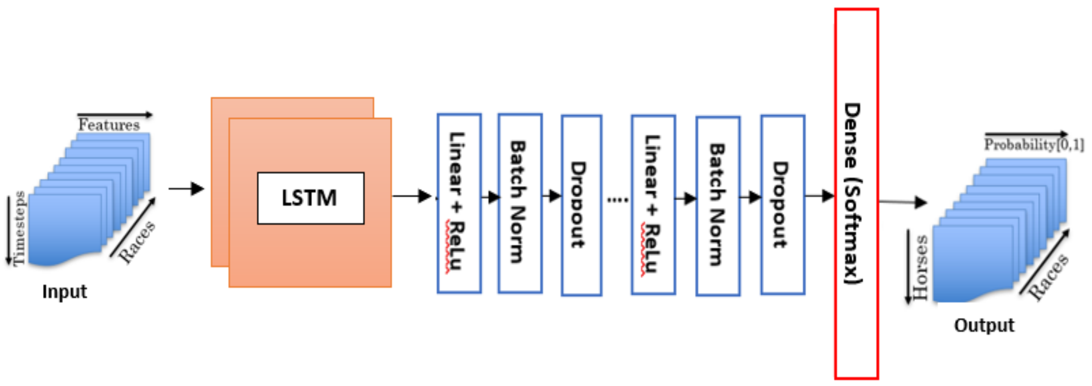


Figure 7.5: LSTM multi-variate + MLP (Technical Data) – Model Architecture

Model Implementation

We instantiated the objects for the LSTM and MLP classes implemented for the models explained earlier. We were able to use the same implementation with slight hyper-parameter tuning. These parameters were passed as arguments to the class constructor for the respective object instantiation. One major change in the implementation was the input dimension of the first layer in the MLP layer because it should be equal to the dimension of the last hidden state in the LSTM layer.

3.3.6 LSTM Univariate – Technical Data

Model Architecture

Previous LSTM models implemented were multi-variate wherein more than one feature was fed into the LSTM layer. We used two features 'runner_tv' and 'ltp' by modeling it into an input tensor explained earlier. The 20 classes in the output tensor representing 20 horses are corelated with the input tensor by the index position. That is the feature at column 1 corresponds to the horse at row index 1 in the output tensor. Ideally, each column in the input tensor should correspond to the horse index in the output tensor.

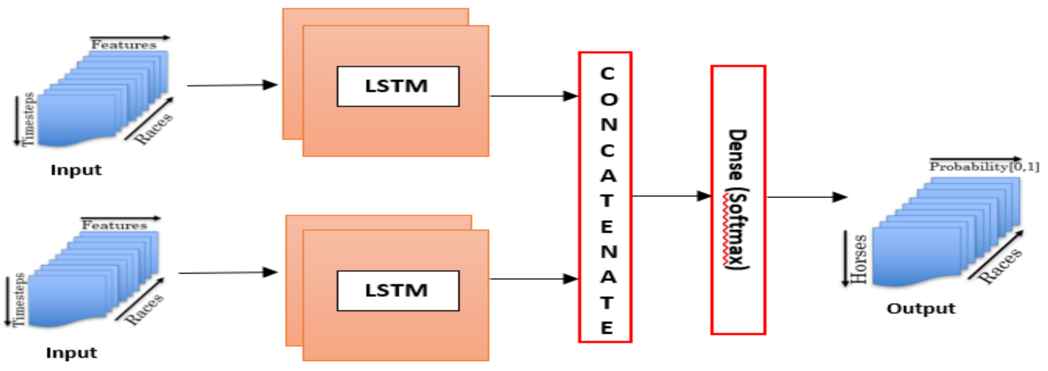


Figure 7.6: LSTM Univariate (Technical Data) – Model Architecture

We approached with another neural network model shown in Figure 7.6, wherein we create two different input tensors (one each for ‘runner_tv’ and ‘ltp’) of the shape (r,r). r – is the number of horses participating in the race, and is equal to 20, thus number of columns is also 20 because each column corresponds to a unique horse. The two input tensors are fed into the two different LSTM layers having the same input dimension, hidden state dimension and number of stacked LSTM layers. The output of the LSTM layers are the two hidden state embeddings of the same dimension which are concatenated and subsequently passed to the output layer. The input size of the output layer is twice the dimension of the hidden state. Like, other models softmax function is used in the output layer to determine the winner probability distribution of each race in the input batch.

Model Implementation

We implemented the model by first creating two separate input tensors for each feature namely ‘runner_tv’ and ‘ltp’. The two separate LSTM layers were instantiated using nn.LSTM class with input_size equal to 20, while rest of the parameters were same i.e., 3 stacked hidden layers while hidden state dimension as 512. Likewise, ’batch_first’ was set to True. We used an additional inbuilt torch.cat() function to concatenate the output of the two LSTM layers. The output layer’s input size was set to 1024 (twice the hidden state dimension).

3.3.7 LSTM Univariate + MLP – Technical Data

Model Architecture

The model demonstrated in the Figure 7.7 is an extension to the LSTM univariate model explained in the previous section. It can be decomposed into two major parts – i) LSTM layer and ii) MLP layer. The first part is an exact replica of the LSTM univariate model excluding the output layer. We designed this model with the hypothesis to improve the performance of the last model by adding the MLP layer to the neural network architecture.

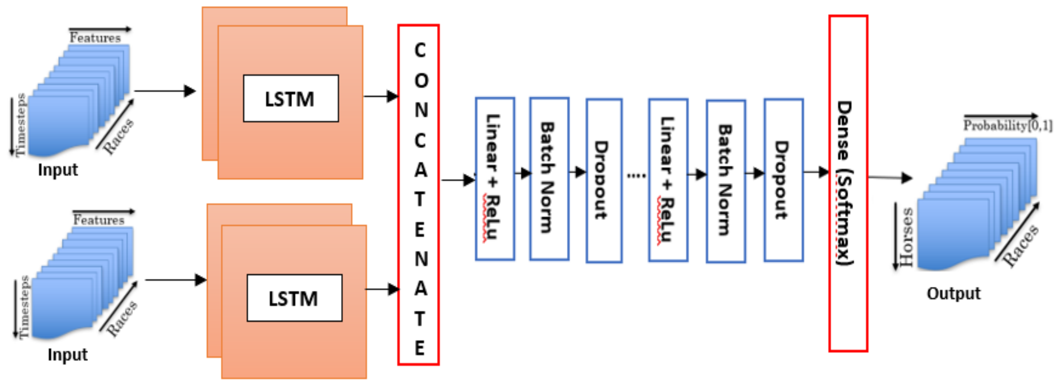


Figure 7.7: LSTM Univariate + MLP (Technical Data) – Model Architecture

This model works on the same underlying principles, generated separate embeddings for each feature which are concatenated to form one single embedding. Unlike the last model instead of passing these embeddings straight to the linear output layer, they are fed into the MLP layer comprising of a sequence of hidden dense layers followed by the linear output layer having 20 as the input size. Like, other models softmax function is used to predict the outcome in terms of probability of each horse winning the race. The composition of the MLP layer is identical to the ones used in the previous models.

Model Implementation

We used the same function used in the last model to create the input tensors for each feature 'runner_tv' and 'ltp'. Even for the LSTM layer and MLP layer we used the classes already implemented. We used the respective class constructors with the same parameters to create class instances. The input size of the first hidden layer in the MLP layer was changed to twice the dimension of the hidden state (1024). We did not change other configurations such as number of LSTM stacks, hidden state dimension. We used the same composition of MLP layer also except for the input size of the first hidden layer mentioned earlier.

3.3.8 Ensembled Hybrid Model – Fundamental + Technical Data

Model Architecture

Figure 7.8 illustrates a novel neural network architecture, a hybrid model focused on utilization of information hidden in both fundamental and technical data. As explained earlier in this chapter in the Dataset Description section, fundamental data represents horse's individual information, its past performance, information of race like surface type, jockey's characteristics, trainer's attributes, and owner's information. We did critically analyze that fundamental data misses on few aspects which could be provided by the technical data. Technical Data contains the market odd information about the race. Therefore, we designed an original neural network.

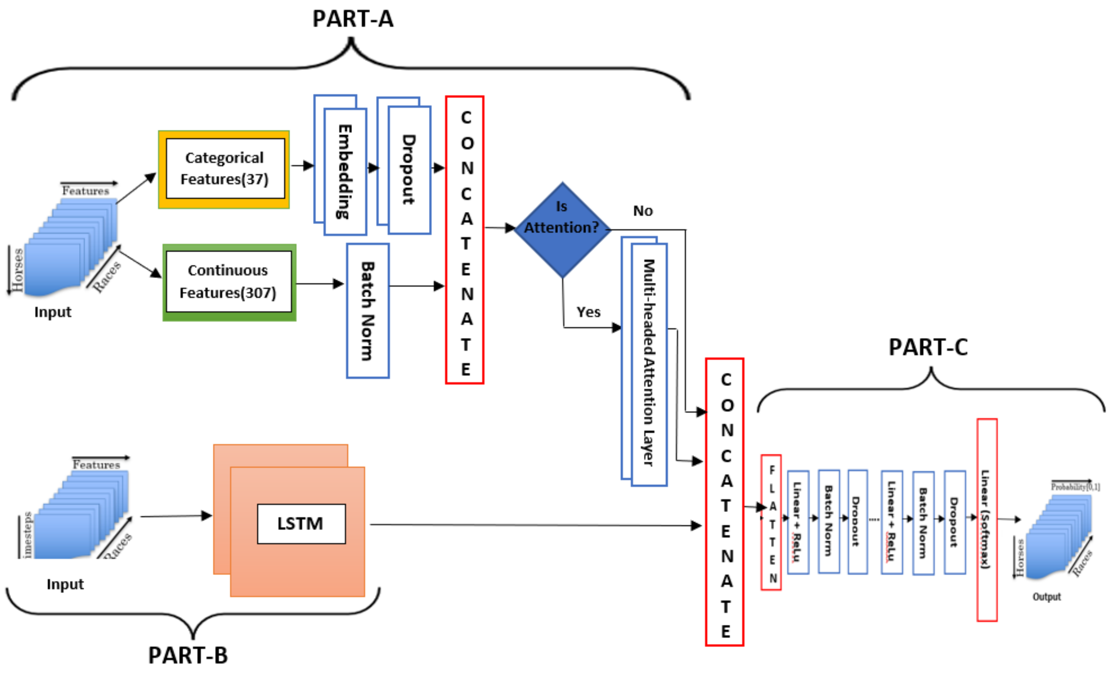
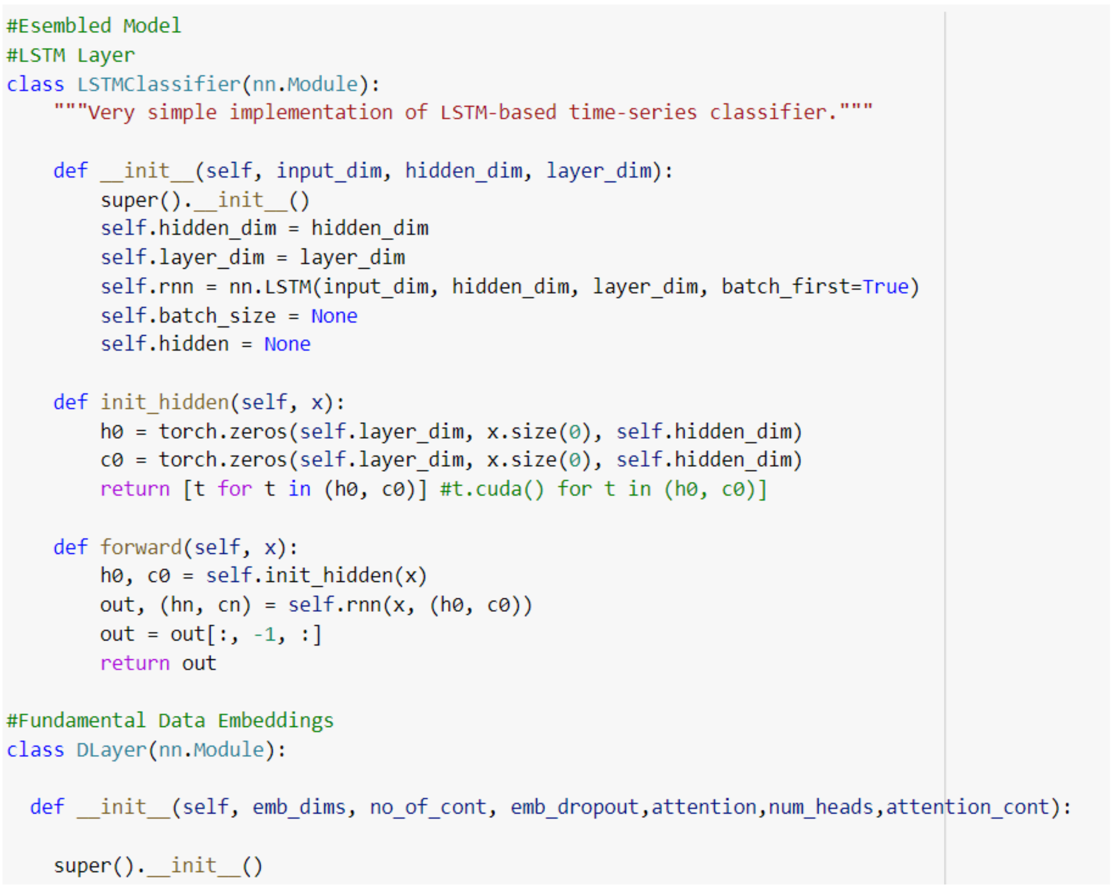


Figure 7.8: Hybrid Model – Fundamental + Technical Data – Model Architecture

The overall model architecture can be divided into three different parts: i) Part A, Fundamental Data Embeddings ii) Part B – Technical Data Embeddings ii) Part C – Ensembled embedding and MLP layer. Part A is composed of layers to generate distributional embeddings for the categorical features in the fundamental data which are concatenated with the continuous features to generate the fundamental data embedding. We also used multi-head self-attention mechanism to further learn salient features in the fundamental data embeddings. Part B consists of LSTM layer which received technical data input tensor. The LSTM embeddings thus generated are concatenated with the fundamental data embeddings. Consequently, we extracted information from both fundamental and technical data into a single embedding. Part C is the MLP layer having hidden layers and Relu as the activation function, followed by the linear output layer and finally the softmax function which generates the winning probability distribution of the races in the input batch.

Model Implementation

We implemented the model using the same class defined for the models explained earlier. However, we implemented a slightly different data processing pipeline to create input tensors for this model. The data was cleaned and processed likewise post-which we implemented functions to ensure that the input tensor created for the fundamental data and technical data have the same races, in the same order, and horses within a race were also in the same order. This was a crucial step in implementing this model. The remaining implementation was like the other models.



```
#Esembled Model
#LSTM Layer
class LSTMClassifier(nn.Module):
    """Very simple implementation of LSTM-based time-series classifier."""

    def __init__(self, input_dim, hidden_dim, layer_dim):
        super().__init__()
        self.hidden_dim = hidden_dim
        self.layer_dim = layer_dim
        self.rnn = nn.LSTM(input_dim, hidden_dim, layer_dim, batch_first=True)
        self.batch_size = None
        self.hidden = None

    def init_hidden(self, x):
        h0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_dim)
        c0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_dim)
        return [t for t in (h0, c0)] #t.cuda() for t in (h0, c0)]

    def forward(self, x):
        h0, c0 = self.init_hidden(x)
        out, (hn, cn) = self.rnn(x, (h0, c0))
        out = out[:, -1, :]
        return out

#Fundamental Data Embeddings
class DLayer(nn.Module):

  def __init__(self, emb_dims, no_of_cont, emb_dropout,attention,num_heads,attention_cont):

    super().__init__()
```

Chapter 4 Experiments and Evaluation

In this chapter, we set forth the approach followed to develop the models, experiments performed to evaluate different model variants and their respective results are critically analyzed. We also test the hypothesis detailed in Chapter 2. The metrics used to evaluate the models are also discussed. As mentioned in Chapter 3, the models were trained and tested using two different datasets namely fundamental and technical dataset. Models purely trained on fundamental dataset were tested against two test sets – 20% of 2019 fundamental dataset and complete 2021 fundamental dataset. While models based upon technical data were evaluated using 20% of the 2019 technical dataset only. We conclude the chapter with discussion of the novel model and its outcomes.

4.1.1 Model Development Approach

We followed the best practice to develop the models. In the initial phase, we implemented a simple shallow neural network for each model variant and trained it to ensure if the data loading and network initialization was as expected. With this shallow network, overall average accuracy achieved on the validation set was around quite good but not equally good on the test set. After ensuring that the model implemented was logically correct, we increased the complexity of the model by increasing the number of hidden layers, number of neurons in a hidden layer to the extent that the model overfitted the training data. We employed regularization techniques like batch normalization, dropout layers, average pooling and so on depending upon the underlying principle of the model variant in consideration to mitigate overfitting so that the model can generalize well when exposed to unseen data. Further, we conducted several experiments to empirically analyze and tune the hyper-parameters such as number of epochs, batch size, learning rate for each variant. These hyperparameters are not common to all the models because of the difference in fundamental theory the model is based upon. We used the same data split ratio for all the variants i.e., 65, 15 and 20 for training, validation and testing respectively.

4.1.2 Evaluation Metrics

We assessed the performance of all the variants based upon the following metrics:

Accuracy – It is defined as the percentage of correct predictions or the fraction of races out of the total number of races whose winner model predicts correctly. Prediction of horse winning the race is a multi-class classification problem since we compute the probability distribution of all the horses participating in a race and the one with the highest probability is considered as the winner of the respective horse race.

Precision – It is an important metric to evaluate the model because, frequent incorrect prediction of the winner in a race could prove quite costly. The betting strategies contingent upon a model with poor precision would be highly risky. Precision optimization pushes the model to effectively forecast the winner of the race.

Recall – There is always a trade-off between Recall and Precision. While improving, the precision, recall should not be completely ignored. A low recall drops the probability of number of times horse race outcome prediction could be correct. Thus, to devise a betting strategy recall should also be considered.

F1 score – It is a balanced evaluation metric, which helps to keeps a check on the trade-off between Precision and Recall. Therefore, a good F1 score is desired for a good betting strategy.

Later, in the chapter we have done a comparative study of all the 9 model variants using these metrics.

4.1.3 Experiments and Result Discussion

We can broadly categorize the models implemented in terms of dataset used into two primary categories – i) Models using Fundamental Dataset ii) Models using Technical Data. We will begin with the first category of models followed by the second and finally the novel implementation, a hybrid model exploiting both fundamental and technical data.

1.) Baseline - MLP (Fundamental Data) is the baseline we implemented because of its simplicity and to gain an insight about the data distribution. We trained the model for 20 epochs with a batch size of 256. We used AdamW as the optimizer along with cyclic LR scheduler and cross entropy as the loss function since it is a multi-class-classification problem. The adaptive nature of AdamW helps to regulate the learning rate leading to a faster convergence and refrain from using additional hyperparameter -weight day. We kept the learning rate to 0.0001 because a higher rate could lead to divergence. We can infer from Figure 8.1 loss curve(left) and accuracy curve (right) that around 15 epochs were enough for the model to train post-which, validation loss is close to constant and parallel to the training loss. Likewise, validation accuracy is also constant and parallel to the training accuracy curve which is an indication of good fit. If the training is continuing further, model is likely to overfit. We use two different approaches early stopping and 10-fold cross validation apart from model regularization to mitigate overfitting.

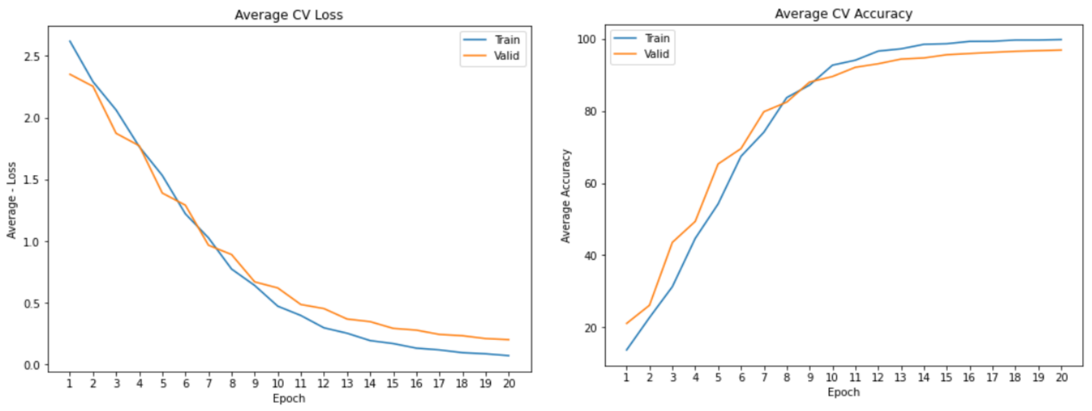


Figure 8.1: MLP – Fundamental Data

As stated in the hypothesis (Chapter 2), MLP though a simple deep neural network proved to be quite effective gaining 69% accuracy and close to 0.70 weighted f1_score on the 2021 fundamental test set. Precision and Recall were also impressive – 0.742 and 0.694 respectively.

2.) Explore Self-Attention – Contrary to our hypothesis that self-attention did not help improve the effectiveness of the baseline MLP model. We infused multi-headed self-attention (2 heads) into the baseline MLP to explore its potential. We tried tuning the hyperparameters, but model could not really learn the representation. It is quite evident from the loss and accuracy curve shown in Figure 8.2, loss decreases sharply in the first 2-3 epochs while accuracy surges exponentially.

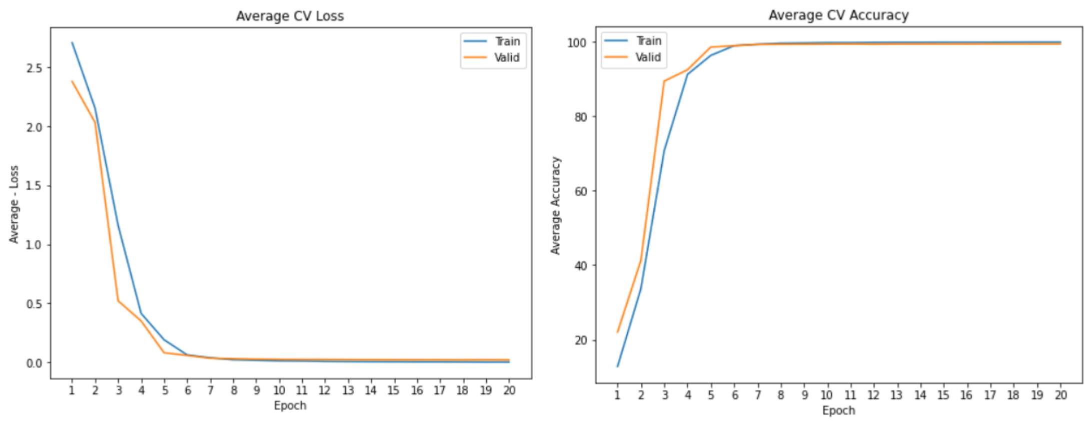


Figure 8.2: MLP with Self-Attention – Fundamental Data

This can also be concluded from the fact, that model attained high accuracy (97%), precision (0.975), recall (0.973) and f1 score (0.974) for the 2019 test set but accuracy was mere 37% for the 2021 test set. The model failed to learn the data distribution which could probably be due to small dataset, high learning rate.

3.) CNN for fundamental dataset with and without self-attention Convolutional network have proven to be quite effective in computer vision, image processing etc. However, for tabular datasets like the one we have its application is limited. From the previous research work we identified instances where it was equally efficient precisely Conv1D. The result of our experiment supports this hypothesis.

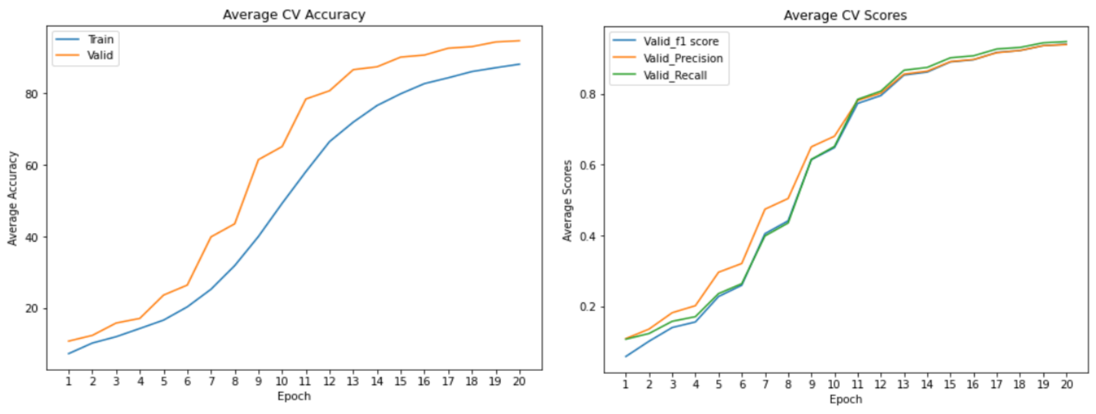


Figure 8.3: CNN – Fundamental Data

We explored two variants of CNN i.e., with and without multi-headed self-attention mechanism. Surprisingly, attention mechanism did not really help to improve the scores, but it hampered its accuracy apparently. The reason after this abrupt behavior could be the underlying principle self-attention is based upon, i.e., it generally is more effective on the data distribution having sequential nature. Its success in the NLP domain explains the potential it has but for our dataset the hypothesis that self-attention could be game changer was nullified. However, CNN proved to be more effective with tabular dataset. In Figure 8.3, the gradual increase in accuracy on the left while on the right f1 score, precision and recall on the validation set in 10-fold cross validation explains steady learning of the model during training.

4.) LSTM for technical data – As explained earlier, technical data is a time series data which makes LSTM an ideal candidate to solve the problem. In this capacity, we employed four variants of LSTM on technical data and all of them were equally effective but with the score to improve further. We performed few experiments to nullify our hypothesis that a recurrent neural network like LSTM (Long Short-Term Memory) is powerful in predicting and learning time series-based data. We explored and approached the problem in several ways - multivariate (two features), multivariate with MLP wherein the hidden state of the last LSTM layer is fed into the MLP layer because it can learn non-linearity. The approach to mitigate position bias proved to be effective. From the data analysis we found, generally winners are among the first few horses i.e., technically in the output tensor, around 70% approx., winner is in the top 5-7 indexes which makes the model learn that probability of a winner in the top 5 indexes is quite high, thus making it prone to position bias. We shuffled the position of winner in both the input and output tensor which also helped in oversampling the data. Univariate implementation was also successful wherein we used to different LSTM layers in the neural network one each for traded volume and last traded price attribute. The results were not so exciting, accuracy ranging around 30-31% but it can perform better. One of the reasons after an average accuracy is the amount of data. We also inferred of all the models LSTM took maximum time/number of epochs to train because of its complexity.

4.1.4 Ensembled Hybrid Model - Novel Approach Critical Analysis

Ensembled Hybrid model is a novel implementation that is trained over macro (fundamental data) and mini (technical data) data. The results of this model are quite interesting having an accuracy of above 90% for the test set. The figure is questionable, and it is most likely overfitting the data since we could not test it against a different test set like we did for fundamental data (used 2019 dataset primarily for training and validation while 2021 dataset purely for testing), thought the test set (20% of the dataset) used was also unseen for the model. In the absence of a different test set, we ran several experiments with different settings to verify and validate the high accuracy value. We trained the model with different batch sizes, a smaller number of epochs, also varied the learning rate. The outcome of the experiments was similar in all the settings except with one for the variation in number of epochs. This does reduce the overfitting theory or the claim the data was not original. However, from the accuracy curve, we can infer that the increment was gradual and smooth enabling the model to learn over several epochs. Moreover, validation loss was near about constant on reaching 30-45 epochs. Early stopping should help to avoid the model from memorizing the data.

Overall, the hypothesis to concatenate the embeddings of fundamental data and technical data did improve accuracy of predicting horse race outcomes. But results should be further confirmed by testing the model using different test sets.

Refer Appendix B - For all the model variants performance summary report.

Chapter 5 Conclusion

In this last chapter, we provide an overall view of the project and outline its key aspects and findings that can be taken away as a learning. We review the hypothesis drawn in Chapter 2, challenges faced followed by the improvements, alternative approaches not limited to the models developed as part of this project but ideas to build up the next model with new functionality.

5.1.1 Discussion

We started the project by defining the hypothesis and objectives to fulfill as part of this project. In order to test the hypothesis, we performed several experiments before withdrawing the conclusion. The central idea of the project was to devise a model that can efficiently and effectively predict horse race outcomes. The crucial stage in developing a model is identify the feature, its underlying distribution and draw an inference if the principle of the selected algorithm suitable for the given feature distribution and problem statement. We did a thorough background research to understand the related works, reason for the poor accuracy and the good ones. This helped us in constructing the appropriate hypothesis, solving which would solve the actual problem. It was from the research work came to know majority of the horse race prediction models are trained on macro (fundamental data) thus missing the valuable insight which can be brought to the table by the marker odds.

In this capacity, we came up with the primary research question i.e., the consequence of using fundamental and technical data together to train a deep learning model. We preferred deep learning algorithms over machine learning algorithms based upon the past work and the lack of domain knowledge (knowledge about horse racing and betting). Deep Learning has evolved immensely over a past few years and the algorithm options are plenty. Moreover, we had two different distributions of data – fundamental data was a tabular data while technical data was a time-series based data. Consequently, we constructed another hypothesis to investigate MLP and CNN for fundamental data while LSTM given its network architecture for technical data because it has capability to retain the past information in the form of hidden states.

With these assumptions, we developed the novel model using both fundamental and technical data which proved to be effective based upon the outcomes of experiments conducted. We developed in total nine model variants. LSTM model with just market odds also had a reasonable accuracy and precision, close to 0.31-0.32. We also used self-attention mechanism so that the model can learn relevant features and assign more weight to those specific features. However, it did not produce better results.

The major challenge was to load the data because of its size and limited computational power. This was partially solved by using Google Colab Pro.

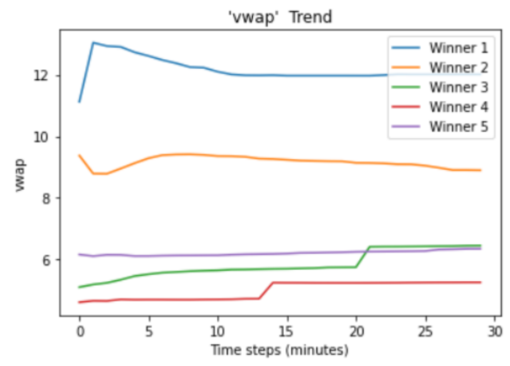
5.1.2 Future Work

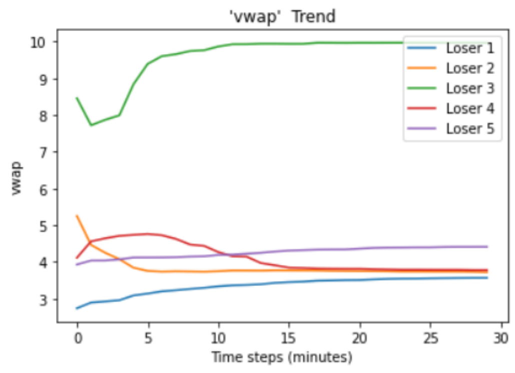
The novel ensembled hybrid model implemented did obtain good results, but a high accuracy and f1 score raised question the model's ability to generalize on the unseen data. The question raised was inconclusive though we did perform multiple experiments with different settings. However, this model can be further enhanced by gaining more insight from the match reviews, online reviews, fan sentiments and similar contents. These reviews and comments can be crawled and cleaned up. The extracted content can then be processed to read the sentiment of the fans and the experts separately, giving more weights to the experts of course in the first step. Further we can generate distributional embeddings using BERT or other transformer-based language models. These latent space embedding can be concatenated with the embeddings of fundamental data generated using CNN and technical data embeddings generated using LSTM. The concatenated embedding can then be fed into the MLP layer with SoftMax function in the output layer to forecast the winning probability of a race.

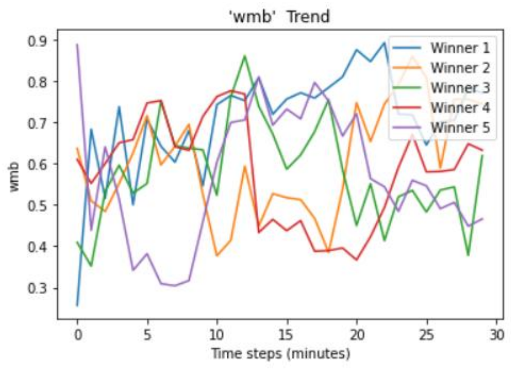
Hyper-parameter tuning is one of the key aspects in building an effective and efficient model. We did it for the model variants empirically, a more suited approach would be to use Bayesian Optimization which creates the probability distribution model of the loss function and applies it to sample hyperparameters to the evaluate the actual loss function.

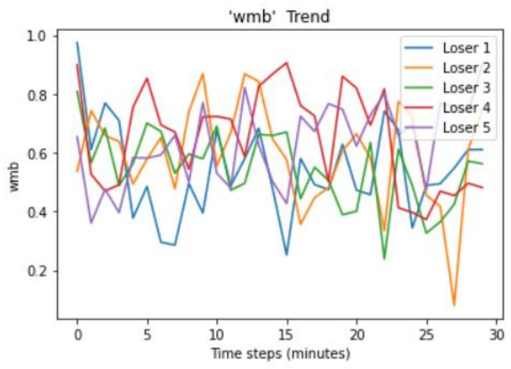
Lastly, an interesting area would be to explore on developing a model which not only forecast the outcome of a race but also recommends a betting strategy for the race.

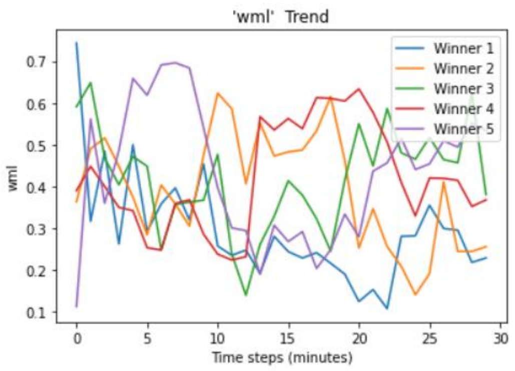
Appendix A Technical Data Exploratory Analysis

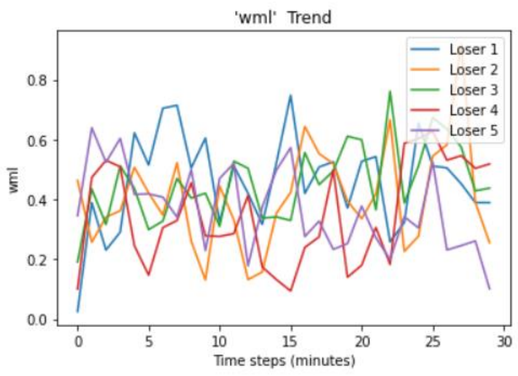


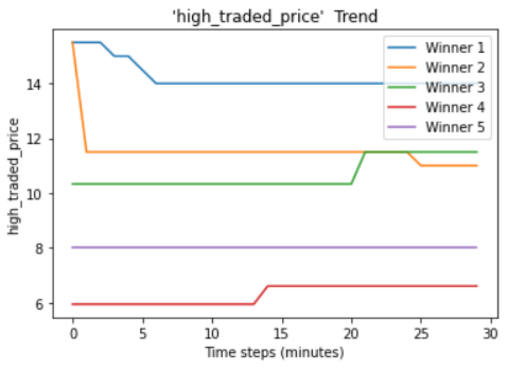


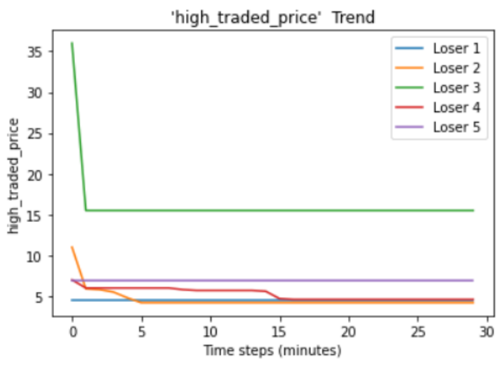


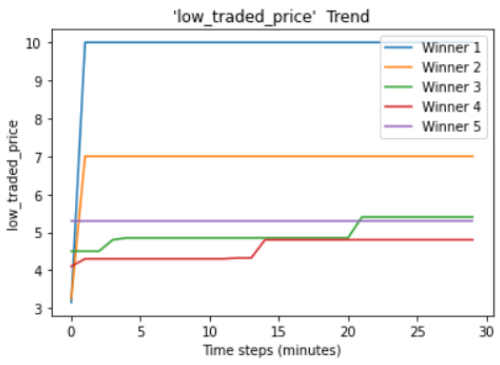




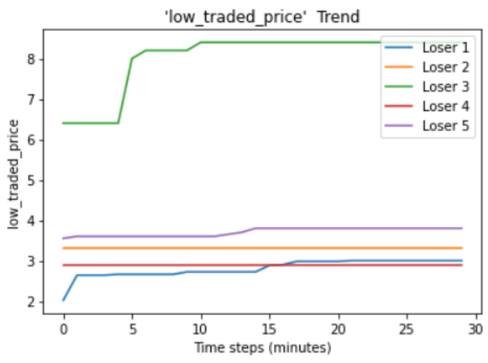








'low_traded_price' Trend
low_traded_price
Time steps (minutes)
Winner 1
Winner 2
Winner 3
Winner 4
Winner 5



'low_traded_price' Trend
low_traded_price
Time steps (minutes)
Loser 1
Loser 2
Loser 3
Loser 4
Loser 5

Appendix B Model Performance Summary Report

Model / Dataset	2019 - Validation Set				2019 - Test Set				2021 - Test Set (Only Fundamental Data)			
	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
MLP	78.31	0.773	0.778	0.783	97.38	0.975	0.973	0.974	69.39	0.742	0.694	0.698
MLP with Self Attention	91.76	0.912	0.918	0.909	99.66	0.996	0.996	0.996	37.23	0.487	0.372	0.350
CNN	60.24	0.618	0.602	0.592	95.78	0.955	0.957	0.955	92.50	0.924	0.925	0.918
CNN with Self Attention	30.07	0.270	0.301	0.235	51.64	0.443	0.516	0.415	47.93	0.384	0.479	0.393
LSTM multi-variate	22.27	0.255	0.223	0.220	22.34	0.239	0.223	0.226	NA	NA	NA	NA
LSTM multi-variate + MLP	24.53	0.292	0.245	0.238	27.46	0.322	0.275	0.278	NA	NA	NA	NA
LSTM Univariate	34.21	0.325	0.312	0.31	31.01	0.338	0.301	0.31	NA	NA	NA	NA
LSTM Univariate + MLP	0.291	0.289	0.312	0.291	30.1	0.311	0.287	0.301	-	-	-	-
Hybrid Ensemble Model	99.63	-	-	-	-	-	-	-	-	-	-	-

References

[1] How much money is wagered on the Kentucky Derby each year? [Internet]. Betfirm.com. 2021 [cited 2022 Jan 21]. Available from: https://www.betfirm.com/how-much-is-bet-on-the-kentucky-derby/

[2] Deep Blue [Internet]. IBM Corporation. 2012 [cited 2022 Jan 21]. Available from: https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/

[3] Vincent J. Former Go champion beaten by DeepMind retires after declaring AI invincible [Internet]. The Verge. 2019 [cited 2022 Jan 21]. Available from: https://www.theverge.com/2019/11/27/20985260/ai-go-alphago-lee-se-dol-retired-deepmind-defeat

[4] Lu D. AI beats professionals at six-player Texas Hold ’Em poker [Internet]. New Scientist. 2019 [cited 2022 Jan 21]. Available from: https://www.newscientist.com/article/2209631-ai-beats-professionals-at-six-player-texas-hold-em-poker/

[5] Engelbrecht, A. P., Computational Intelligence, University of Pretoria, 2007.

[6] Brockwell, P.J., Davis, R.A.: Introduction to timeseries and forecasting, Springer-Verlan, New York,1996.

[7] D. B. Hausch, V. Lo, and W. T. Ziemba. Efficiency of Racetrack Betting Markets. 2008.

[8] Mollick E. Establishing moore’s law. IEEE ann hist comput [Internet]. 2006;28(3):62–75. Available from: http://dx.doi.org/10.1109/mahc.2006.45

[9] C. X. Wong. Precision: Statistical and Mathematical Methods in Horse Racing. 2011.

[10] S. Lessmann, M. Sung, and J. E. V. Johnson. Adapting Least-Square Support Vector Regression Models to Forecast the Outcome of Horse Races. pages 1–19, 2007.

[11] S. Pudaruth, N. Medard, and Z. Dookhun. Horse Racing Prediction at the Champ De Mars Using a Weighted Probabilistic Approach. pages 1–4, 2013.

[12] Riess S. The Cyclical History of Horse Racing: The USA’s Oldest and (Sometimes) Most Popular Spectator Sport. Taylor & Francis Online; 2014.

[13] Allinson NM. Successful prediction of horse racing results using a neural network. In: IEE Colloquium on Neural Networks: Design Techniques and Tools. 1991.

[14] Edelman D. Adapting support vector machine methods for horserace odds prediction. Springer Link; 2006.

[15] Williams J. A case study using neural networks algorithms: horse racing predictions in. 2008.

[16] Elnaz Davoodi ARK. Horse Racing Prediction Using Artificial Neural Networks. RECENT ADVANCES in NEURAL NETWORKS. FUZZY SYSTEMS & EVOLUTIONARY COMPUTING. 2010;

[17] Sameerchand Pudaruth NMBD. Horse Racing Prediction at the Champ De Mars using a Weighted Probabilistic Approach. International Journal of Computer Applications. 2013;72(5).

[18] Sameerchand Pudaruth MJ. Using Artificial Neural Networks to Predict Winners in Horseraces: A Case Study at the Champs de Mars. s.l., IST-Africa. Conference Proceedings. 2015;

[19] Chung W-C. A SVM-Based committee machine for prediction of Hong Kong horse racing. s.l. IEEE; 2017.

[20] Pudaruth S, Jogeeah M, Chandoo AK. Using Artificial Neural Networks to Predict Winners in Horse Races: A Case Study at the Champs de Mars. In: 10th IST-Africa Conference. IST-Africa; Lilongwe, Malawi; 2015. p. 1–8.

[21] Cinar YG, Mirisaee H, Goswami P, Gaussier E, A¨ıt-Bachir A. Period-Aware Content Attention RNNs for Time Series Forecasting with Missing Values. 2018;1–10.

[22] Cinar YG, Mirisaee H, Goswami P, Gaussier E, A¨ıt-Bachir ´. A., Strijov VV. Position-Based Content Attention for Time Series Forecasting with Sequence-to-Sequence RNNs. In: Neural Information Processing - 24th International Conference. Guangzhou, China; 2017. p. 1–10.

[23] Somepalli G. SAINT: Improved Neural Networks for Tabular Data via Row Attention and Contrastive Pre-Training. 2021.

[24] T. Drapkin and R. Forsyth. The Punter's Revenge: Computers in the World of Gambling. 1986.

[25] C. X. Wong. Precision: Statistical and Mathematical Methods in Horse Racing. 2011.