VILNIUS UNIVERSITY

Faculty of Mathematics and Informatics
Master's study programme
Modelling and Data Analysis

Master's thesis

The Modelling and Analysis of Flights Time Delay and Arrival Times Using Deep Neural Networks

Skrydžių vėlavimų ir atvykimo laikų modeliavimas ir analizė taikant giliuosius neuroninius tinklus

Liavonava Anastasiya

Supervisor: Assist., Dr. Linas Petkevičius

VILNIUS (2022)

Skrydžių vėlavimų ir atvykimo laikų modeliavimas ir analizė taikant giliuosius neuroninius tinklus

Santrauka

Nuo 1927 m., kai atsirado komercinės avialinijos, ir iki šiol skrydžių vėlavimas yra svarbi ir neišspręsta aviacijos problema. Tai turi didelį poveikį, daugiausia ekonominį, visiems rinkos dalyviams. Atsižvelgiant į numatomą nuolatinį skrydžių vėlavimo augimą, dabartinis tyrimas yra svarbus, ir galėtų turėti praktinę naudą. Baigiamajame magistro darbe nagrinėjamos skrydžio vėlavimo ir likusio atvykimo laiko prognozavimo problemos, bei įvairių neuroninių tinklų taikymai tam prognozuoti. Tyrimo metu gauta, kad kai kurie modeliai, prognozuojantys tikimybį pasiskirstymą, veikia geriau nei regresijos modeliai.

Raktiniai žodžiai : Regresija, Tikimybinis pasiskirstymas, Gilieji neuroniniai tinklai, Transformeriai, Skrydžio vėlavimo prognozavimas

The Modelling and Analysis of Flights Time Delay and Arrival Times Using Deep Neural Networks

Abstract

Since 1927, when a commercial airline system had been born, and to this moment flights delay is an important and unsolved problem in aviation. It has negative impacts, mainly economic, for all market representatives. In view of the expected continued flights delay growth, the current research is relevant. The thesis is dedicated to the exploration of flight delay problems and different neural networks estimation to predict it. Additionally to that, remaining time before landing was predicted to expand the information provided about delay. It is experimentally proved that models estimating probability-distribution parameters perform better then models estimating value of delay itself.

Keywords : Regression, Probability distribution, Deep neural networks, Transformers, Flight delay

Table of Contents

1 Introduction

The two major challenges aviation faces today are reducing the environmental impact of air traffic and improving the ability of the network to scale capacity up or down depending on growth or decline of traffic [10]. Flights delay, as an important part of those challenges, is still a big issue in aviation field. According to latest available information, in Q3 2021 the average delay on arrival was 10.3 minutes per flight. This was an increase of 5.4 minutes per flight when compared to the same period of 2020 where the average delay per flight was 4.9 minutes [13]. Analysis into the causes of delay shows reactionary delay contributed the most to the average delay per flight at 4.1 minutes, with airline causes ranking second with 3.2 minutes per flight. Delays due to Governmental causes remain in third place contributing 1.0 minute to the average delay per flight, translating into a 16% share of generated primary delay minutes. This cause remains a significant burden to airline and airport operations, being down to the extra time taken, mostly at check-in, needed to check what the destination specifically requires and whether the passengers have the right combinations of COVID19 test and vaccination certificates. ATFM airport and en-route delay despite increasing, remained low at around 0.6 minutes per flight.

Moreover, flight delay have negative impacts, mainly economic, for passengers, airlines, and airports. In view of the expected continued growth, all signs are that the delay situation will deteriorate further and dramatically if stringent actions are not taken [23]. This might confirm the need for the current research.

Due to explained situation, flight delay prediction is an important direction to work in. Of course, most important and complex R&D are done by big companies, connected with aviation directly, like SESAR (Single European Sky ATM Research), Thales (Aerospace company) and ECTL MUAC (Eurocontrol's Maastricht Upper Area Control center). But as average delay is only growing, it's prediction is still actual.

Historic commercial flights data used in the thesis contains plan and fact information about unique numeric identifier for each flight, number of the points crossed by a flight, time (UTC) at which the point was crossed, and three space parameters: latitude, longitude and altitude in flight levels at which the point was crossed. Eleven more features were created additionally, like countries and airports of departure and arrival, time based variables. All data is provided by Eurocontrol - a pan-European, civil-military organization dedicated to supporting European aviation. More information in Chapter 4 Data.

The goal of current research is to apply neural networks to estimate flight delay which were not used before for flight delay prediction. In this thesis we propose to solve flight delay prediction problem with basic regression approach and probabilistic prediction methods that predict not just a single-point estimate but a probability-distribution parameters. We created the Deep neural networks and Transformer models that estimate the delay and the distribution of delay as having parameters being the output of a neural network. More information about chosen models in Chapter 2 Preliminaries.

All above, several model types used to apply for delay prediction in the thesis. The basic model is linear regression (OLS) which was used as a starting point to compare with advanced models. It was chosen by purpose, as there are a list of papers which predict flight delay exactly by this method. Additionally, two deep-neural network (DNN) model types were trained: DNN regression and DNN probabilistic (estimate Normal and Weibull distributions parameters). Moreover, a new approach - Transformer neural network (TNN), based solely on attention mechanisms, dispensing with recurrence and convolutions entirely, was applied to estimate delay itself (TNN regression) and distribution parameters (TNN probabilistic for Normal and Weibull distributions).

All the NN models were trained on same data, with same loss functions (regression - with MSE, probabilistic - with negative log likelihood), with same optimizer (Adam), on same number of epochs (50). There are a range of techniques applied to optimize the models (see Chapter 5 Experiments). The results obtained show that regression model as predicting mechanism is not the best choice. In contrast to this, models estimating probability-distribution parameters perform better. Moreover, probability-distribution parameters can provide user with information about confidence interval for a given confidence threshold.

After the comparison of test metrics it become clear that DNN models performs better. Probabilistic models that estimated Weibull distribution parameters are the most productive. Final model architecture which show highest performance, presented in Chapter 5 Experiments.

Summing up the above, the aim of the thesis is to investigate model(s) that learns long term dependencies in sequences in order to estimates the distribution of delay time and make inference about waiting time.

The objectives to achieve the goal:

- make a scientific literature review
- prepare real world data, and covariates for experiments
- trajectories visualization
- investigate models which are capable to estimates the delay and it's distribution
- experiments with different application fields

2 Literature review

This chapter is built from three parts. First one is about used flights data, other projects based on it and relevance of the topic. Second – about recurrent neural networks (RNN) as models which able to estimate the distribution of time to the next event. And the final part is devoted to an overview of machine learning techniques for trajectory prediction.

Historic commercial flights data used in the thesis contains following information about unique numeric identifier for each flight, number of the points crossed by a flight, time (UTC) at which the point was crossed, and three space parameters: latitude, longitude and altitude in flight levels at which the point was crossed. More information about data features and its' analysis are presented in Chapter 4 Data. All data is provided by Eurocontrol - a pan-European, civil-military organization dedicated to supporting European aviation (https://www.eurocontrol.int/). Its R&D Data Archive gives researchers and data scientists access to detailed flight data of 12 million commercial flights across the European network that led to a huge boost in AI and machine-learning applications. Eurocontrol itself developed a number of AI based applications and are working on more. Currently more than thirty AI-based applications are under development in different frameworks notably in the Network of Innovation Labs and SESAR.

The data used in the thesis is used in several up-to-date projects and researches as part of big data. Of course, most important and complex R&D are done by big companies, connected with aviation directly, like SESAR (Single European Sky ATM Research), Thales (Aerospace company) and ECTL MUAC (Eurocontrol's Maastricht Upper Area Control center). Among other things they work under traffic predictions, forecasts, modeling in order to improve predictions of aircraft trajectories, reducing uncertainty and increasing capacity. This will be enabled by a by a digital transformation of the underlying infrastructure system, characterized by a significant increase in levels of automation and connectivity [23].

Except outstanding representatives of the aviation market, smaller market players, universities, individual researchers and data scientists use historic commercial flights data for independent research, like [28], [9] and [14]. Some papers are closer to the thesis and explore the problem of delays. For example, in [3] authors say that a major factor in increased airspace efficiency and capacity is accurate prediction of Estimated Time of Arrival and introduce prediction system for commercial flights. Researches from Delft University of Technology propose a method to predict the uncertainty on an Estimated Time of Arrival by estimating distributions using historic data [18].

According to Eurocontrol, the two major challenges aviation faces today are reducing the environmental impact of air traffic and improving the ability of the network to scale capacity up or down depending on growth or decline of traffic [10]. We reckon data science combined with machine learning have the potential to tackle those problems.

Flights delay, as an important part of those challenges, is still a big issue in aviation field, for example, according to latest available information, in Q3 2021 the average delay per flight on arrival was 10.3 minutes per flight. This was an increase of 5.4 minutes per flight when compared to the same period of 2020 where the average delay per flight was 4.9 minutes [13]. Moreover, they have negative impacts, mainly economic, for passengers, airlines, and airports. In view of the expected continued growth, all signs are that the delay situation will deteriorate further and dramatically if stringent actions are not taken [23]. This might confirm the need for further research. Using data science as a part of a problem-solving instrument enables more accurate predictions and more sophisticated tools to increase productivity and improve decision-making.

According to 2020 Transport Reviews [6] the main problems related to flight delay prediction from Data Science methods perspectives are: statistical analysis, probabilistic models, network representation, operational research and machine learning. Statistical analysis usually encompasses the use of regression models, correlation analysis, econometric models, parametric/non-parametric tests, and multivariate analysis. It mostly used for delay propagation effects and estimation the costs of delays. Probabilistic Models encompass analysis tools that estimate the probability of an event based on historical data. Network representation encompasses the study of flight systems according to a graph theory. Papers there is more oriented for delay propagation, airports' schedules and comparison. Operational Research includes advanced analytical methods (such as optimization, simulations, and queue theory). Machine learning method is a leading one, as has the biggest number of papers and high performance metrics. The methods commonly used include k-Nearest Neighbor, Neural Networks, Support Vector Machine, Fuzzy Logic, and Random Forests [6]. Mentioned above [9] worked under delay from economic perspective - the authors estimate European airline delay cost values. Interesting approach used by Balakrishna at [5] and [4], where reinforcement learning algorithm implemented to predict taxi-out delays. In one of the latest paper [27] flight delay prediction based on deep learning and Levenberg-Marquart algorithm.

In this thesis a bit specific approach used compared to existing ones. E. Martinsson paper [19] was accepted as the starting base, where the author proposed a model that estimates the distribution of time to the next event (TTE) as having a discrete or continuous Weibull distribution with parameters being the output of a recurrent neural network (RNN). The model is trained using a special objective function commonly used in survival analysis. The author describes the Weibull RNN-TTE using a general framework for censored data which can easily be extended with other distributions and adapted for multivariate prediction. Basically, this is a simple framework for time-series TTE prediction applicable when we have any or all of the problems of continuous or discrete time, right censoring, recurrent events, temporal patterns, time varying covariates or time series of varying lengths. For example, customer churn, remaining useful life, failure, spike-train and event prediction.

Exactly this paper provide me with idea to estimate parameters of probability distribution of flight delay, but not delay itself. E. Martinsson idea [19] was citated during 2018-2021 on `https://www.webofscience.com/` and `https://scholar.google.com/` in total in 12 publications.

In [1] authors worked under one of key challenges in predictive maintenance – prediction of the impending downtime of equipment with a reasonable prediction horizon so that countermeasures can be put in place. Weibull RNN-TTE was used to learn the underlying failure dynamics. Also it can leverage the non-failed equipment data for remaining useful life estimation. Their colleagues went further and created a full system for real-time distributed prognostics [21].

Researches from University of Toronto used RNN-TTE in estimation of multivariate arrival times for personalized demand forecasting [7]. This paper is interesting because model directly the inter-arrival times as well as the partially observed information at each time step to model purchase times jointly over several products. This paper is interesting because authors model directly the inter-arrival times as well as the partially observed information at each time step to model purchase times jointly over several products. So they adapted RNN-TTE for multivariate prediction.

Quite interesting application of RNN-TTE was proposed by researchers from Arkanzas University [30]. They aim to detect and prevent fraudulent activities via survival analysis based fraud early detection model, which maps dynamic user activities to survival probabilities that are guaranteed to be monotonically decreasing along time.

Not trivial is paper [20], where authors consider the problem of future event prediction in video. It shows the ability to predict events far in the future, up to 10 seconds before they occur; and it can determine which areas of the image sequence are responsible for these predictions. The main difference here in definition of TTE itself. In [15] TTE is the time when the observed system enters first a certain state of interest. The system may enter and leave the state of interest several times or it may already be in said state at time 0, or the event may not occur at all in the near future. During the work, number of models were tried, in the result GMMH (hybrid between a heatmap and a Mixture of Gaussians) outperformed Weibull RNN-TTE.

The author of [19] used his master thesis ideas in [15] in combination with [2] for musical onset detection problem (task of finding the starting points of all relevant musical events in audio signals).

Finally, Weibull TTE-RNN by [19] was used as individually conditional breast cancer recurrence prediction model in [16]. Authors compare its performance with such models as logistic regression, random forest, and gradient boosting (because they are the methods most typically used in medical applications) and show by practical example that Weibull TTE-RNN outperforms the existing machine learning-based models.

Part of this thesis' objectives is achieved by prediction of trajectory, in particular identification of hot spots and precise airspace division.

Most trajectory prediction algorithms work for land traffic, which rely on points of interest (POIs) and are only suitable for stationary road condition. Compared with land traffic prediction, flight trajectory prediction is very difficult because way-points are sparse and the flight envelopes are heavily affected by external factors.

Generally, flight trajectory prediction underpins much of the functionality of air traffic management systems, both in the tactical (air traffic control) and pre-tactical (air traffic flow and capacity management) phases of a flight. Systems in use today generally apply predefined rules and models to predict trajectories from available input data. Prediction logic is static and is grounded on domain knowledge of human experts and kinematic equations.

By now, accuracy of the predicted trajectories is far from perfect, degrading performance of the Air Traffic Management System. The Eurocontrol forecasts a strong increase of the European air traffic till the year 2035. This growth justifies the development of new concepts and tools in order to ensure services to airspace users. Trajectory prediction is at the core of these developments.

In general, existing approaches are the following: kinematics and dynamics-based approach (KDA), filter-based approach and its variations, machine learning-based approach. KDA approach splits flight operation into different phases, takes into account flight dynamics and aircraft performance, uses kinematics equations. Filter-based approach mostly includes articles dedicated to the Kalman filter use to build the motion state transitions between two prediction instants. The core idea of machine learning type of approach is mining frequent transition patterns from historical trajectories, which is further used to build the trajectory patterns of the predicted flights. Taking into account the peculiarities of this thesis, we will consider literature with the use of machine learning-based approaches.

Back in 2018, Eurocontrol itself in [12] presented a feed-forward NN with 3 hidden layers containing 170 units each is used (implemented in TensorFlow). In practice, the predicted route is used to construct a 'what if?' request for the Flight Data Processing (FDP) system. The request triggers the system to predict a 4D trajectory using its internal logic but constraining the route to the coordinates provided in the request. The 'what if?' trajectory is maintained in parallel to the original trajectory, both are displayed as traffic load to users (supervisors and flow managers).

Talking about the main R&D institutions in aviation, Data-driven Aircraft Trajectory Prediction Research (DART) [11] was presented within SESAR 2020 Exploratory Research. DART addresses the trajectory prediction task by combining elements of three approaches: hidden Markov models, clustering and regression. Within the same SESAR Research another idea was presented [25] about short-term trajectory prediction model, which combines big preprocessing step (including PCA and density-based clustering) and Multi-Cells Neural Networks technique will be applied to generate the predicted trajectory for different patterns.

Gaussian processes were also studied in relation to the flight path.

For example, researches from University of California in [17] model the track points on trajectories as conditional Gaussian mixtures with parameters to be learned from proposed deep generative model, which is an end-to-end convolutional RNN that consists of a long short-term memory (LSTM) encoder network and a mixture density LSTM decoder network.

Also, one of the latest work about flight trajectory prediction proposes to use a deep Gaussian process based framework [8]. The Gaussian distribution is applied to serve as the probabilistic representation for illustrating the transition patterns of the flight trajectory, based on which a stochastic process is generated to build the temporal correlations among flight positions, i.e., Gaussian process. The model has the ability of predicting both the flight trajectory and its confidence interval.

Neural networks were also developed to predict flight trajectory by fitting the data distribution (i.e., flight transition patterns).

4D trajectory prediction model based on the backpropagation (BP) neural network was studied in [26]. It shows high performance: predicted output is close to the real flight data, and the time error at the crossing point is no more than 1 min and the altitude error at the crossing point is no more than 50 m, which is of high accuracy.

In [29] two types of deep learning models are trained: deep feed-forward neural networks (to make a one-step-ahead prediction on the deviation along latitude and longitude between the actual flight trajectory and target flight trajectory) and deep Long Short-Term Memory neural networks (to make longer-term predictions on the flight trajectory over several subsequent time instants). Then the two different types of deep learning models are blended together to create a multi-fidelity prediction. The model prediction uncertainty is characterized following a Bayesian approach. Except [29], Long Short-Term Memory (LSTM) approach was used in [22].

3 Preliminaries

3.1 Problem type

In statistical modeling, regression analysis is a set of statistical processes for estimating the relationships between a dependent variable and one or more independent variables. The most common form of regression analysis is linear regression, in which one finds the line (or a more complex linear combination) that most closely fits the data according to a specific mathematical criterion.

Regression analysis is primarily used for two conceptually distinct purposes. First, regression analysis is widely used for prediction and forecasting, where its use has substantial overlap with the field of machine learning. Second, in some situations regression analysis can be used to infer causal relationships between the independent and dependent variables.

In practice, researchers first select a model they would like to estimate and then use their chosen method (e.g., ordinary least squares) to estimate the parameters of that model. Regression models involve the following components:

- The unknown parameters, often denoted as a scalar or vector β
- The independent variables, which are observed in data and are often denoted as a vector X_i
- The dependent variable, which are observed in data and often denoted using the scalar Y_i
- The error terms, which are not directly observed in data and are often denoted using the scalar e_i

Most regression models propose that Y_i is a function of X_i and β, with e_i representing an additive error term that may stand in for un-modeled determinants of Y_i or random statistical noise 1.

$$Y_i = f(X_i, \beta) + e_i \tag{1}$$

The researchers' goal is to estimate the function $f(X_i, \beta)$ that most closely fits the data. Once researchers determine their preferred statistical model, different forms of regression analysis provide tools to estimate the parameters β. For example, least squares finds the value of β that minimizes the sum of squared errors $\sum_i (Y_i - f(X_i, \beta))^2$. A given regression method will ultimately provide an estimate of β, usually denoted $\hat{\beta}$ to distinguish the estimate from the true (unknown) parameter value that generated the data. Using this estimate, the researcher can then use the fitted value $\hat{Y_i} = f(X_i, \hat{\beta})$ for prediction or to assess the accuracy of the model in explaining the data.

By itself, a regression is simply a calculation using the data. In order to interpret the output of a regression as a meaningful statistical quantity that measures real-world relationships, researchers often rely on a number of classical assumptions. These often include:

- The sample is representative of the population at large.
- The independent variables are measured with no error.
- Deviations from the model have an expected value of zero, conditional on covariates: $E(e_i|X_i) = 0$
- The variance of the residuals e_i is constant across observations (homoscedasticity).
- The residuals e_i are uncorrelated with one another.

Regressions might be linear and nonlinear.

Linear regression is a linear approach for modelling the relationship between a scalar response and one or more explanatory variables. The case of one explanatory variable is called simple linear regression; for more than one, the process is called multiple linear regression.

Linear regression was the first type of regression analysis to be studied rigorously, and to be used extensively in practical applications. This is because models which depend linearly on their unknown parameters are easier to fit than models which are non-linearly related to their parameters and because the statistical properties of the resulting estimators are easier to determine.

Given a data set $\{y_i, x_{i1}, ..., x_{ip}\}_{i=1}^{n}$ of n statistical units, a linear regression model assumes that the relationship between the dependent variable y and the p-vector of regressors x is linear. This relationship is modeled through a disturbance term or error variable e — an unobserved random variable that adds "noise" to the linear relationship between the dependent variable and regressors. Thus the model takes the form by 2.

$$y_i = \beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip} + e_i = X_t^T \beta + e_i, \quad i = 1, ..., n \tag{2}$$

A fitted linear regression model can be used to identify the relationship between a single predictor variable x_j and the response variable y when all the other predictor variables in the model are "held fixed". Specifically, the interpretation of $_j$ is the expected change in y for a one-unit change in x_j when the other covariates are held fixed—that is, the expected value of the partial derivative of y with respect to x_j. This is sometimes called the unique effect of x_j on y. In contrast, the marginal effect of x_j on y can be assessed using a correlation coefficient or simple linear regression model relating only x_j to y; this effect is the total derivative of y with respect to x_j.

A large number of procedures have been developed for parameter estimation and inference in linear regression. These methods differ in computational simplicity of algorithms, presence of a closed-form solution, robustness with respect to heavy-tailed distributions, and theoretical assumptions needed to validate desirable statistical properties such as consistency and asymptotic efficiency. Some of the more common estimation techniques for linear regression are: least-squares estimation (ordinary, weighted and generalized), maximum-likelihood estimation (when error terms belong to a certain parametric family $\check{C}_\theta$ of probability distributions, Ridge regression, Lasso regression), least absolute deviation and adaptive estimation. Of course, there are a range of other estimation techniques, like Bayesian linear regression, quantile regression, least-angle regression, mixed models and more.

When the model function is not linear in the parameters, the sum of squares must be minimized by an iterative procedure. In nonlinear regression observational data are modeled by a function which is a nonlinear combination of the model parameters and depends on one or more independent variables. The data are fitted by a method of successive approximations. In general, form of statistical model has a form 3, relates a vector of independent variables $\mathbf{x}$, and its associated observed dependent variables, $\mathbf{y}$.

$$\mathbf{y} \sim f(\mathbf{x}, \boldsymbol{\beta}) \tag{3}$$

The function $\mathbf{f}$ is nonlinear in the components of the vector of parameters $\boldsymbol{\beta}$, but otherwise arbitrary. This function is nonlinear because it cannot be expressed as a linear combination of the two βs.

Examples of nonlinear functions include exponential functions, logarithmic functions, trigonometric functions, power functions, Gaussian function, and Lorentz distributions. Some functions, such as the exponential or logarithmic functions, can be transformed so that they are linear. When so transformed, standard linear regression can be performed but must be applied with caution.

In general, there is no closed-form expression for the best-fitting parameters, as there is in linear regression. Usually numerical optimization algorithms are applied to determine the best-fitting parameters. Again in contrast to linear regression, there may be many local minima of the function to be optimized and even the global minimum may produce a biased estimate. In practice, estimated values of the parameters are used, in conjunction with the optimization algorithm, to attempt to find the global minimum of a sum of squares.

3.2 Probability distributions

Probability distributions are statistical functions that describe the likelihood of obtaining possible values that a random variable can take. In other words, the values of the variable vary based on the underlying probability distribution.

To define probability distributions for the specific case of random variables (so the sample space can be seen as a numeric set), it is common to distinguish between discrete and continuous random variables. A probability distribution can be described in various forms, such as by a probability mass function or a cumulative distribution function. One of the most general descriptions, which applies for continuous and discrete variables, is by means of a probability function $P : \mathcal{A} \to R$ whose input space $\mathcal{A}$ is related to the sample space, and gives a real number probability as its output.

The probability function P can take as argument subsets of the sample space itself. However, because of the widespread use of random variables, which transform the sample space into a set of numbers, it is more common to study probability distributions whose argument are subsets of these particular kinds of sets (number sets). It is common to denote as $P(X \in E)$ the probability that a certain variable X belongs to a certain event E.

The probability function only characterizes a probability distribution if it satisfies all the axioms:

- $P(X \in E) \geq 0 \quad \forall E \in \ \mathcal{A}$, so the probability is non-negative
- $P(X \in E) \leq 1 \quad \forall E \in \ \mathcal{A}$, so no probability exceeds 1
- $P(X \in \bigcup_i E_i) = \sum_i P(X \in E_i)$ for any disjoint family of sets E_i.

Probability distributions usually belong to one of two classes. A discrete probability distribution is applicable to the scenarios where the set of possible outcomes is discrete (e.g. a coin toss, a roll of a die) and the probabilities are encoded by a discrete list of the probabilities of the outcomes; in this case the discrete probability distribution is known as probability mass function. On the other hand, continuous probability distributions are applicable to scenarios where the set of possible outcomes can take on values in a continuous range (e.g. real numbers), such as the temperature on a given day. In the case of real numbers, the continuous probability distribution is described by the cumulative distribution function. In the continuous case, probabilities are described by a probability density function, and the probability distribution is by definition the integral of the probability density function.

A continuous probability distribution is a probability distribution on the real numbers with a lot of possible values, and where the probability of any event can be expressed as an integral 4. More precisely, a real random variable X has a continuous probability distribution if there is a function $f : R \to [0, \infty]$ such that for each interval $[a, b] \subset R$ the probability of X belonging to $[a, b]$ is given by the integral of f:

$$P(a \leq X \leq b) = \int_a^b f(x)\, dx \tag{4}$$

This is the definition of a probability density function, so that continuous probability distributions are exactly those with a probability density function. There are many examples of continuous probability distributions: normal, chi-squared, and others.

Flight delay falling within a particular range of values, it takes values from 0 and not limited from above in theory, but always limited on practice with maximum value from the investigated data sample (in the preprocessed data frame this value in 175 min). This is continuous case, flight delay might possible take a huge range of values, hardly counted. For this random variable probability density function is suitable type of probability distribution.

3.3 The selected distributions

To analyse probability distribution suitable for of delay time, we need to visualize approximate representation of its' distribution on figure 1. In conjunction with analysis of delay, time remaining until the end of a flight is analysed too (to expand the information provided about delay) and it's histogram is on figure 2.

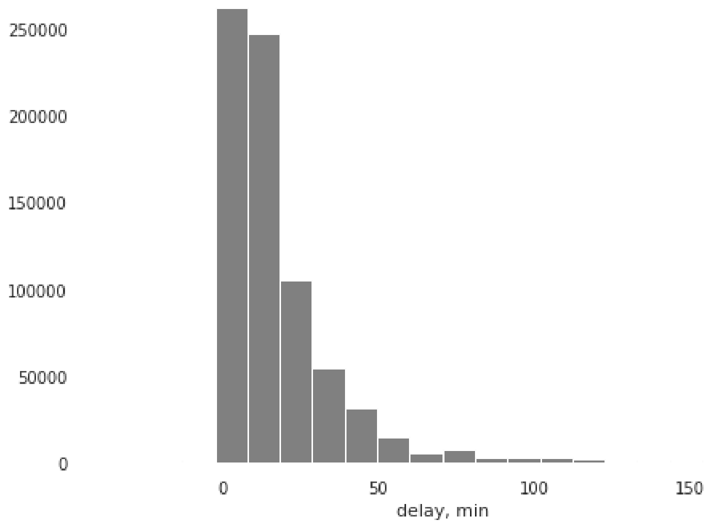

Figure 1: Delay time histogram

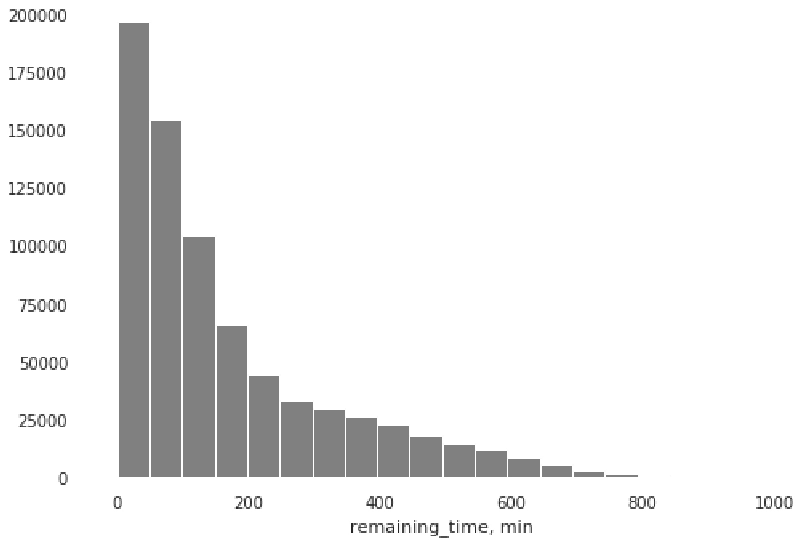

Figure 2: Remaining time histogram

For prediction of parameters of probability distribution of delay, two types of distributions were selected - Normal and Weibull.

Normal (or Gaussian) distribution is a type of continuous probability distribution for a real-valued random variable. The general form of its probability density function is equation 6.

$$X \sim \mathcal{N}(\mu,\, \sigma^2)\,. \tag{5}$$

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) \tag{6}$$

A random variable X with a Gaussian distribution is said to be normally distributed 5, and is called a normal deviate. The parameter μ is the mean or expectation of the distribution (and also its median and mode), while the parameter σ is its standard deviation. During training exactly those two parameters (μ and σ) were estimated.

The normal distribution with density $f(x)$ (mean μ and standard deviation σ) has the following main properties:

- It is symmetric around the point $x = \mu$, which is at the same time the mode, the median and the mean of the distribution.
- It is unimodal: its first derivative is positive for $x < \mu$, negative for $x > \mu$, and zero only at $x = \mu$
- The area bounded by the curve and the x-axis is unity (i.e. equal to one)
- About 68% of values drawn from a normal distribution are within one σ away from μ; about 95% of the values lie within two σs; and about 99.7% are within three σs.
- Its density has two inflection points, located one standard deviation away from the mean, namely at $x = \mu - \sigma$ and $x = \mu + \sigma$
- It is the only distribution whose cumulants beyond the first two (i.e., other than the mean and variance) are zero
- It is also the continuous distribution with the maximum entropy for a specified mean and variance.
- Its density is log-concave, infinitely differentiable, indeed supersmooth of order 2

Normal distributions are initial choice to use, because they are important in statistics (partly due to the central limit theorem) and often used. Moreover, Gaussian distributions have some unique properties that are valuable in analytic studies. Also many results and methods can be derived analytically in explicit form when the relevant variables are normally distributed.

Weibull distribution is a type of continuous probability distribution for random variable with 8 probability density function.

$$X \sim \mathcal{W}(k,\, \lambda)\,. \tag{7}$$

$$f_X(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{k}\right)^{k-1} \exp\bigl(-(x/\lambda)^k\bigr) & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{8}$$

k is a positive shape parameter - it affects the shape of a distribution rather than simply shifting it or stretching, λ is a positive scale parameter of the distribution - responsible for how much the distribution is spread out. During training exactly those two parameters (k and λ) were estimated.

The Weibull distribution has the following main properties:

- It is related to a number of other probability distributions; in particular, it interpolates between the exponential distribution ($k = 1$) and the Rayleigh distribution ($k = 2$ and $\lambda = \sqrt{2}\sigma$)
- It is a generalized gamma distribution with both shape parameters equal to k
- The form of the density function changes drastically with the value of k. The density function has infinite negative slope at $x = 0$ if $0 < k < 1$, infinite positive slope at $x = 0$ if $1 < k < 2$ and null slope at $x = 0$ if $k > 2$. For $k = 1$ the density has a finite negative slope at $x = 0$. For $k = 2$ the density has a finite positive slope at $x = 0$. As k goes to infinity, the Weibull distribution converges to a Dirac delta distribution centered at $x = \lambda$

While in Normal distribution mean μ and variance σ^2 are the parameters of probability density function itself, in the Weibull distribution they can be computed via formulas 9 and 10.

$$E(X) = \lambda\Gamma(1 + 1/k) \tag{9}$$

$$var(X) = \lambda^2[\Gamma(1 + 2/k) - (\Gamma(1 + 1/k))^2] \tag{10}$$

The Weibull distribution is widely used in survival analysis, reliability engineering and failure analysis. Weibull distribution is extremely useful to use, because it is is related to a number of other probability distributions. Also, Weibull distribution is good to use for current task, because no delay can happen before time zero.

3.4 Deep neural networks

A deep neural network (DNN) is a neural network with a certain level of complexity, a neural network with more than two layers. Deep neural networks use sophisticated mathematical modeling to process data in complex ways. Each mathematical manipulation as such is considered a layer, and complex DNN have many layers, hence the name "deep" networks.

Basically, DNNs can be viewed as defining a function that takes an input (observation) and produces an output (decision): $f_\theta : X \to Y$. A common use of DNNs is really the definition of a class of such functions (where members of the class are obtained by varying parameters, connection weights, or specifics of the architecture such as the number of neurons, number of layers or their connectivity). Mathematical representation of DNN is shown in equation 11.

$$f(x) = E(Y|X = x) = a_0 + a_1z_1 + ... + a_nz_n = \sum_{i=0}^{n} a_iz_i = \sum_{i=0}^{n} a_i\sigma(\sum_{k=0}^{n_i} a_k^i x_k) \tag{11}$$

$$z_n = \sigma(d_0 + d_1x_1 + ... + d_nx_n) \tag{12}$$

Here σ - some non-linear transformations and $a_0, .., a_n, .., b_0, .., b_n, d_0, .., d_n$ - unkown model parameters.

DNNs can model complex non-linear relationships. DNN architectures generate compositional models where the object is expressed as a layered composition of primitives. The extra layers enable composition of features from lower layers, potentially modeling complex data with fewer units than a similarly performing shallow network. Example on DNN structure is shown in 3.

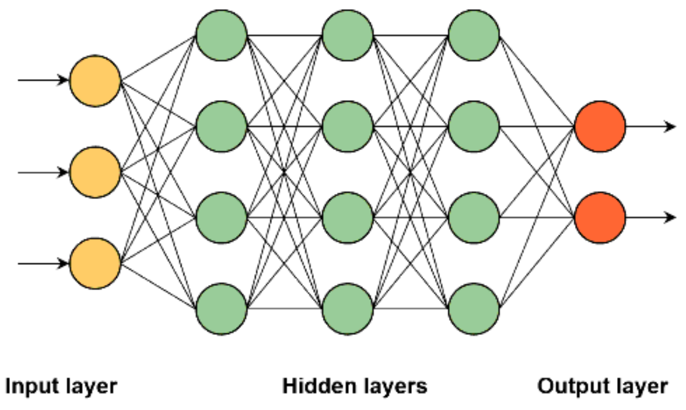

Figure 3: Example of DNN model structure[1]

[1] "Neural Networks with Keras Cookbook", V Kishore Ayyadevara, 2019

DNNs are typically feed-forward networks in which data flows from the input layer to the output layer without looping back. At first, the DNN creates a map of virtual neurons and assigns random numerical values, or "weights", to connections between them. The weights and inputs are multiplied and return an output between 0 and 1. If the network did not accurately recognize a particular pattern, an algorithm would adjust the weights. That way the algorithm can make certain parameters more influential, until it determines the correct mathematical manipulation to fully process the data.

DNNs have achieved great practical success in many machine learning tasks, such as speech recognition, image classification, and natural language processing.

3.5 Transformer neural networks

In 2017 authors of [24] proposed the Transformer, a model architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output. The Transformer is used primarily in the field of natural language processing (NLP) and in computer vision (CV), but can be applied to other tasks instead (in current thesis - regression).

Most competitive neural sequence transduction models have an encoder-decoder structure. Here, the encoder maps an input sequence of symbol representations $(x_1, ..., x_n)$ to a sequence of continuous representations $z = (z_1, ..., z_n)$. Given z, the decoder then generates an output sequence $(y_1, ..., y_m)$ of symbols one element at a time. At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next. The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of 4, respectively.

The encoder is composed of a stack of n identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. Residual connection is employed around each of the two sub-layers, followed by layer normalization.

The decoder is also composed of a stack of n identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, residual connections is employed around each of the sub-layers, followed by layer normalization. Also the self-attention sub-layer is modified in the decoder stack to prevent positions from attending to subsequent positions.

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

Attention mechanism is the most important part of the Transformer. Multi-Head Attention is h parallel running attention layers, or heads. Single attention layer called Scaled Dot-Product Attention 14.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{13}$$

$$\begin{gathered} MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \\ where\ head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \end{gathered} \tag{14}$$

Its' input consists of queries and keys of dimension d_k, and values of dimension d_v. Dot products of the query with all keys computed, each divided by $\sqrt{d_k}$, then softmax function applied to obtain the weights on the values. In practice, the attention function is computed on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V. The matrix of outputs computed by formula 13.

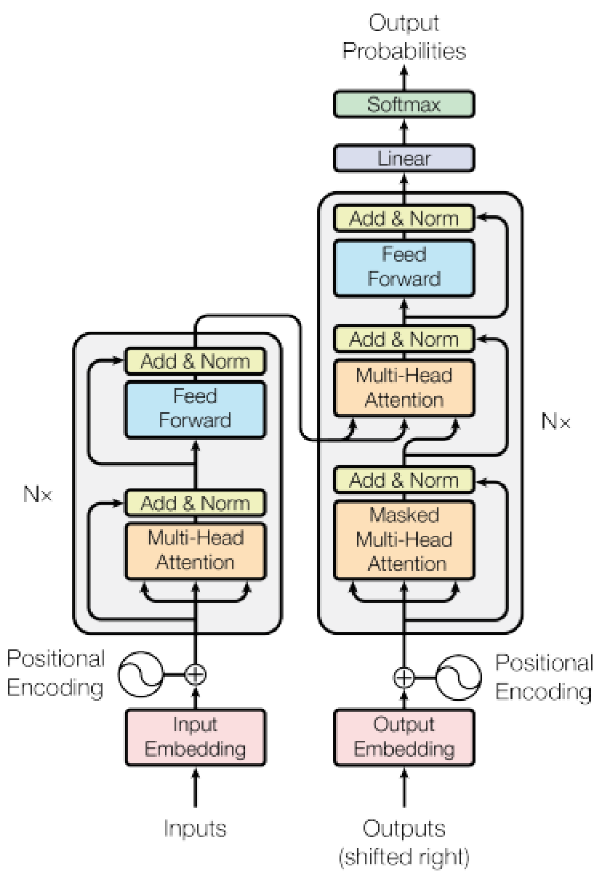

Figure 4: The Transformer - model architecture[2]

3.6 Loss functions

The idea behind Chapter 5 Experiments is that delay might be estimated as single-point (regression model) or as probability-distribution parameters (probabilistic model). Based on this assumption, different loss functions might be applied to train predictive models.

For regression models mostly such loss functions are used: Mean Squared Error (MSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), coefficient of determination (R^2).

The most spread one ans used in current thesis - MSE. It measures the average of the squares of the errors 15. The MSE is a measure of the quality of an estimator. As it is derived from the square of Euclidean distance, it is always a positive value that decreases as the error approaches zero.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{15}$$

In regression analysis the mean of the distance from each point to the predicted regression model can be calculated, and shown as the mean squared error. The squaring is critical to reduce the complexity with negative signs. To minimize MSE, the model could be more accurate, which would mean the model is closer to actual data. One example of a linear regression using this method is the least squares method.

It is clear, that for a model that estimate probability-distribution parameters of the delay, MSE can not be used as a loss function. During training, Normal and Weibull distributions predicted parameters are applied, and loss function should measure their wellness in comparison with real data 17.

For this purpose Maximum likelihood estimation (MLE) was chosen. MLE is a method of estimating

[2]"Attention Is All You Need", Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin, 2017

the parameters of an assumed probability distribution, exactly what we are trying to do with the neural networks. This is achieved by maximizing a likelihood function so that, under the assumed statistical model, the observed data is most probable. But as used models are minimizing error to optimize the models' result, negative MLE is used for probabilistic models 16. The logic of maximum likelihood is both intuitive and flexible, and as such the method has become a dominant means of statistical inference.

$$l(\theta, y) = -lnL_n(\theta, y) \tag{16}$$

$$\theta = \begin{cases} (\mu,\, \sigma) & \text{if } \theta \text{ is parameters of Normal distribution} \\ (k,\, \lambda) & \text{if } \theta \text{ is parameters of Weibull distribution} \end{cases} \tag{17}$$

4 Data

4.1 Resources and description

All data is provided by Eurocontrol - a pan-European, civil-military organization dedicated to supporting European aviation (https://www.eurocontrol.int/). Its R&D Data Archive gives researchers and data scientists access to detailed flight data of 12 million commercial flights across the European network that led to a huge boost in AI and machine-learning applications. Eurocontrol itself developed a number of AI based applications and are working on more. Currently more than thirty AI-based applications are under development in different frameworks notably in the Network of Innovation Labs and SESAR.

The data used in the thesis is used in several up-to-date projects and researches as part of big data. Of course, most important and complex R&D are done by big companies, connected with aviation directly, like SESAR (Single European Sky ATM Research), Thales (Aerospace company) and ECTL MUAC (Eurocontrol's Maastricht Upper Area Control center).

Historic commercial flights data used in the thesis contains following plan and fact information:

- unique numeric identifier for each flight
- number of the point crossed by a flight
- time (UTC) at which the point was crossed
- latitude at which the point was crossed
- longitude at which the point was crossed
- altitude in flight levels at which the point was crossed

Visual comparison of fact and plan data of one random flight is presented in figure 5.

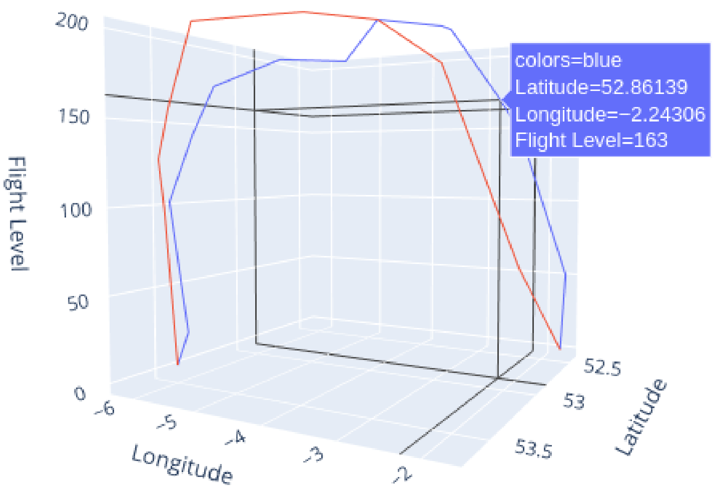

Figure 5: Plan(blue) vs. fact(red) flight trajectories

Data preprocessing include cleaning (duplicates, missing values, outliers, etc.), rounding time variables up to 1 min, expanding information minute-wise, joining fact and plan data frames, computation of extra features.

Additionally, several features were created based on given ones:

- take off country

- take off airport
- landing country
- landing airport
- time variables: day of a month, day of a week, hour, month of a year, week of a year, weekday or weekend
- remaining time until landing in minutes - target column
- flight delay in minutes - target column

Total number of observations (rows) after preprocessing is 1.2 million, number of flight - more than 7 thousand. Variable types in the final data set: numerical and categorical.

After encoding and scaling were implemented. Label encoder was applied to columns describing take off and landing country/airport. It encodes labels of provided column with value between 0 and number of classes-1. Applied min max scaler transforms features by scaling each feature to a given range (range=(0, 1) was used). This estimator scales and translates each feature individually such that it is in the given range on the training set.

4.2 Basic statistical analysis

Histograms of delay time and time remaining before landing are visualized on figure 6.

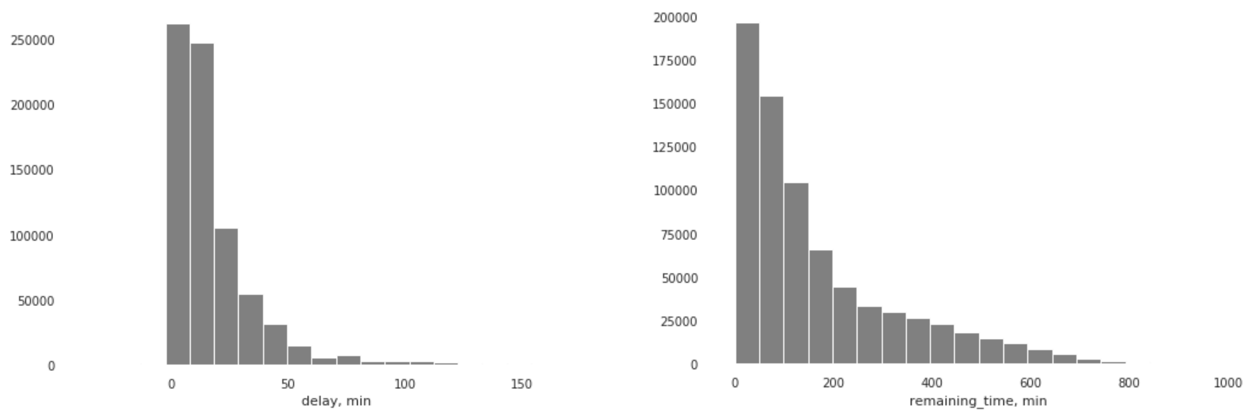

Figure 6: Left: delay time histogram, Right: remaining time histogram

Analysis of correlation authorizes to assert that flight delay and remaining time until landing as target columns have no high dependence with other features. Basic statistics of the target columns is provided in the table 1.

From the flight delay time histogram and it's quantile statistics it is clear that target column has "half-bell" form, highly skewed to the right ($\approx +3$). The 5th percentile value confirms this. Median flight delay is 10 min, while mean value - more than 16 min with relatively high standard deviation (19.5). Due to the fact that most values are in range from 0 to 20, while full range is 0-175, it justifies by really large value of kurtosis (≈ 12).

It's clear from histograms, that remaining time until landing distribution has something in common with the flight delay, specifically, it also has has "half-bell" form and minimal value 0. The difference will be that values of skewness and kurtosis statistics are lying in normal ranges (1.410 and 1.447 respectively). Median remaining time until landing is 107 min, while mean value - about 168 min with relatively high standard deviation (165.358). The difference, is not so dramatic as for the first described target.

Table 1: Quantile and descriptive statistics of target columns

No.	Quantile statistics name	Delay statistics	Remaining time statistics
1	Minimum	0	0
2	5th prercentile	0	9
3	First quartile (Q1)	4	47
4	Median	10	107
5	Third quartile (Q3)	21	237
6	95th prercentile	52	528
7	Maximum	175	989
8	Range	175	989
9	Interquartile range (IQR)	17	190
No.	Quantile statistics name	Delay statistics	Remaining time statistics
1	Standard deviation	19.587	165.358
2	Coefficient of variation (CV)	1.207	0.985
3	Kurtosis	11.979	1.447
4	Mean	16.228	167.832
5	Median Absolute Deviation (MAD)	7	74
6	Skewness	2.934	1.41
7	Variance	383.636	27343.328

5 Experiments

5.1 Regression model

Initial model to train is a basic linear regression model (Ordinary Least Squares - OLS) implemented via statsmodels - module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration for the Python programming language. It was chosen by purpose, as there are a list of papers which predict flight delay exactly by this method and it's results are the starting point to compare with another models. There are several iterations of fit (for each target and different features) on train sample of preprocessed data (85%/15% train/test split).

Flight delay columns was predicted in the first instance as the main target.

The results of fit contains detailed information about the linear regression. In the summary we interested in the p-value from each coefficient. Looking at coefficients, we have three p-values that are more than 0.05 (chosen significance level), other p-values are very low (although not exactly 0). This means that most features have a strong connection with the target 'delay'. Results provided in the table 2. The results of the model fitted on the relevant, statistically significant variables only, also provided in the table 2. The not used features are x_2 - fact flight level, x_3 - fact latitude, x_{11} - plan latitude. Removing that predictors would slightly reduce the R^2 value (from 0.075 to 0.066 - on 12%), but we might make better predictions.

Table 2: Flight delay OLS models summary

	All	features	fitted	model	Relevant	features	fitted	model
	Coeff	Std Error	t values	P>|t|	Coeff	Std Error	t values	P>|t|
const	0.2506	0.001	220.562	0.000	0.2192	0.001	208.600	0.000
x1 (ectrl id)	-0.0610	0.001	-105.357	0.000	-0.0636	0.001	-111.406	0.000
x2 (flight level fact)	-0.0020	0.002	-1.223	0.221	-	-	-	-
x3 (latitude fact)	-0.0402	0.035	-1.156	0.248	-	-	-	-
x4 (longitude fact)	-1.0816	0.033	-32.515	0.000	-0.0046	0.002	-2.767	0.006
x5 (take off airport)	0.0174	0.000	40.229	0.000	0.0198	0.000	46.045	0.000
x6 (take off country)	0.0082	0.000	20.232	0.000	0.0134	0.000	33.769	0.000
x7 (landing airport)	-0.0312	0.000	-75.412	0.000	-0.0320	0.000	-77.156	0.000
x8 (landing country)	-0.0263	0.000	-60.544	0.000	-0.0226	0.000	-52.116	0.000
x9 (hour)	0.1671	0.001	168.718	0.000	0.1579	0.001	160.378	0.000
x10 (flight level plan)	-0.0079	0.002	-4.620	0.000	-0.0050	0.002	-2.928	0.003
x11 (latitude plan)	0.0632	0.035	1.818	0.069	-	-	-	-
x12 (longitude plan)	1.0280	0.033	30.866	0.000	0.0248	0.001	17.888	0.000

No doubt, linear regression is a fundamental tool that has distinct advantages over other regression algorithms. Due to its simplicity, it's an exceptionally quick algorithm to train, thus typically makes it a good baseline algorithm for common regression scenarios. More importantly, models trained with linear regression are the most interpretable kind of regression models available - meaning it's easier to take action from the results of a linear regression model. However, if the assumptions are not satisfied, the interpretation of the results will not always be valid. This can be very dangerous depending on the application. The following assumptions of linear regression were checked:

- linearity
- normality
- no or little multicollinearity
- no auto-correlation
- homoscedasticity

Linearity assumes that there is a linear relationship between the features (independent variables) and the target (e.g. dependent variable). This also assumes that the predictors are additive. There may not just be a linear relationship among the data. Modeling is about trying to estimate a function that explains a process, and linear regression would not be a fitting estimator (pun intended) if there is no linear relationship. Scatter plots for each feature-target pair showed, that there is no linear relationship of the target with at least one of predictors. But as we work with multivariate model, have more then one predictor, separate relation between each feature and target is not representative.

Normality assumption assumes that the error terms of the model are normally distributed. Linear regressions other than OLS may also assume normality of the predictors, but that is not the case here. This can actually happen if either the predictors or the target are significantly non-normal. A violation of this assumption could cause issues with either shrinking or inflating our confidence intervals.

There are a variety of ways to do detect normality, but we looked at both a histogram and tests: the Kolmogorov-Smirnov test and Anderson test. Both test results showed same output - the residuals do not follow a normal distribution. Let's confirm it with figure 7.

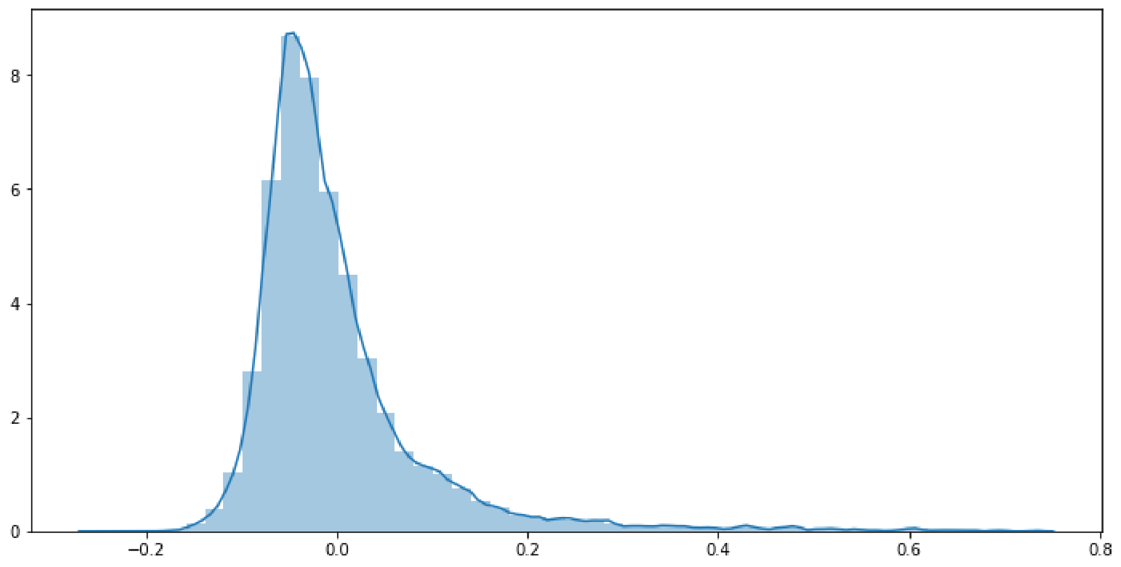

Figure 7: Distribution of the flight delay residuals

Absence of multicollinearity assumes that the predictors used in the regression are not correlated with each other. This won't render our model unusable if violated, but it will cause issues with the interpretability of the model. Multicollinearity causes issues with the interpretation of the coefficients. There are a few ways to detect it. In Chanter 4 Data analysis of correlation was implemented and it asserts that flight delay and remaining time until landing as target columns have no correlation with other features. Additionally, we might examine the variance inflation factor (VIF). According to it, fact and plan flight level features has VIF values up to 15, this is not high enough to make a decision about definite multicollinearity, only possible multicollinearity. All other features have VIF values less than 1.3. So, the isn't quite as egregious as our normality assumption violation, but there is possible multicollinearity for two of the variables in this dataset.

Next assumption assumes no autocorrelation of the error terms. Autocorrelation being present typically indicates that we are missing some information that should be captured by the model. Durbin-Watson test can be used to determine if either positive or negative correlation is present. The received result is 0.013. While the test value can be between 0 and 4, values around 2 indicate no autocorrelation. As a rule of thumb values from 1.5 to 2.5 show that there is no auto-correlation in the data. However, the Durbin-Watson test only analyses linear autocorrelation and only between direct neighbors, which are first order effects. 0.013 result signs of positive autocorrelation, which means that assumption is not satisfied.

Homoscedasticity is the last assumption. Heteroscedasticity occurs when we don't have an even variance across the error terms. We might plot the residuals and see if the variance appears to be uniform. The Goldfeld-Quandt Test can also be used to test for heteroscedasticity. Test p-value $\approx$0, which means that the variance in one subsample is larger than in the other subsample. Looking at the residuals plot 8 we can't see a fully uniform variance across our residuals, so this is potentially problematic.

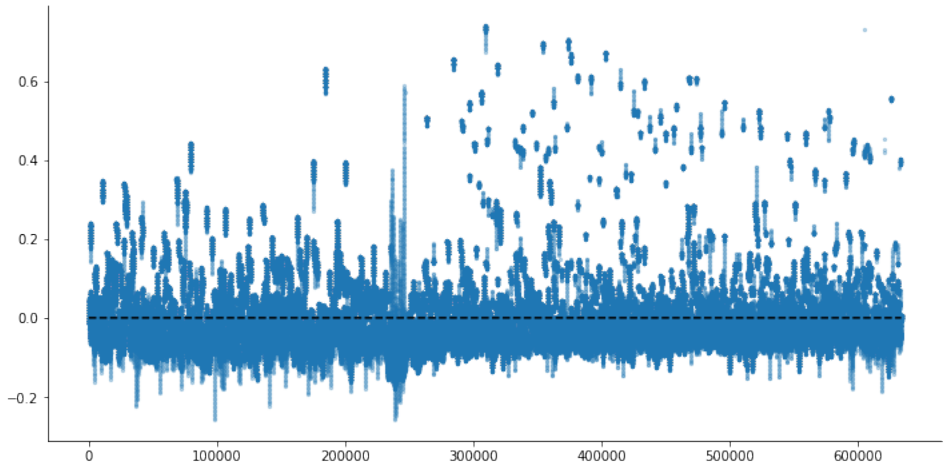

Figure 8: Flight delay residuals

We can clearly see that a linear regression model on the flight data violates a number of assumptions which cause significant problems with the interpretation of the model itself. So we are aware of the flaws in the model.

The same process was applied to prediction of remaining time until landing. That model is assumed to help describing main(flight delay) predictive model results more fully. In order not to repeat the full process of training, we provide here only the final results and conclusions. First fit of OLS model on full features shows that all features are relevant, statistically significant (all p-values around 0). This means that most features have a strong connection with the 'remaining time' column. R^2 is 0.18. But, during analysis of multicollinearity assumption, fact and plan latitude and longitude showed significantly huge values of VIF, and new OLS model, fitted without those features, has higher R^2 (0.136, increase by 24.4%).

Same assumptions were checked. Scatter plots for each feature-target pair showed, that there is no linear relationship of the target with at least one of predictors. But as we work with multivariate model, have more then one predictor, separate relation between each feature and target is not representative.

Kolmogorov-Smirnov and Anderson test results, that check the normality assumption, showed same output - the residuals do not follow a normal distribution. Let's confirm it with figure 9.

Durbin-Watson test (used to determine whether autocorrelation being present) received result is 0.012, that signs of positive autocorrelation, which means that assumption is not satisfied.

Homoscedasticity is the last assumption. In contradistinction to flight delay, remaining time residuals p-value of Goldfeld-Quandt test 0.9 $>$ 0.05. Looking at the residuals plot 10 we can not see if the variance appears to be uniform, but at the same time don't appear to be any obvious problems with that.

According to the received results, we might make a conclusion that OLS regression models (both, for flight delay prediction and remaining time until landing prediction) are not the best choice for this dataset, as not all the required assumptions are required.

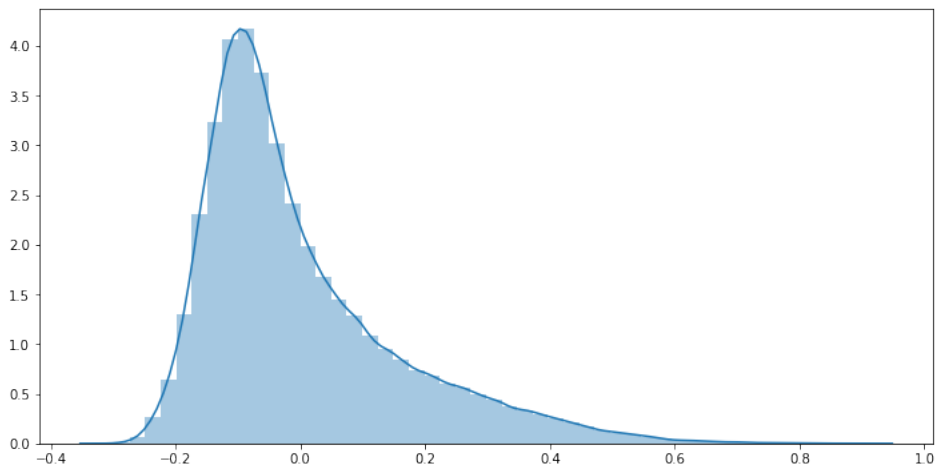
Figure 9: Distribution of the remaining time until landing residuals

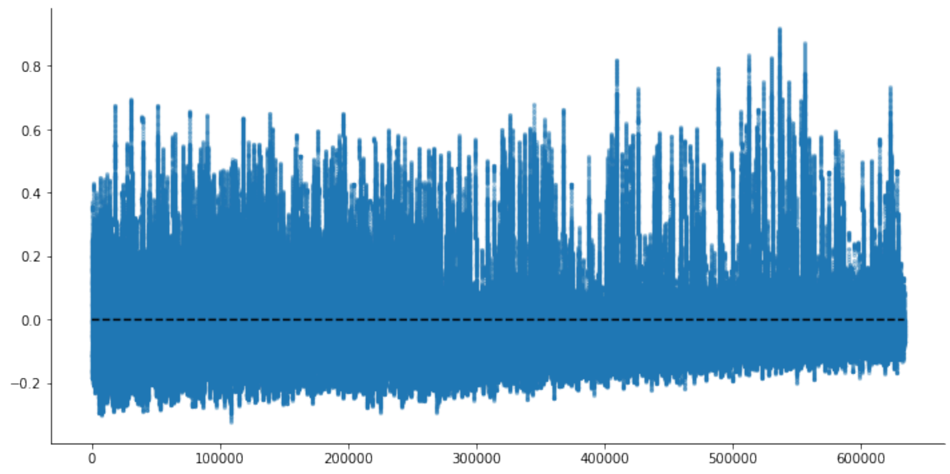
Figure 10: Remaining time until landing residuals

5.2 Deep neural networks

Next model type to train is DNN models implemented via Pytorch - an open source machine learning library for the Python programming language based on the Torch library (Facebook's AI Research lab). There are several iterations of fit on train sample of preprocessed data (85%/15% train/test split). For all iterations several parameters are equal:

- usage of liner layers
- output shape for single-point estimate models - 1 (flight delay value)
- output shape for probability-distribution parameters estimate models - 2 (parameters of distribution). μ and σ in case of Normal distribution; k and λ in case of Weibull distribution
- training criterion for single-point estimate models - MSE loss
- training criterion for probability-distribution parameters estimate models - negative log-likelihood
- optimizer - Adam
- number of epochs for training - 50

It should be emphasized that during models training and optimization for each target (flight delay and remaining time until landing) two types of DNN models were applied: single-point estimate model, let's call it DNN regression in short, and probability-distribution parameters estimate model, let's call it probabilistic DNN. DNN regression predicts target value itself, probabilistic DNN predicts distribution parameters of the target, based on which, we might calculate different statistics (for example, mean flight delay and it's standard deviation).

The main technical distinctions are the output shape of neural net (1 for DNN regression, 2 for probabilistic DNN) and training loss function (MSE for DNN regression, negative log-likelihood for probabilistic DNN). During optimization of DNN architecture, each applied potential improvement method, was used twice, for DNN regression and probabilistic DNN for each target prediction.

Implemented architecture structures using five liner layers, each with the following output:

- 64/32/16/8/1(2)[3]
- 64/64/32/32/1(2)
- 128/64/32/16/1(2)
- 512/256/138/32/1(2)

To each of this architecture following techniques were added:

- bath normalization
- ReLU activation function
- 20% dropout
- L2 regularization
- fluctuating learning rate

Those four techniques were applied to each of four architecture structures not only separately, but in combination too, like: BatchNorm1d+ReLU, BatchNorm1d+ReLU+Dropout, ReLU+Dropout, BatchNorm1d+L2, ReLU+L2.

Layers of type 'Liner' apply a linear transformation to the incoming data ($y = xA^T + b$), based on size of each input and output sample. Also the layer learns an additive bias.

Adam optimizer is a replacement optimization algorithm for stochastic gradient descent for training DNNs to update network weights iterative based in training data.

BatchNorm1d is algorithmic method which makes the training faster and more stable. It consists of normalizing activation vectors from hidden layers using the first and the second statistical moments (mean and variance) of the current batch.

ReLU is a non-linear activation function applies the rectified linear unit function element-wise ($RELU(x) = (x)^+ = max(0, x)$)

Dropout is a regularization technique for reducing overfitting. It randomly omit units (both hidden and visible) during the training process of a neural network.

L2 regularization in Pytorch is a part of an training optimizer (Adam in current case). It adds a small penalty, the L2 norm of the weights (all the weights of the model), to the loss function.

Fluctuating learning rate was applied through CosineAnnealingLR, it sets the learning rate of each parameter group using a cosine annealing schedule. For each epoch it lower the learning rate to its minimum in each epoch and then restart from the base.

Additionally, feature generation was implemented. It is the process of creating new features from one or multiple existing features. This process adds new information to be accessible during the model construction and therefore hopefully result in a more accurate model. Using basic arithmetic operations, we

[3]here and below, 1 of for DNN regression, value 2 in brackets - for probabilistic DNN

estimated models not only of provided and preprocessed data, but on additional features too (144 in total). Unfortunately, this approach did not show high performance, only it's deterioration.

Counting up, total number of DNN models trained is 148. Based on the comparison of test sample MSE the best models for flight delay prediction are:

- 64/64/32/32/1 architecture + batch normalization after each layer - for DNN regression
- 64/32/16/8/2 architecture + L2 regularization - for probabilistic DNN (for each Normal and Weibull distributions estimation)

And the best models for remaining time until landing prediction are:

- 64/32/16/8/1 architecture + batch normalization after each layer - for DNN regression
- 128/64/32/16/2 architecture + batch normalization - for probabilistic DNN (for each Normal and Weibull distributions estimation)

5.3 Transformer neural networks

Next model type to train is Transformer neural network (TNN) models implemented via Keras - an open-source software library that provides a Python interface for artificial neural networks (acts as an interface for the TensorFlow library). There are several iterations of fit on train sample of preprocessed data (85%/15% train/test split). As in previous sub-chapter, for all iterations several parameters are equal:

- transformer architecture
- output shape for single-point estimate models - 1 (flight delay value)
- output shape for probability-distribution parameters estimate models - 2 (parameters of distribution). μ and σ in case of Normal distribution; k and λ in case of Weibull distribution
- training criterion for single-point estimate models - MSE loss
- training criterion for probability-distribution parameters estimate models - negative log-likelihood
- number of epochs for training - 50

Just as it was done in DNN models, two types of TNN models were applied: single-point estimate model (TNN regression) and probability-distribution parameters estimate model (probabilistic TNN). And at the same way TNN regression predicts target value itself, probabilistic TNN predicts distribution parameters of the target, based on which, we might calculate different statistics.

The same differences are present here too: output shape of neural net (1 for TNN regression, 2 for probabilistic TNN) and training loss function (MSE for TNN regression, negative log-likelihood for probabilistic TNN). During optimization of TNN architecture, each applied potential improvement method, was used twice, for DNN regression and probabilistic DNN for each target prediction.

In contradistinction to DNN models, Transformer models' architecture is provided by the paper [24] and fixed. Basically, that means, that we are not able to implement a wide range of TNN models by applying different methods and techniques.

But what we might do is to add following techniques to the fixed architecture:

- L2 regularization
- fluctuating learning rate
- 20% dropout

To implement L2 regularizaion in keras, we used so called AdamW optimizer, that actually implements the Adam algorithm but with weight decay.

Fluctuating learning rate was applied at the same way, through CosineDecay that uses a cosine decay schedule to set the learning rate. Basically, this is analog of pytorch CosineAnnealingLR in keras.

Here also, we estimated models not only of provided and preprocessed data, but on additional features too (144 in total). And at the same way, this approach did not succeed.

Counting up, total number of TNN models trained is 13. Based on the comparison of test sample MSE the best models for flight delay prediction are:

- Transformer + L2 regularization - for TNN regression
- Transformer + L2 regularization - for probabilistic TNN (for each Normal and Weibull distributions estimation)

And the best models for remaining time until landing prediction are:

- Transformer without tuning
- Transformer without tuning - for probabilistic TNN (for each Normal and Weibull distributions estimation)

6 Discussion

It is essential to compare models exactly on test dataset, which was not seen by any model before. For comparison we chose MSE metric, which measures the average of the squares of the errors and gives information about the quality of a model. That metric was taken as R^2 metric is about the same, fluctuation up to 0.05. So we decided on one metric to have consistent comparison. In the table 3 metrics of each model type (best in its segment) are presented.

Table 3: Flight delay test metric comparison

No.	Model	Flight delay test MSE	Remaining time test MSE
1.	OLS	0.00730	0.01721
2.	DNN regression	0.00764	0.02292
3.	DNN probabilistic (Normal dist.)	0.00719	0.00184
4.	DNN probabilistic (Weibull dist.)	0.00711	0.01722
5.	TNN regression	0.01118	0.02838
6.	TNN probabilistic (Normal dist.)	0.00794	0.01935
7.	TNN probabilistic (Weibull dist.)	0.00796	0.01928

From the flight delay predictive models' test metrics, we make a conclusion that results are quite similar, but probabilistic DNN, which predict delay Weibull distribution parameter has the lowest MSE. OLS models was created as a basic one and the best models overperforms it only by $\approx 2.7\%$.

From the remaining time predictive models' test metrics, we make a conclusion that results are quite similar, but it's clear that TNN models preform worse than any others. Probabilistic DNN, which predict delay Weibull distribution parameter, and OLS basic model, has same score. From which we conclude that we did not succeed in creating higher performance model, but but nevertheless we believe that it is more efficient to rely on probabilistic DNN model prediction, as probability-distribution parameters can provide a potential user with the information about confidence interval for a given confidence threshold. It is more reliable and user-friendly to inform that the delay, for example, is no more than 25 minutes than to give an answer, that delay is 15 min precisely.

According to received results, it should be noted that even the best models do not perform well. Flight delay is an big issue, which have not yet been well predicted. We came to the conclusion that first of all proper data should be collected to receive good performance metrics, for example, weather different conditions (temperature, humidity, wind speed, etc.), explanation of delay, its' reasons and other. Of course, there a huge range of model type and machine learning approaches which might be applied to the flight delay problem, but any of them will not perform well without sufficient information base.

Conclusion

Motivated by delay negative impacts and its predicted deterioration [23], we propose a Deep neural network models for flight delay and remaining time until landing prediction. In particular, we propose the models that predict a probability distribution, that additionally can provide user with information about confidence interval of estimated delay.

The goal of current research is reached, different neural networks and methods were applied to estimate flight delay which were not used before for flight delay prediction. Additionally to that, remaining time before landing was predicted to expand the information provided about delay.

During project implementation five model types were created to apply for delay prediction: basic liner regression (OLS), DNN regression and DNN probabilistic (estimate Normal and Weibull distributions parameters), TNN regression and TNN probabilistic (estimate Normal and Weibull distributions parameters).

The results obtained show that regression models as predicting mechanism is not the best choice. In contrast to this, models estimating probability-distribution parameters perform better. Moreover, probability-distribution parameters can provide user with information about confidence interval for a given confidence threshold.

Also, in comparison with basic liner regression and Transformer models with novel attention mechanism, Deep neural networks performs better. Probabilistic models that estimated Weibull distribution parameters are the most productive for prediction delay.

It should be noted that even the best models do not perform well. Flight delay is an big issue, which have not yet been well predicted. We came to the conclusion that first of all proper data should be collected to receive good performance metrics.

For further research we plan to expand data and train models on hundreds of millions of rows. Other distributions will be applied. Moreover, prediction of trajectory will be investigated on the same data additionally.

References

[1] K. Aggarwal, O. Atan, A. K. Farahat, C. Zhang, K. Ristovski, and C. Gupta. Two Birds with One Network: Unifying Failure Event Prediction and Time-to-failure Modeling. pages 1308–1317, 2018.

[2] D. G. Altman and J. M. Bland. Time to event (survival) data. *BMJ*, page 468–469, 1998.

[3] S. Ayhan, P. Costas, and H. Samet. Predicting Estimated Time of Arrival for Commercial Flights. pages 33–42, 2018.

[4] P. Balakrishna, R. Ganesan, and L. Sherry. Accuracy of reinforcement learning algorithms for predicting aircraft taxi-out times: A case-study of tampa bay departures. *Transportation Research Part C: Emerging Technologies*, 18(6):950–962, 2010. Special issue on Transportation Simulation Advances in Air Transportation Research.

[5] P. Balakrishna, R. Ganesan, L. Sherry, and B. S. Levy. Estimating taxi-out times with a reinforcement learning algorithm. *2008 IEEE/AIAA 27th Digital Avionics Systems Conference*, pages 3.D.3–1–3.D.3–12, 2008.

[6] L. Carvalho, A. Sternberg, L. Maia Goncalves, A. Beatriz Cruz, J. A. Soares, D. Brandao, D. Carvalho, and E. Ogasawara. On the relevance of data science for flight delay research: a systematic review. *TRANSPORT REVIEWS*, pages 499–528, 2021.

[7] T. Chen, B. Keng, and J. Moreno. Multivariate Arrival Times with Recurrent Neural Networks for Personalized Demand Forecasting. pages 810–819, 2018.

[8] Z. Chen, D. Guo, and Y. Lin. A deep gaussian process-based flight trajectory prediction approach and its application on conflict detection. *Algorithms*, 13(11), 2020.

[9] A. Cook and G. Tanner. European airline delay cost reference values, 2015.

[10] European Aviation Atrificial Intelligence High Level Group. The FLY AI report, 2020.

[11] E. C. Fernández, J. M. Cordero, G. A. Vouros, N. Pelekis, T. Kravaris, H. V. Georgiou, G. Fuchs, E. Casado, P. Costas, and S. Ayhan. Dart : A machine-learning approach to trajectory prediction and demand-capacity balancing. *Seventh SESAR Innovation Days*, 2017.

[12] E. O. for the Safety of Air Navigation (EUROCONTROL). Predicting flight routes with a deep neural network in the operational air traffic flow and capacity management system, 2018.

[13] E. O. for the Safety of Air Navigation (EUROCONTROL). All-causes delay to air transport in europe for q3 2021, 2021.

[14] X. Guan, L. S. Renli Lv, and Y. Liu. A study of 4d trajectory prediction based on machine deep learning. *2016 12th World Congress on Intelligent Control and Automation (WCICA)*, pages 24–27, 2016.

[15] J. Huh, E. Martinsson, A. Kim, and J.-W. Ha. Modeling musical onset probabilities via neural distribution learning, 2020.

[16] J.-Y. Kim, Y. S. Lee, J. Yu, Y. Park, S. K. Lee, M. Lee, J. E. Lee, S. W. Kim, S. J. Nam, Y. H. Park, J. S. Ahn, M. Kang, and Y.-H. Im. Deep learning-based prediction model for breast cancer recurrence using adjuvant breast cancer cohort in tertiary cancer center registry. *Frontiers in oncology*, page 596364, 2021.

[17] Y. Liu and M. Hansen. Predicting aircraft trajectories: A deep generative convolutional recurrent neural networks approach, 2018.

[18] C. B. M. Tielrooij, M. Van Paassen, and M. Mulder. Predicting arrival time uncertainty from actual flight information. 2015.

[19] E. Martinsson. WTTE-RNN : Weibull Time To Event Recurrent Neural Network. Master's thesis, Chalmers University Of Technology, 2016.

[20] L. Neumann, A. Zisserman, and A. Vedaldi. Future Event Prediction: If and When. pages 2935–2943, 2019.

[21] A. S. Palau, M. H. Dhada, K. Bakliwal, and A. K. Parlikad. An Industrial Multi Agent System for real-time distributed collaborative prognostics. *ENGINEERING APPLICATIONS OF ARTIFICIAL INTELLIGENCE*, pages 590–606, 2019.

[22] Z. Shi, M. Xu, Q. Pan, B. Yan, and H. Zhang. Lstm-based flight trajectory prediction. pages 1–8, 2018.

[23] S. J. Undertaking. European atm master plan, 2020.

[24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017.

[25] Z. Wang, A. Liang, and D. Delahaye. Short-term 4d trajectory prediction using machine learning methods. *Seventh SESAR Innovation Days*, 2017.

[26] Z.-J. Wu, S. Tian, and L. Ma. A 4d trajectory prediction model based on the bp neural network. *Journal of Intelligent Systems*, 29(1):1545–1557, 2020.

[27] M. F. Yazdi, S. R. Kamel, S. J. M. Chabok, and M. Kheirabadi. Flight delay prediction based on deep learning and levenberg-marquart algorithm. *Journal of Big Data*, 2020.

[28] M. Zanin, D. Perez, K. Chatterjee, D. S. Kolovos, R. F. Paige, A. Horst, and B. Rumpe. On demand data analysis and filtering for inaccurate flight trajectories, 2014.

[29] X. Zhang and S. Mahadevan. Bayesian neural networks for flight trajectory prediction and safety assessment. *Decision Support Systems*, 131:113246, 2020.

[30] P. Zheng, S. Yuan, and X. Wu. SAFE: A Neural Survival Analysis Model for Fraud Early Detection. pages 1278–1285, 2019.

Python code for Chapter 4 Data

```
[1]: import pandas as pd
     import numpy as np
     import math
     import random
     import datetime
     from datetime import timedelta
```

1 CLEAN DATA

```
[ ]: fact = pd.read_csv("Flight_Points_Actual_20180601_20180630.csv")
```

```
[ ]: fact=fact.drop(columns=['Unnamed: 0'])
```

```
[ ]: fact.shape
```

```
[ ]: # by index
     to_drop1 = fact[(fact['Sequence Number']==1) & (fact['Flight Level']==0)].index
     to_drop2 = fact[(fact['Sequence Number']==2) & (fact['Flight Level']==0)].index
```

```
[ ]: fact = fact.drop(axis=0, index=to_drop1)
     fact = fact.drop(axis=0, index=to_drop2)
```

```
[ ]: fact=fact.reset_index(drop=True)
```

```
[ ]: fact = fact[:299999]
```

```
[ ]: fact=fact.reset_index(drop=True)
```

```
[ ]: fact.shape, plan.shape
```

2 REMOVE SMALL FLIGHTS

```
[ ]: temp_df = pd.DataFrame(fact.groupby('ECTRL ID')['Sequence Number'].max().
      ↪sort_values())
```

```
[ ]: to_drop_f = temp_df[temp_df['Sequence Number']<10].index
     to_drop5 = fact[fact['ECTRL ID'].isin(to_drop_f.tolist())].index
```

```
[ ]: fact = fact.drop(axis=0, index=to_drop3)
```

```
[ ]: fact=fact.reset_index(drop=True)
```

```
[ ]: fact.shape
```

3 MAKE 1 MIN ROUND

```
[ ]: fact['Time Over'] = pd.to_datetime(fact['Time Over'], format = "%Y-%m-%d %H:%M:
      ↪%S")
```

```
[ ]: fact['Time Over'] = fact['Time Over'].dt.round('1min')
```

4 GEOPY

```
[ ]: df = pd.read_csv("fact_clean_250th.csv")
     df = df.drop(columns=['Unnamed: 0'])
```

```
[ ]: flight_level_0_indexes = df[df['Flight Level']==0].index
```

```
[ ]: print("NUMBER OF FLIGHTS: ", int(len(flight_level_0_indexes)/2))
```

```
[ ]: # initialize Nominatim API
     from geopy.geocoders import Nominatim
     geolocator = Nominatim(user_agent="geoapiExercises")
```

```
[ ]: start_i = 0
     start_j = 0

     for i,j in enumerate(flight_level_0_indexes):
         print("start_i", start_i)

         location_start = geolocator.reverse(str(df.loc[start_j,␣
      ↪'Latitude'])+","+str(df.loc[start_j, 'Longitude']),language="en")
         if "aeroway" in location_start.raw['address']:
```

```
        df.loc[start_j:flight_level_0_indexes[start_i+1],'airport_start'] =␣
↪location_start.raw['address']['aeroway']
    else:
        res0 = list(location_start.raw['address'].keys())[0]
        res1 = list(location_start.raw['address'].keys())[1]
        df.loc[start_j:flight_level_0_indexes[start_i+1],'airport_start'] =␣
↪location_start.raw['address'][res0]+" "+location_start.raw['address'][res1]
    df.loc[start_j:flight_level_0_indexes[start_i+1],'country_start'] =␣
↪location_start.raw['address']['country']


    location_end = geolocator.reverse(str(df.
↪loc[flight_level_0_indexes[start_i+1], 'Latitude'])+","+str(df.
↪loc[flight_level_0_indexes[start_i+1], 'Longitude']),language="en")
    if "aeroway" in location_end.raw['address']:
        df.loc[start_j:flight_level_0_indexes[start_i+1],'airport_end'] =␣
↪location_end.raw['address']['aeroway']
    else:
        res0 = list(location_end.raw['address'].keys())[0]
        res1 = list(location_end.raw['address'].keys())[1]
        df.loc[start_j:flight_level_0_indexes[start_i+1],'airport_end'] =␣
↪location_end.raw['address'][res0]+" "+location_end.raw['address'][res1]
    if 'country' in location_end.raw['address']:
        df.loc[start_j:flight_level_0_indexes[start_i+1],'country_end'] =␣
↪location_end.raw['address']['country']
    else:
        df.loc[start_j:flight_level_0_indexes[start_i+1],'country_end'] =␣
↪'Unknown'
    start_i += 2
```

5 TIME VARIABLES

[ ]:
```
df['Time Over'] = pd.to_datetime(df['Time Over'])
```

[ ]:
```
for index, row in df.iterrows():
    df.loc[index, 'Month of a Year'] = df.loc[index, 'Time Over'].month
    df.loc[index, 'Week of a Year'] = df.loc[index, 'Time Over'].week
    df.loc[index, 'Day of a Month'] = df.loc[index, 'Time Over'].day
    df.loc[index, 'Day of a Week'] = df.loc[index, 'Time Over'].isoweekday()
    df.loc[index, 'Hour'] = df.loc[index, 'Time Over'].hour
    if df.loc[index, 'Day of a Week'] < 6:
        df.loc[index, 'Weekday or Weekend'] = 0
    else:
        df.loc[index, 'Weekday or Weekend'] = 1
```

6 SPLIT MINUTEWISE

```
[ ]: flight_level_0_indexes = df[df['Flight Level']==0].index
     print("NUMBER OF FLIGHTS: ", int(len(flight_level_0_indexes)/2))
```

```
[ ]: new_df = pd.DataFrame(columns=['ECTRL ID', 'Time Over',
                                    'Flight Level', 'Latitude', 'Longitude',
                                    'airport_start', 'country_start',
                                    'airport_end', 'country_end'])


     start_i = 0
     start_j = 0

     for i,j in enumerate(flight_level_0_indexes):

         print("START OF j", start_j)
         print("from ", start_j, "to", flight_level_0_indexes[start_i+1])
         full_flight_steps = 0

         for index, row in df[start_j:flight_level_0_indexes[start_i+1]].iterrows():

             print("subcycle from ", index, "to", index+1)
             number_of_steps = int((df.loc[index+1, "Time Over"] - df.loc[index,␣
      ↪"Time Over"]).total_seconds()/60.0)
             print("number_of_steps", number_of_steps)
             if number_of_steps != 0:
                 one_step_time_range = [df.loc[index, "Time Over"]+datetime.
      ↪timedelta(minutes=1)*i for i in range(number_of_steps+1)]
                 flight_level_step = (df.loc[index+1, "Flight Level"] - df.
      ↪loc[index, "Flight Level"])/number_of_steps
                 flight_level = [df.loc[index, "Flight Level"] +␣
      ↪flight_level_step*step for step in range(number_of_steps+1)]
                 latitude_step = (df.loc[index+1, "Latitude"] - df.loc[index,␣
      ↪"Latitude"])/number_of_steps
                 latitude = [df.loc[index, "Latitude"] + latitude_step*step for step␣
      ↪in range(1, number_of_steps+2)]
                 longitude_step = (df.loc[index+1, "Longitude"] - df.loc[index,␣
      ↪"Longitude"])/number_of_steps
                 longitude = [df.loc[index, "Longitude"] + longitude_step*step for␣
      ↪step in range(1, number_of_steps+2)]

                 print("len: level, lat, lon, time", len(flight_level),␣
      ↪len(latitude),
                                                     len(longitude),␣
      ↪len(one_step_time_range))
```

```
            new_df=new_df.append(pd.DataFrame({'ECTRL ID':[df.loc[start_j,␣
↪'ECTRL ID']]*(number_of_steps+1),
                                                'Time Over':one_step_time_range,
                                                'Flight Level':flight_level,
                                                'Latitude':latitude,
                                                'Longitude':longitude,

                                                'airport_start':[df.
↪loc[start_j, "airport_start"]]*(number_of_steps+1),
                                                'country_start':[df.
↪loc[start_j, "country_start"]]*(number_of_steps+1),
                                                'airport_end':[df.loc[start_j,␣
↪"airport_end"]]*(number_of_steps+1),
                                                'country_end':[df.loc[start_j,␣
↪"country_end"]]*(number_of_steps+1)}))

            new_df = new_df[:-1]
            full_flight_steps+=number_of_steps

    print("full_flight_steps", full_flight_steps)

    last_row = df.loc[flight_level_0_indexes[start_i+1]]
    new_df = new_df.append(last_row)
    print("END OF j", start_j)
    print("__________")
    start_i += 2
    start_j = flight_level_0_indexes[start_i]
```

```
[ ]: for index, row in new_df.iterrows():
         new_df.loc[index, 'Month of a Year'] = new_df.loc[index, 'Time Over'].month
         new_df.loc[index, 'Week of a Year'] = new_df.loc[index, 'Time Over'].week
         new_df.loc[index, 'Day of a Month'] = new_df.loc[index, 'Time Over'].day
         new_df.loc[index, 'Day of a Week'] = new_df.loc[index, 'Time Over'].
     ↪isoweekday()
         new_df.loc[index, 'Hour'] = new_df.loc[index, 'Time Over'].hour
         if new_df.loc[index, 'Day of a Week'] < 6:
             new_df.loc[index, 'Weekday or Weekend'] = 0
         else:
             new_df.loc[index, 'Weekday or Weekend'] = 1
```

7 TRIP TIME

[ ]:
```
flight_level_0_indexes = new_df[new_df['Flight Level']==0].index
print("NUMBER OF FLIGHTS: ", int(len(flight_level_0_indexes)/2))
```

[ ]:
```
start_i = 0
start_j = 0

for i,j in enumerate(flight_level_0_indexes):
    shape = flight_level_0_indexes[start_i+1]-start_j
    new_df.loc[start_j:flight_level_0_indexes[start_i+1],'trip_time'] =␣
 ↪list(range(shape+1))
    start_i += 2
    start_j = flight_level_0_indexes[start_i]
```

[ ]:
```
new_df.to_csv("fact_clean_250th_minwise.csv")
```

8 SHORT PLAN

[ ]:
```
fact = new_df.copy()
```

[ ]:
```
plan = pd.read_csv("Flight_Points_Filed_20180601_20180630.csv")
to_drop3 = plan[(plan['Sequence Number']==1) & (plan['Flight Level']==0)].index
to_drop4 = plan[(plan['Sequence Number']==2) & (plan['Flight Level']==0)].index
```

[ ]:
```
plan = plan.drop(axis=0, index=to_drop3)
plan = plan.drop(axis=0, index=to_drop4)
```

[ ]:
```
plan=plan.reset_index(drop=True)
```

[ ]:
```
diff_id =list(set(plan[:256050]['ECTRL ID'])-set(fact['ECTRL ID'].unique()))
```

[ ]:
```
to_drop5 = plan[plan['ECTRL ID'].isin(diff_id)].index
```

[ ]:
```
plan = plan[:256050]
plan = plan.drop(axis=0, index=to_drop5)
plan=plan.reset_index(drop=True)
```

[ ]:
```
set(plan['ECTRL ID'])-set(fact['ECTRL ID'])
```

[ ]:
```
fact['ECTRL ID'].nunique(), plan['ECTRL ID'].nunique()
```

9 MINWISE FOR PLAN

```
[ ]: plan['Time Over'] = pd.to_datetime(plan['Time Over'], format = "%d-%m-%Y %H:%M:
      ↪%S")
```

```
[ ]: plan['Time Over'] = plan['Time Over'].dt.round('1min')
```

```
[ ]: df = plan.copy()
```

```
[ ]: flight_level_0_indexes = df[df['Flight Level']==0].index
     print("NUMBER OF FLIGHTS: ", int(len(flight_level_0_indexes)/2))
```

```
[ ]: new_df = pd.DataFrame(columns=['ECTRL ID', 'Time Over',
                                    'Flight Level', 'Latitude', 'Longitude'])

     start_i = 0
     start_j = 0

     for i,j in enumerate(flight_level_0_indexes):

         print("START OF j", start_j)
         print("from ", start_j, "to", flight_level_0_indexes[start_i+1])
         full_flight_steps = 0

         for index, row in df[start_j:flight_level_0_indexes[start_i+1]].iterrows():

             print("subcycle from ", index, "to", index+1)
             number_of_steps = int((df.loc[index+1, "Time Over"] - df.loc[index,␣
      ↪"Time Over"]).total_seconds()/60.0)
             print("number_of_steps", number_of_steps)
             if number_of_steps != 0:
                 one_step_time_range = [df.loc[index, "Time Over"]+datetime.
      ↪timedelta(minutes=1)*i for i in range(number_of_steps+1)]
                 flight_level_step = (df.loc[index+1, "Flight Level"] - df.
      ↪loc[index, "Flight Level"])/number_of_steps
                 flight_level = [df.loc[index, "Flight Level"] +␣
      ↪flight_level_step*step for step in range(number_of_steps+1)]
                 latitude_step = (df.loc[index+1, "Latitude"] - df.loc[index,␣
      ↪"Latitude"])/number_of_steps
                 latitude = [df.loc[index, "Latitude"] + latitude_step*step for step␣
      ↪in range(1, number_of_steps+2)]
                 longitude_step = (df.loc[index+1, "Longitude"] - df.loc[index,␣
      ↪"Longitude"])/number_of_steps
                 longitude = [df.loc[index, "Longitude"] + longitude_step*step for␣
      ↪step in range(1, number_of_steps+2)]
```

```
            print("len: level, lat, lon, time", len(flight_level),␣
↪len(latitude),
                                                len(longitude),␣
↪len(one_step_time_range))

            new_df=new_df.append(pd.DataFrame({'ECTRL ID':[df.loc[start_j,␣
↪'ECTRL ID']]*(number_of_steps+1),
                                                'Time Over':one_step_time_range,
                                                'Flight Level':flight_level,
                                                'Latitude':latitude,
                                                'Longitude':longitude}))

            new_df = new_df[:-1]
            full_flight_steps+=number_of_steps

    print("full_flight_steps", full_flight_steps)

    last_row = df.loc[flight_level_0_indexes[start_i+1]]
    new_df = new_df.append(last_row)
    print("END OF j", start_j)
    print("__________")
    start_i += 2
    start_j = flight_level_0_indexes[start_i]
```

10 JOIN FACT AND PLAN

[ ]:
```
fact = pd.read_csv("fact_clean_250th_minwise.csv")
plan = pd.read_csv("plan_clean_250th_minwise.csv")
```

[ ]:
```
fact=fact.drop(columns=['Unnamed: 0', 'Sequence Number'])
plan=plan.drop(columns=['Unnamed: 0', 'Sequence Number'])
```

[ ]:
```
fact.shape, plan.shape
```

[ ]:
```
set(plan['ECTRL ID'])-set(fact['ECTRL ID'])
```

[ ]:
```
new_cols = plan.columns+"_plan"
```

[ ]:
```
plan.columns = new_cols
```

[ ]:
```
flight_level_0_fact = fact[fact['Flight Level']==0].index
flight_level_0_plan = plan[plan['Flight Level_plan']==0].index
print("NUMBER OF FLIGHTS fact: ", int(len(flight_level_0_fact)/2))
print("NUMBER OF FLIGHTS plan: ", int(len(flight_level_0_plan)/2))
```

```
[ ]: full_df = pd.DataFrame(columns=list(fact.columns)+list(plan.columns))

     start_i = 0
     start_j_f = 0
     start_j_p = 0

     for i,j in enumerate(flight_level_0_fact):
         print("start_i", start_i)

         one_id_df = pd.concat(objs=(fact[start_j_f:flight_level_0_fact[start_i+2]].
      ↪reset_index(drop=True),
                                     plan[start_j_p:flight_level_0_plan[start_i+2]].
      ↪reset_index(drop=True)),
                                axis=1, ignore_index=True)
         one_id_df.columns = list(fact.columns)+list(plan.columns)
         full_df=full_df.append(one_id_df)

         start_i += 2
         start_j_f = flight_level_0_fact[start_i]
         start_j_p = flight_level_0_plan[start_i]
```

```
[ ]: full_df = full_df.drop(columns=['ECTRL ID_plan'])
```

```
[ ]: full_df=full_df.reset_index(drop=True)
```

11 TARGET COLUMN

```
[ ]: flight_level_0 = full_df[full_df['Flight Level']==0].index
```

```
[ ]: start_i = 0
     start_j = 0

     for i,j in enumerate(flight_level_0):
         print("from", start_j, "to", flight_level_0[start_i+1])
         for index, row in full_df[start_j:flight_level_0[start_i+1]+1].iterrows():
             print("current idx", index)

             if type(full_df.loc[index, 'Time Over_plan'])==pd._libs.tslibs.
      ↪timestamps.Timestamp:

                 full_df.loc[index, 'Time Over'] = pd.to_datetime(full_df.loc[index,␣
      ↪'Time Over'], format = "%Y-%m-%d %H:%M:%S")
                 full_df.loc[index, 'Time Over_plan'] = pd.to_datetime(full_df.
      ↪loc[index, 'Time Over_plan'], format = "%Y-%m-%d %H:%M:%S")
```

```
            full_df.loc[index, "delay"] = full_df.loc[index, 'Time Over'] -␣
↪full_df.loc[index, 'Time Over_plan']

            if full_df.loc[index,"delay"].days < 0:
                full_df.loc[index,"delay"] = -1*int(abs(full_df.
↪loc[index,"delay"]).seconds/60)
            else:
                full_df.loc[index,"delay"] = int(abs(full_df.
↪loc[index,"delay"]).seconds/60)

        elif type(full_df.loc[index, 'Time Over_plan'])==pd._libs.tslibs.
↪nattype.NaTType or type(full_df.loc[index, 'Time Over_plan'])==float:
            previous = full_df.loc[index-1, "delay"]

            if previous >= 0:
                full_df.loc[index, "delay"] = full_df.loc[index-1, "delay"]+1
            else:
                full_df.loc[index, "delay"] = full_df.loc[index-1, "delay"]-1



        elif type(full_df.loc[index, 'Time Over_plan'])==str:
            full_df.loc[index, 'Time Over'] = pd.to_datetime(full_df.loc[index,␣
↪'Time Over'], format = "%Y-%m-%d %H:%M:%S")
            full_df.loc[index, 'Time Over_plan'] = pd.to_datetime(full_df.
↪loc[index, 'Time Over_plan'], format = "%Y-%m-%d %H:%M:%S")

            full_df.loc[index, "delay"] = full_df.loc[index, 'Time Over'] -␣
↪full_df.loc[index, 'Time Over_plan']

            if full_df.loc[index,"delay"].days < 0:
                full_df.loc[index,"delay"] = -1*int(abs(full_df.
↪loc[index,"delay"]).seconds/60)
            else:
                full_df.loc[index,"delay"] = int(abs(full_df.
↪loc[index,"delay"]).seconds/60)

        else:
            raise ValueError


    start_i += 2
    start_j = flight_level_0[start_i]
```

A Code (models)

Python code for Chapter 5.1. Regression model

This is the code for basic regression model (OLS). Current code was used twice: first time for flight delay (that version), second time for remaining time until landing (with target change).

[1]:
```
# basic libs
import pandas as pd
import numpy as np
import pickle

#sklearn lib
from sklearn.metrics import mean_squared_error, mean_absolute_error,␣
↪mean_absolute_percentage_error, r2_score, explained_variance_score
from sklearn import preprocessing
from sklearn.linear_model import LinearRegression

# stat tools
from scipy.stats import anderson
import statsmodels.api as sm
from statsmodels.stats.diagnostic import lilliefors as ks
from statsmodels.stats.stattools import durbin_watson
from statsmodels.stats.diagnostic import het_goldfeldquandt
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

#visialization
import seaborn as sns
import matplotlib.pyplot as plt
```

[15]:
```
df = pd.read_csv('full_df4.csv')
```

[17]:
```
le1 = preprocessing.LabelEncoder()
df['airport_start'] = le1.fit_transform(df['airport_start'])
le2 = preprocessing.LabelEncoder()
df['country_start'] = le2.fit_transform(df['country_start'])
le3 = preprocessing.LabelEncoder()
df['airport_end'] = le3.fit_transform(df['airport_end'])
le4 = preprocessing.LabelEncoder()
df['country_end'] = le4.fit_transform(df['country_end'])
```

[18]:
```
cols = df.columns
```

[19]:
```
scaler = preprocessing.MinMaxScaler()
df = scaler.fit_transform(df)
df = pd.DataFrame(df, columns=cols)
```

[21]:
```
target = 'delay'
target2 = 'remaining_time'
```

1 Prediction part

[22]:
```
train_df = df[:633605]
test_df = df[633605:]

y_train1 = train_df[target1]
y_test1 = test_df[target1]
x_train1 = train_df.drop(columns=[target1, target2])
x_test1 = test_df.drop(columns=[target1, target2])
```

[25]:
```
# stats model initial
```

[33]:
```
x_train1_ = sm.add_constant(np.array(x_train1))
est = sm.OLS(y_train1, x_train1_).fit()
print(est.summary());
```

[34]:
```
# reproduce with significant variables

not_significant = x_train1.columns[[2, 3, 11]].tolist()
x_train1_ = sm.add_constant(np.array(x_train1.drop(columns=not_significant)))
est = sm.OLS(y_train1, x_train1_).fit()
print(est.summary());
```

[29]:
```
# test data evaluation
y_pred1 = est.predict(x_test1.drop(columns=not_significant))
```

[31]:
```
# metrics
mse1 = mean_squared_error(y_test1, y_pred1)
mae1 = mean_absolute_error(y_test1, y_pred1)
mape1 = mean_absolute_percentage_error(y_test1, y_pred1)
r21 = r2_score(y_test1, y_pred1)
exp_var1 = explained_variance_score(y_test1, y_pred1)
```

2 Assumptions

[ ]:
```
for col in df.drop(columns=[target1, target2]):
    plt.scatter(df[col], df[target2], s=0.1)
    name = str(df[col].name)
    plt.xlabel(name)
    plt.ylabel("target")
    plt.show()
```

[ ]:
```
# Extract the residuals
resid = est.resid
```

[ ]:
```
y_predtr1 = lm.predict(x_train1.drop(columns=not_significant))
df_results = pd.DataFrame({'actual': y_train1, 'predicted': y_predtr1,␣
 ↪'residuals': resid})
```

[ ]:
```
sns.lmplot(y='actual', x='predicted', data=df_results, fit_reg=False, size=7,␣
 ↪scatter_kws={"s": 2}, line_kws={'color': 'grey'})
line_coords = np.arange(df_results.min().min(), df_results.max().max())
plt.plot(line_coords, line_coords,
         color='black', linestyle='--')
plt.xlim(0, 0.8)
plt.ylim(0, 1)
```

[ ]:
```
# Kolmogorov-Smirnov test
_, p = ks(resid, dist='norm')
print('Not normal | p-value:' if p < 0.05 else 'Normal | p-value:', p)
print('-------------------------------')
```

[ ]:
```
# Anderson
stat, p3, _ = anderson(resid, dist='norm')
print('Anderson:')
print('Not normal | stat:' if stat > p3[2] else 'Normal | stat:', stat, '::␣
 ↪p-value:', p3[2])
```

[ ]:
```
# QQPlot
plt.figure(figsize=(5,5))
scipy.stats.probplot(resid, dist='norm', plot=plt)
plt.show()
```

[ ]:
```
plt.subplots(figsize=(12, 6))
sns.distplot(resid)
plt.show()
```

[ ]:
```
durbinWatson = durbin_watson(resid)
print('Durbin-Watson:', durbinWatson)
```

[ ]:
```
plt.subplots(figsize=(12, 6))
ax = plt.subplot(111)
plt.scatter(x=df_results.index, y=df_results.residuals, alpha=0.3, s=5, c='tab:
 ↪blue')
plt.plot(np.repeat(0, df_results.index.max()), color='black', linestyle='--')
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.show()
```

[ ]:
```
# goldfeldquandt test
het_goldfeldquandt(resid, est.model.exog)
```

[ ]:
```
# VIF Test
vif = add_constant(x_train1)
pd.Series([variance_inflation_factor(vif.values, i) for i in range(vif.
 ↪shape[1])], index=vif.columns)
```

Python code for Chapter 5.2. Deep neural networks

This is basic code for Deep neural network (following architecture is used: 64/64/32/32/1(2)+bath normalization). Curent code was used for each target type (flight delay, remaining time) and for each model type (DNN regression, DNN probabilistic (twice: for Normal and Weibull)).

Based on target type - x and y value are changed, based of model type - if regresson output shape is 1 and loss function MSE, if probabilistic - output shape is 2 and loss function is negative log lokelohood.

Additionally, different architectures were applied (in class Model(Module)), like 64/32/16/8/1(2), 128/64/32/16/1(2), 512/256/138/32/1(2).

bath normalization, ReLU activation function, L2 regularizaion, fluctuating learning, dropout and they combinations were added to training process. There is no point to include all versions, as there are pretty similar (about 80%).

[1]:
```
import torch
print(torch.__version__)
```

```
1.9.0+cu102
```

[2]:
```
from torch.utils.data import DataLoader, Dataset
from torch.nn import Linear, Dropout
from torch.nn import ReLU
from torch.nn import Sigmoid
from torch import Tensor
from torch.nn import Module
from torch.optim import SGD, Adam
from torch.nn import MSELoss
from torch.optim import lr_scheduler
from torch.nn.init import xavier_uniform_
from torch.nn import functional as F
```

[3]:
```
import pandas as pd
from sklearn.metrics import mean_squared_error, mean_absolute_error,␣
↪mean_absolute_percentage_error, r2_score, explained_variance_score
import numpy as np
from sklearn import preprocessing
```

[7]:
```
import matplotlib.pyplot as plt
```

[49]:
```
df = pd.read_csv('full_df4.csv')
```

[16]:
```
le1 = preprocessing.LabelEncoder()
df['airport_start'] = le1.fit_transform(df['airport_start'])
le2 = preprocessing.LabelEncoder()
df['country_start'] = le2.fit_transform(df['country_start'])
le3 = preprocessing.LabelEncoder()
df['airport_end'] = le3.fit_transform(df['airport_end'])
le4 = preprocessing.LabelEncoder()
df['country_end'] = le4.fit_transform(df['country_end'])
```

[17]:
```
trans = preprocessing.MinMaxScaler()
df = trans.fit_transform(df)
df = pd.DataFrame(df, columns=cols)
```

[28]:
```
train_df = df[:633605]
test_df = df[633605:]
```

[29]:
```
target1 = 'delay'
target2 = 'remaining_time'
```

1 Prepare the Data.

[30]:
```
# dataset definition
class CSVDataset(Dataset):

    def __init__(self, df, target_use, target_drop):

        self.X = df.drop(columns=[target_use, target_drop]).values.astype(float)
        self.y = df[target_use].values.astype(float)
        print(df[target_use].name)
        self.y = self.y.reshape((len(self.y), 1))
        self.X = torch.tensor(self.X).float()
        self.y = torch.tensor(self.y).float()

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return [self.X[idx], self.y[idx]]
```

[31]:
```
def prepare_data(train_df, test_df, target_1, target_2):

    train = CSVDataset(train_df, target_1, target_2)
    test = CSVDataset(test_df, target_1, target_2)
    train_dl = DataLoader(train, batch_size=256, shuffle=True)
    test_dl = DataLoader(test, batch_size=256, shuffle=False)

    return train_dl, test_dl
```

[1]:
```
train_dl, test_dl = prepare_data(train_df, test_df, target1, target2)
print(len(train_dl.dataset), len(test_dl.dataset))
```

[2]:
```
len(train_dl.dataset)/(len(train_dl.dataset)+len(test_dl.dataset))
```

2 Define the Model.

[36]:
```
# model definition
class Model(Module):

    def __init__(self, n_inputs):
        super(Model, self).__init__()

        self.hidden1 = Linear(n_inputs, 64)
        xavier_uniform_(self.hidden1.weight)
        self.act1 = torch.nn.BatchNorm1d(64)

        self.hidden2 = Linear(64, 64)
        xavier_uniform_(self.hidden2.weight)
        self.act2 = torch.nn.BatchNorm1d(64)

        self.hidden3 = Linear(64, 32)
        xavier_uniform_(self.hidden3.weight)
        self.act3 = torch.nn.BatchNorm1d(32)

        self.hidden4 = Linear(32, 32)
        xavier_uniform_(self.hidden4.weight)
        self.act4 = torch.nn.BatchNorm1d(32)

        self.hidden5 = Linear(32, 1)
        xavier_uniform_(self.hidden5.weight)

        self.dropout = Dropout(0.2)

    def forward(self, X):

        X = self.hidden1(X)
        X = self.act1(X)

        X = self.hidden2(X)
        X = self.act2(X)

        X = self.hidden3(X)
        X = self.act3(X)

        X = self.hidden4(X)
        X = self.act4(X)

        X = self.hidden5(X)

        X = self.dropout(X)

        return X
```

[39]:
```
# define the network
model = Model(df.shape[1]-2)
```

3 Train the Model.

[40]:
```
#torch.autograd.set_detect_anomaly(True)
```

[ ]:
```
def loss_proba_Normal(ypreds, ytrue, dist_type):

    if dist_type =="Normal":
        mu = ypreds[:, 0]
        std = torch.nn.functional.softplus(ypreds[:, 1])
        dist = torch.distributions.Normal(mu, std)
        loss = -1.0 * dist.log_prob(ytrue)

    return loss.sum()
```

[ ]:
```
def loss_proba_Weibull(ypreds, ytrue, dist_type):

    if dist_type=='Weibull':
        eps=1e-7
        lambda_ = torch.nn.functional.softplus(ypreds[:, 0])
        k = torch.nn.functional.softplus(ypreds[:, 1])
        dist = torch.distributions.Weibull(lambda_, k)

        ytrue_eps = ytrue
        for i in ytrue_eps:
            if i == 0:
                i += eps
        loss = -1.0 * dist.log_prob(ytrue_eps)
    return loss.sum()
```

[41]:
```
def train_model(train_dl, model):

    criterion = MSELoss(reduction='mean')
    optimizer = Adam(model.parameters(), lr=0.001)
    losses=[]
    for epoch in range(50):
        print("epoch: ", epoch)
        loss_l = []
        for i, (inputs, targets) in enumerate(train_dl):
            optimizer.zero_grad() # clear gradients
            y_pred = model(inputs) # compute output
            loss = criterion(y_pred, targets) # calculate loss
            loss_l.append(loss)
            loss.backward() # credit assignment
            optimizer.step() # update model weights

        loss_l = [i.item() for i in loss_l]
        losses.append(np.mean(loss_l))

        print("loss: ", np.mean(loss_l))
        print("-------")
    return losses
```

[3]:
```
# train the model
losses=train_model(train_dl, model)
```

[4]:
```
x = list(range(len(losses)))
y = losses
plt.xlabel("iterations")
plt.ylabel("losses")
plt.title("Losses")
plt.plot(x,y)
```

4 Evaluate the Model.

[44]:
```
def evaluate_model(test_dl, model, dist_type):
    pred, true = list(), list()
    for i, (inputs, targets) in enumerate(test_dl):

        y_pred = model(inputs)
        y_pred = y_pred.detach().numpy()
        y_true = targets.numpy()
        y_true = y_true.reshape((len(y_true), 1))
        pred.append(y_pred)
        true.append(y_true)

    pred, true = np.vstack(pred), np.vstack(true)
    if dist_type=='Normal':
        mean_val = pred[:, 0]
    if dist_type=='Weibull':
        mean_val = pred[:, 0]*gamma(1+1/pred[:, 1])

    mse = mean_squared_error(true, pred)
    mae = mean_absolute_error(true, pred)
    mape = mean_absolute_percentage_error(true, pred)
    r2 = r2_score(true, pred)
    exp_var = explained_variance_score(true, pred)

    return mse, mae, mape, r2, exp_var, pred, true
```

[45]:
```
# evaluate the model

mse, mae, mape, r2, exp_var, pred, true = evaluate_model(test_dl, model,␣
 ↪dist_type='none')
```

[5]:
```
mse, mae, mape, r2, exp_var
```

Python code for Chapter 5.3. Transformer neural networks

This is basic code for Transformer neural network. Curent code was used for each target type (flight delay, remaining time) and for each model type (TNN regression, TNN probabilistic (twice: for Normal and Weibull)).

Based on target type - x and y value are changed, based of model type - if regresson output shape is 1 and loss function MSE, if probabilistic - output shape is 2 and loss function is negative log lokelohood.

Additionally, L2 regularizaion, fluctuating learning and dropout were added to training process. There is no point to include all versions, as there are pretty similar (about 90%).

[1]:
```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt

from tensorflow import keras
from tensorflow.keras import layers
import tensorflow as tf
import tensorflow_probability as tfp

from sklearn.metrics import mean_squared_error, mean_absolute_error,␣
 ↪mean_absolute_percentage_error, r2_score, explained_variance_score
from scipy.special import gamma
```

[3]:
```
df = pd.read_csv('full_df4.csv')
```

[6]:
```
le1 = preprocessing.LabelEncoder()
df['airport_start'] = le1.fit_transform(df['airport_start'])
le2 = preprocessing.LabelEncoder()
df['country_start'] = le2.fit_transform(df['country_start'])
le3 = preprocessing.LabelEncoder()
df['airport_end'] = le3.fit_transform(df['airport_end'])
le4 = preprocessing.LabelEncoder()
df['country_end'] = le4.fit_transform(df['country_end'])
```

[7]:
```
trans = preprocessing.MinMaxScaler()
df = trans.fit_transform(df)
df = pd.DataFrame(df, columns=cols)
```

[8]:
```
train_df = df[:633605]
test_df = df[633605:]
```

[9]:
```
target1 = 'delay'
target2 = 'remaining_time'
```

[10]:
```
x_train = train_df.drop(columns=[target1, target2])
y_train = train_df[target1]
```

[11]:
```
x_test = test_df.drop(columns=[target1, target2])
y_test = test_df[target1]
```

[12]:
```
x_train=x_train.to_numpy()
y_train=y_train.to_numpy()
```

[13]:
```
x_test=x_test.to_numpy()
y_test=y_test.to_numpy()
```

[14]:
```
x_train = x_train.reshape((x_train.shape[0], x_train.shape[1], 1))
x_test = x_test.reshape((x_test.shape[0], x_test.shape[1], 1))
```

[15]:
```
x_train.shape, x_test.shape
```

[15]:
```
((633605, 17, 1), (111864, 17, 1))
```

[16]:
```
y_train.shape, y_test.shape
```

[16]:
```
((633605,), (111864,))
```

1 Training part

[39]:
```
tfd = tfp.distributions
```

[20]:
```
def transformer_encoder(inputs, head_size, num_heads, ff_dim, dropout=0):
    # Normalization and Attention
    x = layers.LayerNormalization(epsilon=1e-6)(inputs)
    x = layers.MultiHeadAttention(
        key_dim=head_size, num_heads=num_heads, dropout=dropout
    )(x, x)
    x = layers.Dropout(dropout)(x)
    res = x + inputs

    x = layers.LayerNormalization(epsilon=1e-6)(res)
    x = layers.Conv1D(filters=ff_dim, kernel_size=1, activation="relu")(x)
    x = layers.Dropout(dropout)(x)
    x = layers.Conv1D(filters=inputs.shape[-1], kernel_size=1)(x)
    return x + res
```

[21]:
```
def build_model(
    input_shape,
    head_size,
    num_heads,
    ff_dim,
    num_transformer_blocks,
    mlp_units,
    dropout,
    mlp_dropout,
    output_shape
):
    inputs = keras.Input(shape=input_shape)
    x = inputs
    for _ in range(num_transformer_blocks):
        x = transformer_encoder(x, head_size, num_heads, ff_dim, dropout)

    x = layers.GlobalAveragePooling1D(data_format="channels_first")(x)
    for dim in mlp_units:
        x = layers.Dense(dim, activation="relu")(x)
        x = layers.Dropout(mlp_dropout)(x)
    outputs = layers.Dense(output_shape, activation='linear')(x)
    return keras.Model(inputs, outputs)
```

[ ]:
```
def loss_f_probabilistic_normal(ytrue, ypreds):
    mu = ypreds[:, 0]
    sd = tf.keras.activations.softplus(ypreds[:, 1])
    dist = tfd.Normal(loc=mu, scale=sd)
    return tf.math.reduce_sum(-1.0 * dist.log_prob(ytrue))
```

[ ]:
```
def loss_f_probabilistic_weibull(ytrue, ypreds):
    eps=1e-7
    lambda_ = tf.keras.activations.softplus(ypreds[:, 0])
    k = tf.keras.activations.softplus(ypreds[:, 1])
    dist = tfd.Weibull(k, lambda_)
    ytrue_eps = ytrue
    for i in ytrue_eps:
        if i == 0:
            i += eps
    return tf.math.reduce_sum(-1.0 * dist.log_prob(ytrue))
```

[2]:
```
model = build_model(
    input_shape=x_train.shape[1:],
    head_size=256,
    num_heads=4,
    ff_dim=4,
    num_transformer_blocks=4,
    mlp_units=[128],
    mlp_dropout=0,
    dropout=0,
    output_shape=1
)

model.compile(
    loss="mean_squared_error",
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    metrics=["mean_squared_error", "mean_absolute_error",␣
 ↪"mean_absolute_percentage_error"],
)
model.summary()
```

[3]:
```
model.fit(
    x_train,
    y_train,
    validation_split=0.0,
    epochs=50,
    batch_size=256,
    callbacks=None
)
```

2 Testing part

[4]:
```
model.evaluate(x_test, y_test, batch_size=256)
```

[5]:
```
len(losses)
```

[6]:
```
x = list(range(len(losses)))
y = losses
plt.xlabel("iterations")
plt.ylabel("losses")
plt.title("Losses")
plt.plot(x,y)
```

[35]:
```
dist_type = 'none'
pred = model.predict(x_test)

if dist_type=='Normal':
    pred = pred[:, 0]
if dist_type=='Weibull':
    pred = pred[:, 0]*gamma(1+1/pred[:, 1])
```

```python
pred = model.predict(x_test)
mse = mean_squared_error(y_test, pred)
mae = mean_absolute_error(y_test, pred)
mape = mean_absolute_percentage_error(y_test, pred)
r2 = r2_score(y_test, pred)
exp_var = explained_variance_score(y_test, pred)
```

[7]:
```python
mse, mae, mape, r2, exp_var
```