jamk.fi

Predicting traffic incidents using open data sources

Onni Hakkari

Bachelor's thesis
February 2019
Technology, communication and transport
Degree Programme in Information Technology

jamk.fi

Description

Author(s)	Type of publication	Date
Hakkari, Onni	Bachelor's thesis	February 2019
		Language of publication: English
	Number of pages: 60	Permission for web publication: x

Title of publication
Predicting traffic incidents using open data sources

Degree programme
Information Technology

Supervisor(s)
Rantala, Ari
Huotari, Jouni

Assigned by
Mika Rantonen

Abstract

The purpose of the research was to examine if openly available data sources contain features that affect the occurrence and severity of traffic incidents.

The implementation contains data engineering, feature engineering and building of predictive models. Time series data was used, and combining the time series data became a major part of the research, because new software had to be developed that can combine many time series datasets with different time frequencies. The software had to be implemented in memory efficient manner, since the datasets were too large for memory to handle every dataset at the same time.

As a result, no descriptive feature could be found affecting the amount of traffic incidents or the severity of traffic incidents. However, an interesting perception was made, where it was discovered that while the amount of traffic incidents remains approximately the same no matter what season of year it is; however, in summer, the traffic incidents tend to be severe. One hypothesis is that it could be due motorcycles, because one could draw the conclusion in which traffic incidents involving motorcycles are more severe than those involving only four-wheel vehicles.

Even though the data was not descriptive enough to proceed to building polished predictive neural network models, plenty of data analysis methods were applied to explore how descriptive the data is. Beneficial work has been done, since the developed data combining software can be open sourced in the future, which can offer benefits for other data scientists who need to combine complex time series data.

Keywords/tags (subjects)
Feature selection, Feature importance, Data engineering, Neural network, Python

Miscellaneous (Confidential information)

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Hakkari, Onni	Opinnäytetyö, AMK	Helmikuu 2019
	Sivumäärä 60	Julkaisun kieli Englanti
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Predicting traffic incidents using open data sources

Tutkinto-ohjelma
Tieto- ja viestintätekniikka

Työn ohjaaja(t)
Ari Rantala
Jouni Huotari

Toimeksiantaja(t)
Mika Rantonen

Tiivistelmä

Tutkimuksen tarkoituksena oli selvittää, löytyykö avoimista datalähteistä dataa, joka vaikuttaa liikenneonnettomuuksien tapahtumiseen ja vakavuuteen.

Toteutusosuus pitää sisällään asioita kuten data engineering, feature engineering ja ennustavien mallien ohjelmointia. Data oli aikasarjadataa, ja eri lähteistä olevien aikasarjadatojen yhdistäminen samaan data-aineistoon muodostui isoksi osaksi tutkimusta, koska tarpeisiin sopivaa ohjelmaa ei löytynyt; näin ollen piti kehittää uusi ohjelma. Ohjelman piti pystyä yhdistelemään useita data-aineistoja, joilla on eri frekvenssi ajan suhteen. Ohjelmisto täytyi toteuttaa muistitehokkaasti, koska data-aineistot olivat liian suuria ladattavaksi muistiin samanaikaisesti.

Datasta ei löydetty tarpeeksi kuvailevia ominaisuuksia, jotka olisi vaikuttaneet liikenneonnettomuuksien määrään tai vakavuuteen. Mielenkiintoista oli, että vaikka liikenneonnettomuuksien määrä pysyy suunnilleen samana riippumatta vuodenajasta, kesällä liikenneonnettomuudet ovat kuitenkin vakavampia. Tämän saattaisi selittää moottoripyörien ajokauden ajankohta, koska voisi päätellä, että liikenneonnettomuudet, joissa moottoripyörä on osallisena, ovat yleensä vakavampia ja useammin kuolemaan johtavia.

Vaikka data ei ollutkaan tarpeeksi kuvaileva, jotta olisi voinut alkaa hiomaan hienoja ennustavia neuroverkkomalleja, monia data-analyysimetodeja käytettiin datan kuvailevuuden tutkimiseen. Työ on tuottanut hyötyä, koska kehitetyn aikasarjadatan yhdistelytyökalun voi jatkossa julkaista avoimen lähdekoodin projektina, jolloin se tuottaa hyötyä myös muille tutkijoille.

Avainsanat (asiasanat)
Feature selection, Feature importance, Data engineering, Neural network, Python

Muut tiedot (salassa pidettävät liitteet)

Terminology **5**

1 Introduction **6**

1.1 Motivation 6

1.2 New Business Innovations from Data-analytics public abstract 6

1.3 Development environment 9

2 Data analysis **10**

2.1 Linear correlation 10

2.2 LOWESS (Locally Weighted Scatterplot Smoothing) 10

2.3 Normalization 11

2.4 Standard deviation 11

2.5 Interpolation 11

2.6 Predictive models 12

2.6.1 Classification 12

2.6.2 Regression 12

2.7 Balancing data 13

2.8 Feature selection 13

2.9 Neural networks 16

2.10 Overfitting 17

2.10.1 Detecting overfitting 18

2.10.2 Preventing overfitting 19

2.11 Regressors 19

2.11.1 Decision tree 19

2.11.2 Random forest 20

2.12 Data quality 20

3 Combining datasets **22**

3.1 Downloading data 22

3.1.1 LAM 23
3.1.2 Road weather 23
3.1.3 Tilannehuone (Situation room) 24
3.1.4 Traffic incidents 24
3.1.5 Finnish Meteorological Institute 24
3.1.6 Sunrise and sunset 25
3.2 Combining data 25
3.2.1 Adding timestamp to dataset 26
3.2.2 Dataset combining methods 29
3.2.3 Combining method options 31
3.2.4 Memory efficiency 36

4 Regression (predicting continual variable) 38
4.1 Dataset 38
4.2 Feature engineering 39
4.3 Feature selection 40
4.3.1 Fit line between two variables 40
4.3.2 Feature importance 43
4.4 Neural networks 43
4.5 Regressors 45
4.5.1 Random Forest Regressor 45
4.5.2 Decision Tree Regressor 47
4.6 ReLU zero predictions (dying ReLU) 47

5 Binary classification (predicting categorical variable) 49
5.1 Dataset 49
5.2 Feature engineering 50
5.3 Time accuracy 50

5.4 Feature selection 50

6 Occasions for traffic incidents 52

7 Conclusion 55

References 58

Figure 1. LOWESS example 10
Figure 2. Bad variable for binary classification 15
Figure 3. Good variable for binary classification 15
Figure 4. Example cost and accuracy 17
Figure 5. underfit vs optimal vs overfit 18
Figure 6. k-fold example 19
Figure 7. Decision tree example split 20
Figure 8. LAM locations 23
Figure 9. Weather station location 25
Figure 10. Dataset tool help 26
Figure 11. Add timestamp command 27
Figure 12. Timestamp adding function code 29
Figure 13. Shortest timestamp adding command 29
Figure 14. Combiner usage 31
Figure 15. Combining methods code 36
Figure 16. Temporary file creating code 38
Figure 17. Air temperature vs amount of deaths 40
Figure 18. Air temperature vs amount of injuries 41
Figure 19. Air temperature vs amount of incidents 41
Figure 20. Air temperature vs road surface friction 42
Figure 21. Months vs amount of deaths 43
Figure 22. Common cost and accuracy result 45
Figure 23. Random Forest Regressor prediction distribution histogram 46
Figure 24. Random Forest Regressor prediction distribution scatter 46
Figure 25. Decision Tree Regressor prediction distribution 47
Figure 26. Example Keras model with dying ReLU risk 48

Figure 27: One feature with dying ReLU problem 49
Figure 28. Road surface friction distribution 51
Figure 29: Air temperature distribution 52
Figure 30: Road surface minimum friction outlier 53
Figure 31: Air temperature, where minimum road surface friction is between 0.75 and 0.8 54
Figure 32: Road surface maximum friction occasions 55

Table 1. Dirty data 22
Table 2. Example data GSPC.csv 27
Table 3. Example data GSPC.csv with timestamp 28
Table 4. Forward fill example 30
Table 5. Mean method 31
Table 6. Sum method 32
Table 7. Datapoints method 32
Table 8. Min method 33
Table 9. Max method 33
Table 10. Standard deviation method 34
Table 11. Categorical method 34
Table 12. Categorical_sum method 35
Table 13. lam_dataset.csv example rows 37
Table 14: Compressed feature(s) 39
Table 15. LSTM data example 44
Table 16. Dense data example 44

Terminology

GB	Gigabyte
LOESS	Locally Weighted Scatterplot Smoothing
LOWESS	Locally Weighted Scatterplot Smoothing
LSTM	Long short-term memory
MB	Megabyte
ReLU	Rectified Linear Unit

1 Introduction

1.1 Motivation

The purpose of this thesis is to research the factors causing traffic incidents and make predictions for traffic incidents. The thesis was assigned by JAMK University of Applied Sciences (JAMK), and it is a part of the project New Business Innovations from Data analytics. European Regional Development Fund (ERDF) is the investor for the project New Business Innovations from Data analytics.

Predicting traffic incidents requires combining time series data from different data sources, which allows one to enrich the existing data with additional information that could be utilized to gain more insight and new factors affecting a certain feature that one wants to predict. Before it is possible to make predictions from many different data sources, there is a challenge to be solved: how to combine a huge amount of data when one does not have a super computer with an infinite amount of memory available? This thesis contains two (2) "hands-on" sections, where the first one is about developing new software that allows combining time series data from different data sources memory efficiently, and the second one discusses making predictions and analyzing combined data that the developed software has produced.

1.2 New Business Innovations from Data-analytics public abstract

The following public abstract is copied from the application of the project funding sent to ERDF. (European Regional Development Fund funded project abstract N.d.)

> *With the fast development of technology, the term "Big Data" has gained strong publicity. Data is produced at accelerating pace almost everywhere where people's actions and various, mainly electronic systems meet. There is no accurate definition for the term Big Data, however, generally a vast volume of data, in addition to variety and fast velocity in the increase of the data, are required.*

Big Data as such is raw material without any added value unless it can be refined. Often Big Data is compared to crude oil, which acts as raw material and through refining process produces new products for industry. With Big Data, one can think exactly in the same way. With data analytics, enormous masses of data can be processed and assimilated independent of whether the data is located in a database or real-time dataflow. In addition to gathering and storing data, data masses must be processed, analyzed and visualized, i.e. present it to people in illustrative form to be used as a tool for business planning and as an instrument for management. With data processing, routine work tasks can be automated and external, open source data sources can be added to a company's own data to "enrich" its contents and produce added value to business.

With the development of data analytics environments and tools, utilizing data and ***combining various data-/information sources*** *have become a new opportunity for companies to develop business. In addition to conventional Business Intelligence (BI), it should be possible to make proactive decisions with data analytics. The data produced by companies may include very critical information as far as data protection is concerned, and the storing, filtering and data security need to be taken into account. Data analytics environments based on cloud services face challenges in data security, -protection and juridical questions. An alternative to cloud services is to build a company's own data analytics environment. Besides, the EU General Data Protection Regulation defines sanctioned obligations to companies concerning processing and securing data.*

The needs and the current state of art to utilize data analytics behind this project were surveyed in discussions with several companies in Central Finland. In general, it can be stated that companies have in their databases a great amount of so-called dark data, which could be utilized to create added value and new

innovations. Dark data is data with added value that has never previously been processed or cannot be yet utilized in a company's own business. In addition, companies have a great need for data analytics know-how in order to refine their data to bring about the needed changes in their business models in the digitalization era. In the job markets, there is also a great shortage of personnel with this kind of know-how.

The purpose of the project is to add to data analytics know-how as well as develop and pilot concrete solutions of dark data utilizing data analytics solutions, which help companies to create new business opportunities or increase the growth in their business and add to the amount of vacant jobs. The project measures comprise also building of an integrated, data secure data analytics development environment. Companies in energy sector or businesses closely connected to it are committed to participate in the project and co-finance it. The energy section was selected since the production and consumption of energy affect in one way or other citizens' everyday lives, and thus present a potential field of business development. The participating companies all being in the same branch of industry enable inspection of data analytics and its development in the entire value chain: Production <-> Distribution <-> Consumption <-> Customers. Eight (8) pilot cases were created based on the general development needs for data analytics, and concrete needs brought out by the participating companies. These cases form the content raw material and foundation for the development work. Most of these cases include common interests between the participating companies.

As dissemination and exploitation activities, information gathered as results of the project measures on utilization opportunities of data analytics is exploited to companies operating in the area.

The project will accomplish following results:

An integrated, information-secure development environment enabling the use of tools and usage of data analytics has been built, piloted and implemented.

New operating models are found to utilize dark data through data analytics.

Utilization of AR/VR/MR technologies for data visualization has been developed, and the new methods have been demonstrated.

The knowledge of data-analytics utilization opportunities is increased in companies and other organizations operating in Central Finland area.

1.3 Development environment

For the dataset combining tool development and other programming implementations the choice for the programming language was Python, and Visual Studio Code and VIM were chosen as text editor. The development was carried out on Linux and Windows operating systems, depending on the physical location at the time. The data analysis part was done in Jupyter notebook, which allows one to write Python or R (Python was chosen in this thesis) and markdown language to the notebook, to create a report like document that can contain code, figures and documentation. Machine learning algorithms and models were implemented using Python library Keras, which uses TensorFlow in the backend. Keras makes it easier to create fast model prototypes, where pure TensorFlow is slower to implement. For version control, Git was used and for project management, Gitlab was used.

2 Data analysis

2.1 Linear correlation

Pandas linear Pearson correlation can be calculated using method `corr()` from Pandas DataFrame object. Correlation helps to understand if there is any linear relation between variable x and y. Linear correlation can be implemented with following equation: (Pearson Correlations – Quick Introduction N.d; McCallister N.d.)

$$r = \frac{N\sum xy - (\sum x)(\sum y)}{\sqrt{[N\sum x^2 - (\sum x)^2][N\sum y^2 - (\sum y)^2]}} \quad (1)$$

2.2 LOWESS (Locally Weighted Scatterplot Smoothing)

LOWESS is often used to fit a line in scatterplot to see relationships between variables, where linear line plot does not provide enough information and does not fit very well to data. Python seaborn visualization library uses implementation of LOWESS from statsmodels library. LOWESS line fitted to example data is illustrated in Figure 1. (Stephanie 2013; Cleveland 1979.)

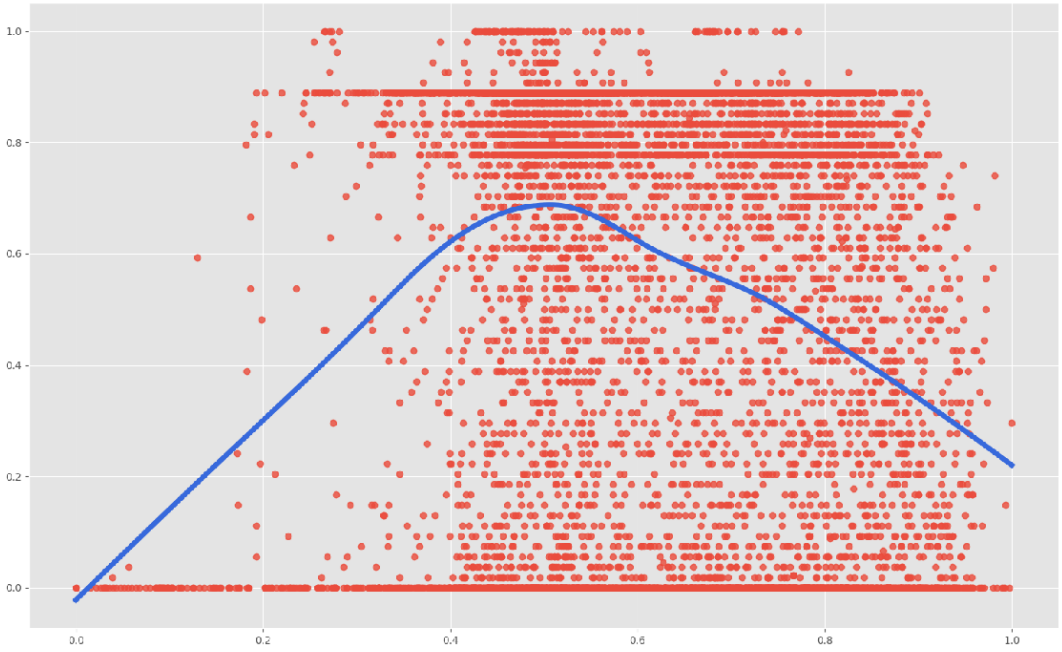


Figure 1. LOWESS example

In LOWESS function, the weights for data points change; hence, the equation for LOWESS used on different data changes. This means that there is no single simple

equation for LOWESS. (Could the equation of the curve provided by LOESS be obtained? 2017.)

2.3 Normalization

There are many normalization methods; however, `Min-Max scaling` was used during the writing of this thesis. Min-Max scaling scales values to the range of 0 – 1. For example, values 1,000, 2,000 and 3,000 would scale to 0, 0.5 and 1. Normalization is often used if a relation of few features to each other is examined where those features have values in a very different range. (Raschka 2014)

Normalization is also used for machine learning algorithms, because those algorithms have weights that it optimizes; however, if there are very large numbers in features that machine learning algorithm is trying to learn, optimizing those weights becomes slow and requires many iterations of training. When those large numbers are normalized to range of 0 – 1, it is much faster to train a machine learning model, because weights optimize faster. Min-Max scaling can be implemented with the following equation: (Raschka 2014)

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (2)$$

2.4 Standard deviation

With standard deviation, the dispersion of given values can be measured. Standard deviation is often used to summarize continuous data, in a way where there can be an idea how values in data variates without having to inspect every value manually. Mean method is often used with standard deviation. Standard deviation can be implemented with this equation: (Standard Deviation N.d.)

$$s = \sqrt{\frac{\sum(X - \bar{X})^2}{n-1}} \quad (3)$$

2.5 Interpolation

Interpolation in data analysis is used to fill non-existing values, also known as nan or null values. There are many interpolation methods, including linear interpolation,

moving average interpolation and other more complex kernel or neural network interpolation methods. In this thesis, only `forward fill` interpolation method is used, when combining data where some feature does have more frequently datapoints than other feature, so less frequent feature can be interpolated to have more frequent data, by using not so frequent data by filling null values with existing values, from closest available time. (Bourke 1999.)

Forward fill is very simple interpolation method. It fills null values, with previously seen existing value. Example array with no interpolation used:

$$[3, 5, null, null, 2, null, 1]$$

Example array with forward fill interpolation used:

$$[3, 5, 5, 5, 2, 2, 1]$$

2.6 Predictive models

2.6.1 Classification

Classification models try to predict a categorical outcome. There is binary classification, which involves only two categories (usually 0 and 1). Additionally, there are models that try to predict multiple categories. In this case, if categories can be sorted from worse to better, they can be represented as integer values; however, if no category is better than another, then it is best to represent them as one hot array. (Dayananda 2018.)

2.6.2 Regression

Regression is different from classification, because it tries to predict a value in continuous space, where classification tries to predict a discrete variable. For example, regression can be used to predict a value between 0-1, where classification predicts 0 or 1. (Dayananda 2018.)

2.7 Balancing data

Data balancing gains an important part of data analysis, when data is highly unbalanced. A good example of data balancing case is for certain the dataset from Kaggle that includes credit card information with binary label fraud or not fraud. Now, since credit card fraud payments are much rarer than not fraud or legit payments, the data is highly unbalanced. There are roughly 99% of legit transactions and 1% of fraud transactions. When training a model to learn fraud transactions on this kind of data, the reported accuracy can be misleading; since, if accuracy is 99%, one would think that it is a remarkably good accuracy; however, on this data 99% accuracy can be achieved by predicting always `not fraud`. This means that there is a 99% accuracy, while not a single fraud transaction was detected. In addition, this is a common outcome for such data, because the model learns more of `not fraud` than `fraud`; consequently, it will most likely always predict `not fraud`. There are two common options to balance the data: undersampling and oversampling. (Credit Card Fraud Detection 2018; In depth skewed data classif. (93% recall acc now) 2016.)

Undersampling balances the data by removing occurrences in the data of which there is too much. It is considered good practice to remove those occurrences randomly to prevent removing too much data from one section of dataset, where too much of certain important data could be lost. (In depth skewed data classif. (93% recall acc now) 2016.)

Oversampling, on the other hand, balances the data by copying data that of which there is very little, until there is the same amount of that data as the other data that of which there is too much. Usually oversampling rather copies the same amount of every existing occurrence of data, than randomly chooses occurrences to copy, because the data needs to be kept as much the same as possible before oversampling. (In depth skewed data classif. (93% recall acc now) 2016.)

2.8 Feature selection

Selecting the most relevant features from a dataset is an important step to take for getting the best results. One method in feature selection and often the first step is a manual method where a feature is defined which one wants to predict, for example,

the amount of traffic incidents, and then use those features one thinks that affect the feature to be predicted. For example, weather information such as the amount of snow, air temperature or length of day could affect the amount of traffic incidents. (Brownlee 2014.)

Selecting features by variance removes variables with too many of the same values. Removing by amount of the same values or close to same values is defined with threshold. Variance feature can be selected with `scikit` Python library. (Feature Selection N.d.)

For binary classification, a certain feature can be plotted by label, which is 0 or 1 as a histogram. This way it can be seen how values variate between those two labels. The more the values variate between those two labels, the better that feature is suitable for binary classification and potentially results in better prediction accuracy. By analyzing a feature this way, there is the attempt to evaluate the importance of certain feature. For example, in a situation where data distributions for label 0 and 1 overlap for the most part, that variable is not very suitable for training a binary classification model, because the model cannot see much difference with data between labels 0 and 1. This is illustrated in Figure 2. In a very different situation, where data distributions overlap only a little or not at all, it will be much easier for the model to see a difference between labels 0 and 1, because data is different between those labels. This is illustrated in Figure 3. This method is good for many cases where it is not certain if there are good features in the dataset, because when using algorithmic feature selection methods rather than this more “manual” histogram plotting method, one can be distracted if one does not know the inner function of such algorithm. This can lead to a continuing data analysis process, even if there is not a single good feature for predicting a certain variable. The use of the histogram plotting method is clearer and shows if there is something wrong about the data, so the data can be fixed, after which the data analysis process can continue. (Predicting Fraud with TensorFlow 2017.)

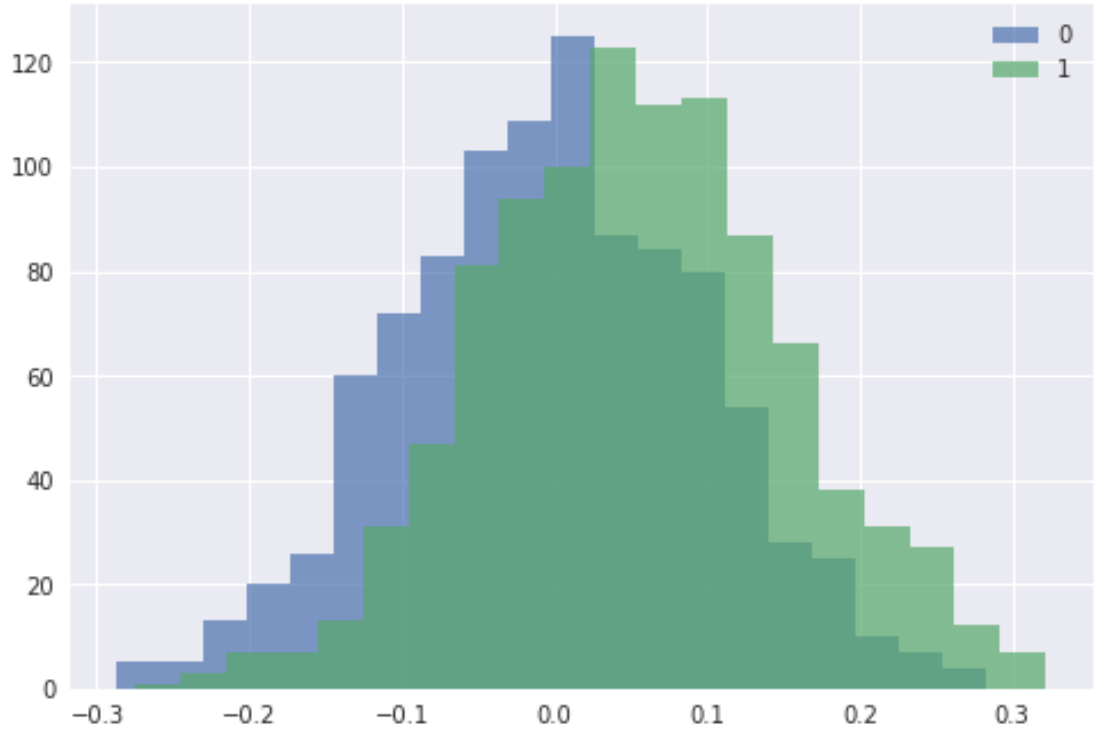


Figure 2. Bad variable for binary classification

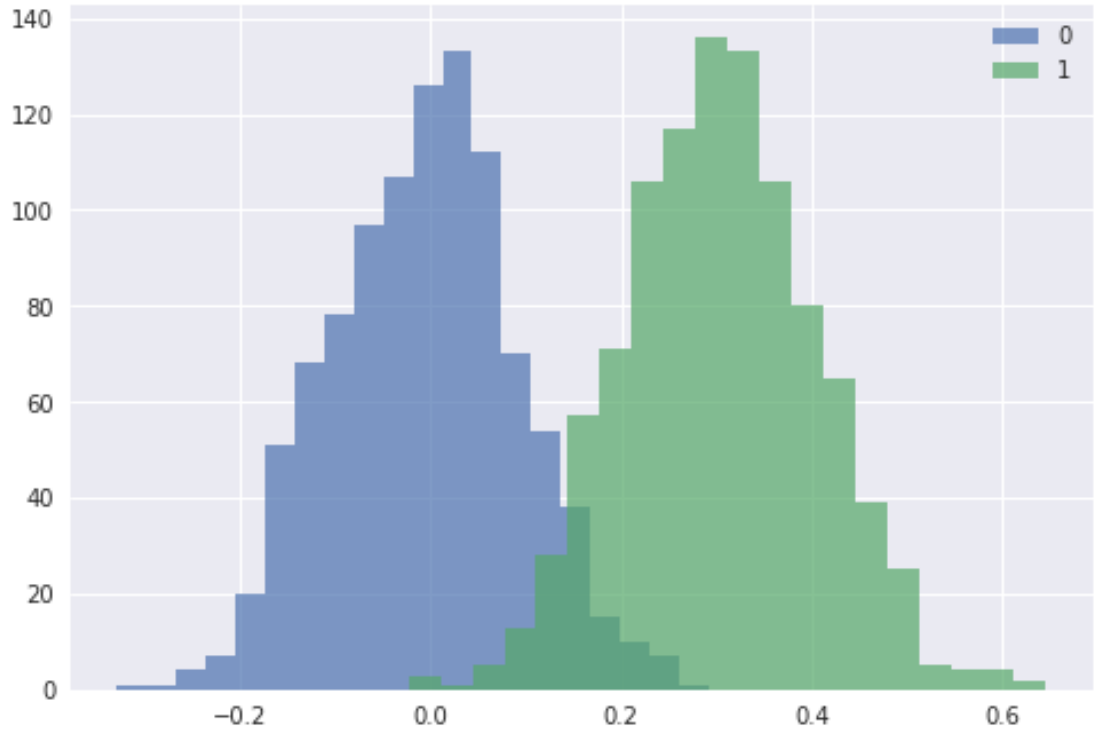


Figure 3. Good variable for binary classification

2.9 Neural networks

Artificial Neural Network is inspired by the way human brain works. It has some same elements as biological nervous systems, like neurons and synaptic connections between neurons. Those neurons can together solve complex problems, such as pattern recognition or natural language processing. Single neuron contains very simple math, but often, there are plenty of neurons in a neural network, and together they can be modeled for very complex problem solving. In some tasks, neural networks have been proven to perform better than humans. Neural networks have been popular recently, but it is by no means a recent invention, because first artificial neuron was produced in 1943. (Siganos & Stergiou N.d.)

When comparing neural networks to regular data analysis methods, advantages that neural networks have, is ability to find meaning from complicated data, and learning adaptively from data. In some cases, neural networks can reduce amount of manual work, because with example code of very generic model, neural network can be trained to be a powerful predictor, that can be trained to learn new data in real-time. Conventional computers use algorithmic methods to solve problems, where algorithms have steps which to follow in order to have problem solved. Neural network is also algorithmic method, but in a way where steps in algorithm can be changed to be better at solving certain problem. These “steps” are series of neurons and weights that defines how much effect each neuron has on certain input. Learning process is determination of the weights, and how those weights are determined, can be defined with different parameters often called `hyperparameters`. Hyperparameters are used to optimize model, and it is common to search for optimal hyperparameters with different methods, such as testing different combinations of hyperparameters and then evaluating each model with different hyperparameters to find out, which hyperparameters are the most optimal. Hyperparameters available are defined by methods used, because different methods have different properties. In hyperparameter optimization, most optimal hyperparameters are those, which minimizes cost function. (Siganos & Stergiou N.d; Amatriain 2016.)

For example, there are hyperparameters as follows:

- Learning rate defines how much to move weights

- Epochs means how many times neural network sees all data
- Number of hidden layers in a neural network
- Number of leaves of a decision tree
- Depth of a decision tree

In machine learning process most common metrics that are used in measuring and examining how well training went, are cost function and accuracy for every epoch. Cost function illustrates how “far” predicted value is from target value, and accuracy measures how accurate prediction was. When training neural network model, one would want to see similar cost and accuracy results as illustrated in Figure 4 for optimal results. In the example Figure 4 there is some noise, and it is not most optimal result for cost function and accuracy, but one can see that result for cost function should be decreasing over time when epochs iterate, because that means that predictions during training are getting closer and closer the target value(s) one is trying to predict. One would want opposite for accuracy, increasing over time when epochs are iterating, and when cost decreases, it results to higher accuracy. In the example Figure 4, neural network learns characteristics of the data and data is descriptive enough for the feature one is trying to predict. (Zheng 2015.)

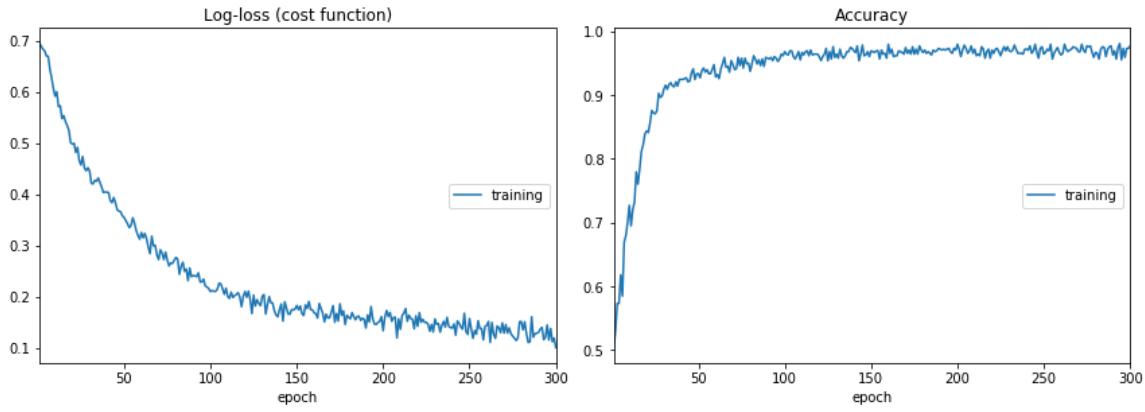


Figure 4. Example cost and accuracy

2.10 Overfitting

Overfitting can occur when a too complex model is used for a simple problem. It means that the model learns noise in the data one is trying to fit into the model. This

becomes a problem when the model learns noise in the training dataset; however, then one needs to use the model on real world data, and it does not have the same noise; hence, the predictive power will be reduced. When overfitting occurs, one can have a 99 percent accuracy on the training dataset; however, only, for example, 50% accuracy on testing data. (Overfitting in Machine Learning: What It Is and How to Prevent It 2017; Overfitting N.d.)

The opposite to overfitting is underfitting, and as opposite, underfitting occurs when the model is too simple. When underfit occurs, the model does not learn data characteristics one is trying to predict, and when overfitting takes place, the model "over-learns" data characteristics. When neither occurs, the situation is optimal, and the model has fitted well, as wanted. Figure 5 illustrates this. (Overfitting in Machine Learning: What It Is and How to Prevent It 2017.)

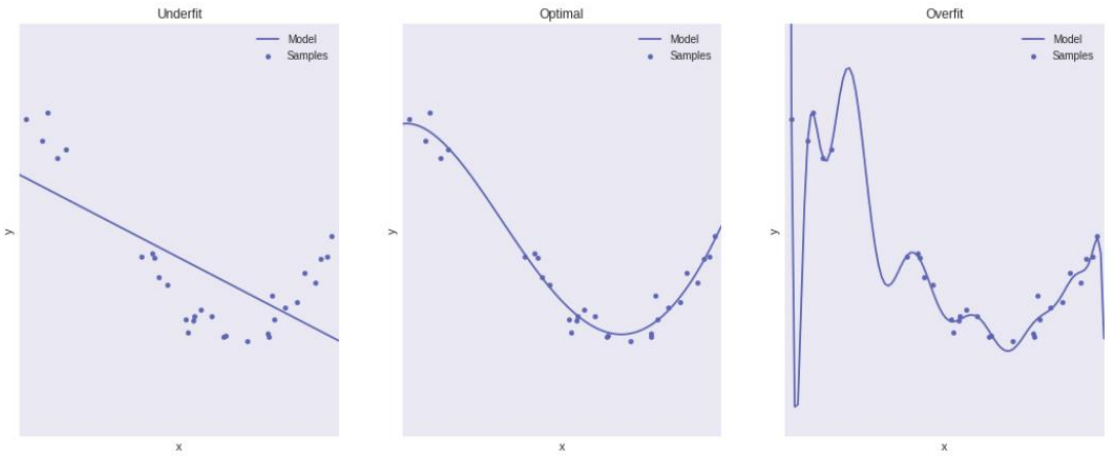


Figure 5. underfit vs optimal vs overfit

2.10.1 Detecting overfitting

The easiest way to detect overfitting is to split the dataset to a training dataset and a test dataset and then train the model with the training dataset; hence, the test dataset is "unseen" for the model at this point. When the model has been trained and tuned, one can evaluate the model using the test dataset and the training dataset, and if the accuracy is much higher on the training dataset than on the test dataset, the model is probably overfitting. (Overfitting in Machine Learning: What It Is and How to Prevent It 2017.)

2.10.2 Preventing overfitting

The most commonly used cross-validation is k-fold. It is a common procedure to split the dataset to 10 groups (k=10) and train as many epochs as there are groups (folds); hence, every fold is given the opportunity to be validation data. This is shown in Figure 5. (Brownlee 2018; Ray 2018; The 5 Levels of Machine Learning Iteration 2017.)

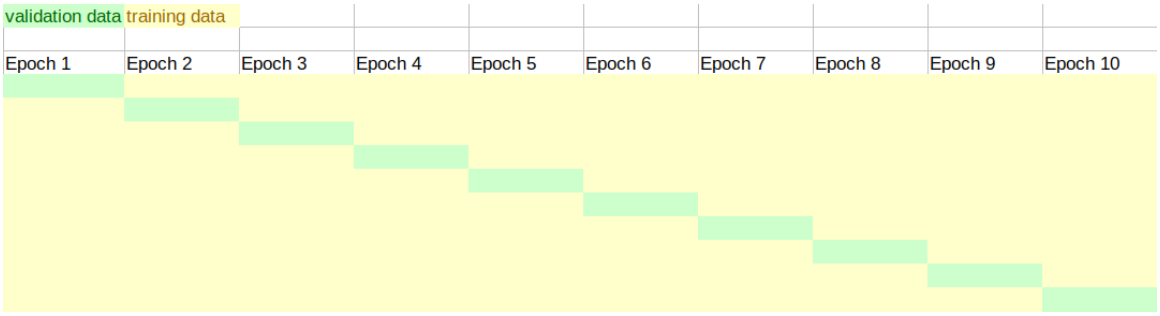


Figure 6. k-fold example

The common steps for k-fold cross-validation are listed as follows:

1. Split the dataset into k groups (k=10 in the example figure x)
2. For each group:
    a. Choose the first group to be used as validation data
    b. Take the remaining groups to be used as training data
    c. Train the model on the training data and evaluate it with the validation data
    d. Save the evaluation score and delete the model
3. Summarize the saved evaluation scores to give the model the total score

2.11 Regressors

2.11.1 Decision tree

Decision trees can be used for both classification and regression. The decision tree is built on smaller subsets of data, where the subset is split using various factors as can be seen in Figure 7. In the decision tree construction, the most important issue is to find an attribute that returns the highest information gain. (Liberman 2017; Sehra 2018.)

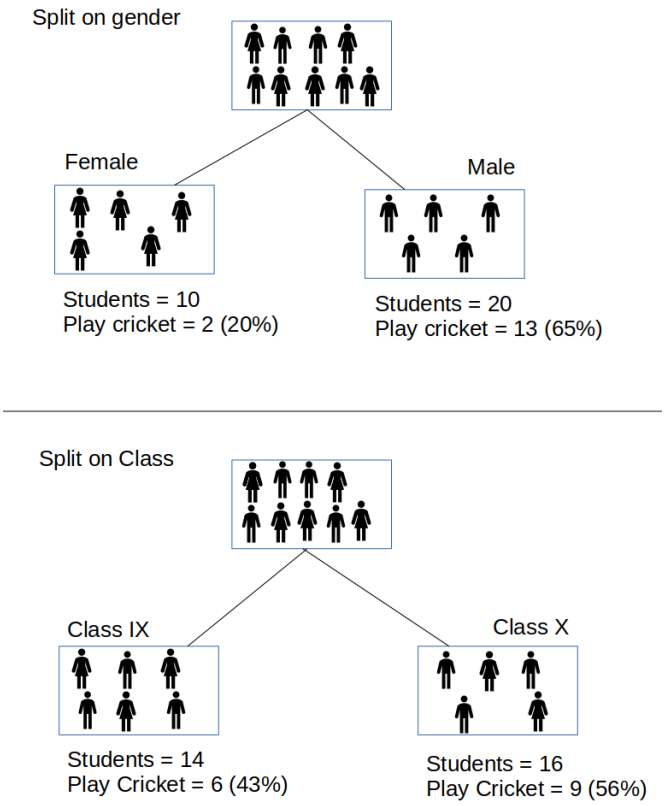


Figure 7. Decision tree example split

2.11.2 Random forest

Random forest is a “forest” of randomly generated decision trees, where every decision tree is used to predict a result and all results are combined to one final result. In this collection of decision trees, all trees do not include all available features but rather use different features in different trees. This prevents the problem where decision tree overfits when too many features are used with too little data. (Liberman 2017.)

2.12 Data quality

No matter how advanced a data analytics tool is used; if data quality is bad, then the results are not reliable. Data quality also affects how much time must be used for preparing and cleaning data. Often in data analytics workflow, 80% of time are used to prepare and clean data. (Ghosh 2018; Redman 2018; Wilder-James 2016.)

Data quality can be estimated by examining several matters. For example, the amount of missing values, amount of misleading values, time series data frequency and how descriptive features the data includes for a certain problem. Sometimes errors on data can be fixed, for example, if there are values “Main St” and “Main Street” where both values mean the same thing, it can be chosen which one to use and change, for example all “Main St” values to “Main Street”. This process is called “data cleaning” or ”preparing data”. (Patel N.d.)

Results can only be as good as data. Data quality is among the most important things when conducting data analysis. If there are values in wrong columns and it would be too much manual work to fix it, it should be made sure not to use those columns with mixed values if there are too many of them. In the example of mixed values, there is numerical and literal data mixed up (See Table 1. Dirty data).

Table 1. Dirty data

Valoisuus	Valsel	Saa	Saasel	Onnpaikka	Onnpaiksel
paljas	kuiva	1	paivanvalo	1	kirkas
marka	4	tie valaistu	2	pilvipouta	1
marka	1	paivanvalo	2	pilvipouta	1
marka	1	paivanvalo	4	vesisade	1
marka	2	hamara	2	pilvipouta	1
3	pimea (valaisematon)	1	kirkas	1	ajorata
kuiva	1	paivanvalo	1	kirkas	1
kuiva	1	paivanvalo	1	kirkas	1
kuiva	1	paivanvalo	1	kirkas	1
kuiva	1	paivanvalo	1	kirkas	1
kuiva	1	paivanvalo	1	kirkas	1
kuiva	4	tie valaistu	2	pilvipouta	1
marka	1	paivanvalo	2	pilvipouta	9
kuiva	1	paivanvalo	2	pilvipouta	1
1	paivanvalo	1	kirkas	9	linja-autopysakki
kuiva	4	tie valaistu	1	kirkas	1
marka	1	paivanvalo	4	vesisade	1
marka	4	tie valaistu	4	vesisade	1
1	paivanvalo	5	lumisade	1	ajorata
kuiva	1	paivanvalo	1	kirkas	1
marka	1	paivanvalo	1	kirkas	1
marka	4	tie valaistu	4	vesisade	1
marka	4	tie valaistu	5	lumisade	1
kuiva	1	paivanvalo	1	kirkas	1
marka	4	tie valaistu	6	rantasade	1
kuiva	1	paivanvalo	1	kirkas	1
3	pimea (valaisematon)	5	lumisade	1	ajorata
kuiva	1	paivanvalo	2	pilvipouta	1
kuiva	1	paivanvalo	1	kirkas	1
marka	4	tie valaistu	4	vesisade	1
marka	1	paivanvalo	4	vesisade	1
kuiva	1	paivanvalo	2	pilvipouta	1
marka	4	tie valaistu	4	vesisade	1
marka	1	paivanvalo	1	kirkas	1
1	paivanvalo	4	vesisade	1	ajorata

3 Combining datasets

Combining datasets is a complicated process where human errors can happen very easily. This chapter presents custom software which makes it easier to combine different kinds of datasets where time is defined (time series data). Problems occur when data needs to be compressed by summing, averaging or other methods and when datasets are so large that they consume all available memory.

3.1 Downloading data

When one is trying to predict something, one needs data to do it. In this thesis, purpose is to predict and analyze traffic incidents, therefore many kinds of different data are needed. All data sources used on this thesis' practical data analysis part, are described in this chapter.

3.1.1 LAM

LAM data is downloaded from: https://aineistot.liikennevirasto.fi/lam/rawdata/ (13.11.2018). The authors' written script is used to download this data. This script has predefined LAM stations that need to be downloaded (116, 126, 145, 146, 147, 148 and 149). All these stations are on road st101 (Ring 1) in Helsinki as depicted in Figure 8.

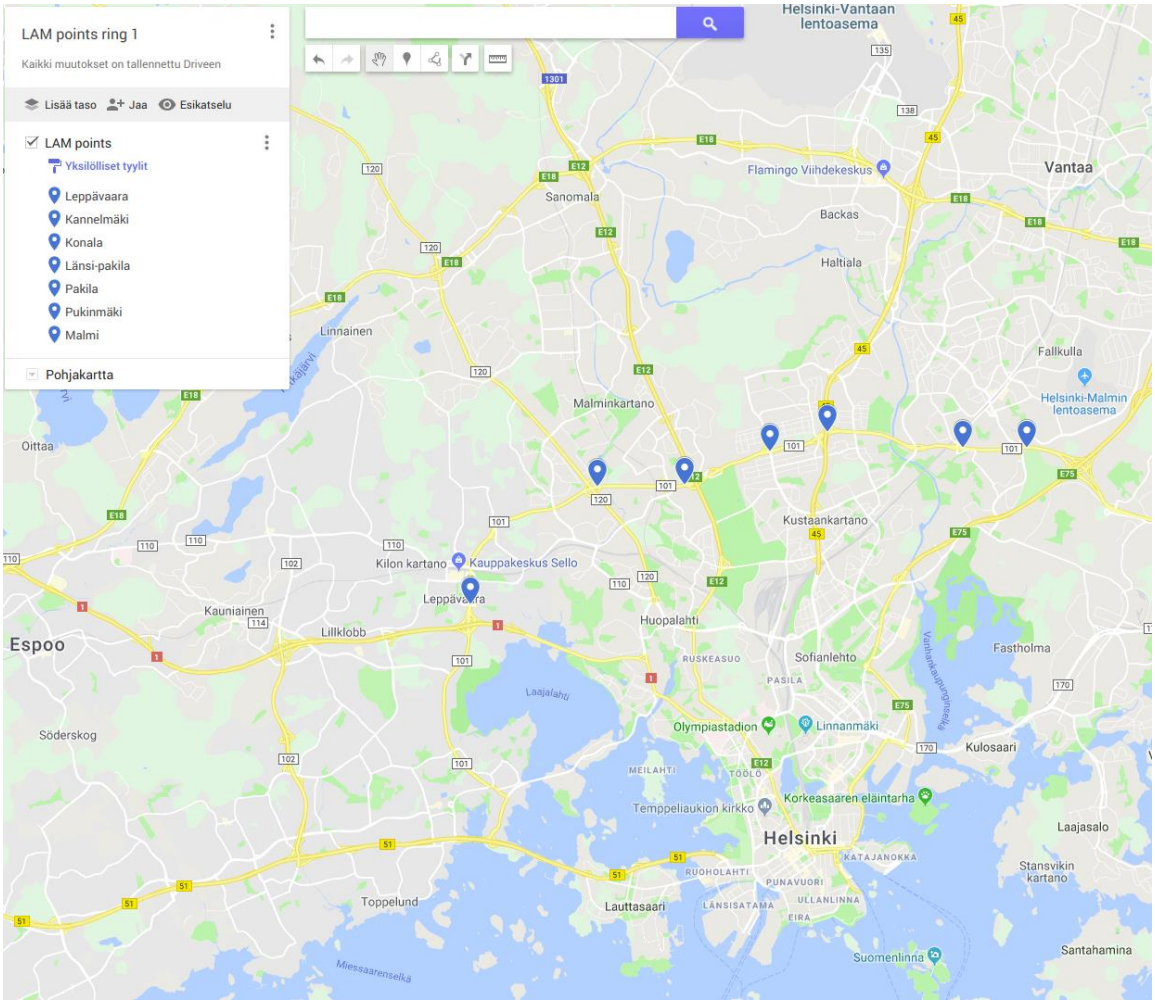


Figure 8. LAM locations

3.1.2 Road weather

Road weather data is downloaded from: https://roadweather.online/data/ (13.11.2018). At the time of acquiring data, there was no authentication; however, at a later date, there appeared a notice that warned about need of authentication in the future.

3.1.3 Tilannehuone (Situation room)

The amount of traffic incident data is downloadable from: www.tilannehuone.fi (13.11.2018). Data is downloaded for the City of Helsinki, not only for Ring 1, because there would be too little amount of data and it is hard to identify the exact location of traffic incident by the given information.

3.1.4 Traffic incidents

Information about traffic incidents is downloaded from avoindata.fi: https://www.avoindata.fi/data/fi/dataset/tieliikenneonnettomuudet (13.11.2018). This includes useful information about traffic incidents including the amount of deaths and injuries per day.

3.1.5 Finnish Meteorological Institute

Weather data is downloaded from open data API of the Finnish Meteorological Institute: https://ilmatieteenlaitos.fi/havaintojen-lataus#!/ (13.11.2018). The station measuring this data is located at Helsinki-Vantaa airport near LAM points (See Figure 9. Weather station location).

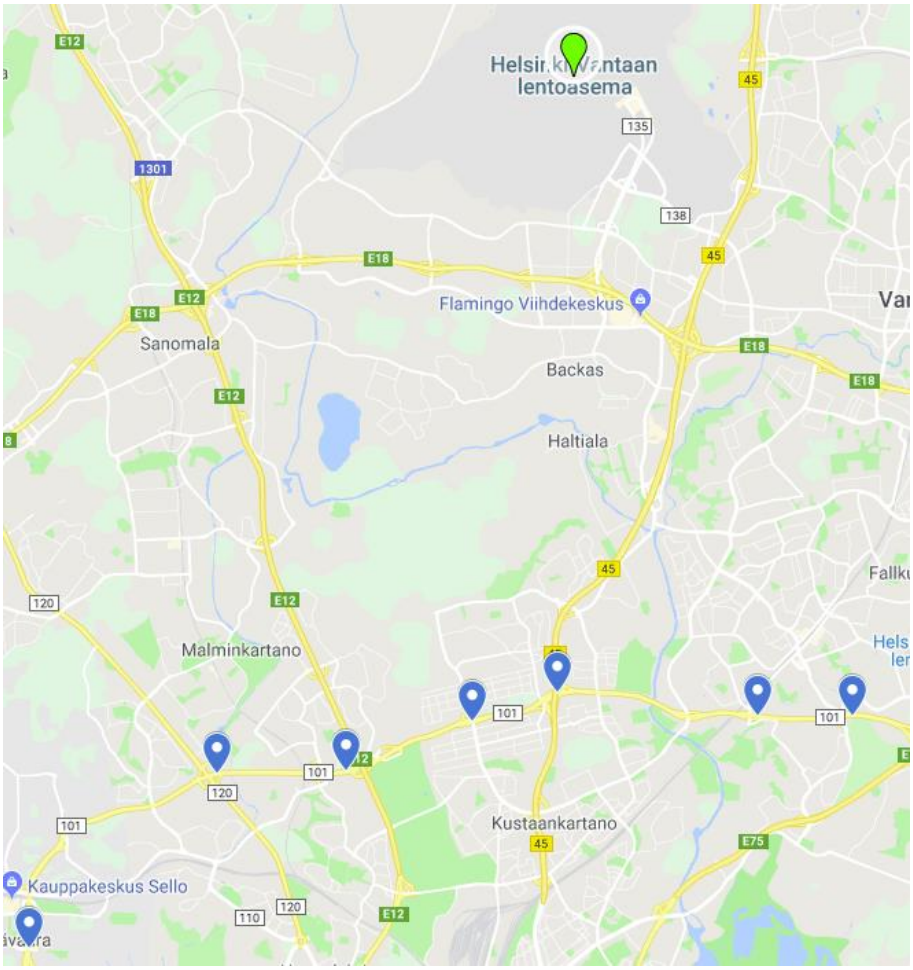


Figure 9. Weather station location

3.1.6 Sunrise and sunset

Sunrise and sunset data is downloaded for latitude `60.230987` and longitude `24.849928` from: https://sunrise-sunset.org/ (26.11.2018). Those coordinates are in Helsinki; therefore, geolocation with other datasets matches.

3.2 Combining data

Combining data from different datasets is challenging. It takes plenty of time to write a script to handle every dataset independently with so much manual work. The author has written a generic program that can handle different kind of datasets by using user given parameters. This program can add a timestamp column to a single dataset using date column and combine many datasets to one by timestamps with defined time precision (year, month, day, hour, minute or second). The program help

page gives information about options and how to combine the datasets and add timestamps to the datasets (See Figure 10. Dataset tool help).

```
usage: dataset_tool.py [-h] [-t] [-df DATESTRING_FORMAT] [-dc DATE_COLUMN]
                       [-c CSV_PATHS] [-p PREFIXES] [-tp TIME_PRECISION] [-ff]
                       [-o OUTPUT]

Dataset tool to add timestamp column to csv by date or combining multiple
datasets by timestamp.

optional arguments:
  -h, --help            show this help message and exit
  -t                    Add this option to add timestamp column to single
                        dataset
  -df DATESTRING_FORMAT, --datestring_format DATESTRING_FORMAT
                        Date string format to parse date column as timestamp.
                        Example: %d-%m-%Y
  -dc DATE_COLUMN, --date_column DATE_COLUMN
                        Name of date column from which to parse timestamp
  -c CSV_PATHS, --csv_paths CSV_PATHS
                        Paths to csv files in the following format:
                        path1,path2,path3...
  -p PREFIXES, --prefixes PREFIXES
                        Prefixes for every provided csv file columns in the
                        following format: prefix1,prefix2,prefix3...
  -tp TIME_PRECISION, --time_precision TIME_PRECISION
                        Time precision to combine datasets. Options: 'year',
                        'month', 'day', 'hour', 'minute' and 'second'
  -ff, --ffill          Toggles forward fill on NaN values on/off. Default:
                        off.
  -o OUTPUT, --output OUTPUT
                        Output file path. If adding timestamp to dataset,
                        multiple paths required in the following format:
                        path1,path2,path3...
```

Figure 10. Dataset tool help

3.2.1 Adding timestamp to dataset

By adding timestamp to dataset, datasets can be combined by time. One needs to know which string format to use for the Date column. The correct string format to use here is `%Y-%m-%d` and the most precise information should always be used. For example, it would not be wise to use here the string format `%Y-%m`, because then it would return timestamps in month precision only instead of day precision. The example file is stock data `GSPC.csv` (See Table 2. Example data GSPC.csv).

Table 2. Example data GSPC.csv

Date	Open	High	Low	Close	Adj Close	Volume
2017-10-23	2578.080078	2578.290039	2564.330078	2564.97998	2564.97998	3211710000
2017-10-24	2568.659912	2572.179932	2565.580078	2569.129883	2569.129883	3427330000
2017-10-25	2566.52002	2567.399902	2544	2557.149902	2557.149902	3874510000
2017-10-26	2560.080078	2567.070068	2559.800049	2560.399902	2560.399902	3869050000
2017-10-27	2570.26001	2582.97998	2565.939941	2581.070068	2581.070068	3887110000
2017-10-30	2577.75	2580.030029	2568.25	2572.830078	2572.830078	3658870000
2017-10-31	2575.98999	2578.290039	2572.149902	2575.26001	2575.26001	3827230000
2017-11-01	2583.209961	2588.399902	2574.919922	2579.360107	2579.360107	3813180000
2017-11-02	2579.459961	2581.110107	2566.169922	2579.850098	2579.850098	4048270000
2017-11-03	2581.929932	2588.419922	2576.77002	2587.840088	2587.840088	3567710000
2017-11-06	2587.469971	2593.379883	2585.659912	2591.129883	2591.129883	3539080000
2017-11-07	2592.110107	2597.02002	2584.350098	2590.639893	2590.639893	3809650000
2017-11-08	2588.709961	2595.469971	2585.02002	2594.379883	2594.379883	3899360000
2017-11-09	2584	2586.5	2566.330078	2584.620117	2584.620117	3831610000
...	...	...	...	...	...	...

The tool can be used from command line interface or it can be imported to one's own Python code as library. From command line, it is easy to add timestamp to CSV files (See Figure 11. Add timestamp command).

```
$ python3 dataset_tool.py -t -c ../dataset/GSPC.csv -o
../dataset/GSPC_timestamp.csv -dc Date -df %Y-%m-%d
Reading file: ../dataset/GSPC.csv, writing to:
../dataset/GSPC_timestamp.csv
```

Figure 11. Add timestamp command

After using the dataset tool to add the timestamp to the dataset, one has timestamps added to the output CSV file, and those timestamps are Unix timestamps calculated from dates in the `Date` column (See Table 3. Example data GSPC.csv with timestamp).

Table 3. Example data GSPC.csv with timestamp

Date	Open	High	Low	Close	Adj Close	Volume	timestamp
2017-10-23	2578.080078	2578.290039	2564.330078	2564.97998	2564.97998	3211710000	1508706000
2017-10-24	2568.659912	2572.179932	2565.580078	2569.129883	2569.129883	3427330000	1508792400
2017-10-25	2566.52002	2567.399902	2544	2557.149902	2557.149902	3874510000	1508878800
2017-10-26	2560.080078	2567.070068	2559.800049	2560.399902	2560.399902	3869050000	1508965200
2017-10-27	2570.26001	2582.97998	2565.939941	2581.070068	2581.070068	3887110000	1509051600
2017-10-30	2577.75	2580.030029	2568.25	2572.830078	2572.830078	3658870000	1509314400
2017-10-31	2575.98999	2578.290039	2572.149902	2575.26001	2575.26001	3827230000	1509400800
2017-11-01	2583.209961	2588.399902	2574.919922	2579.360107	2579.360107	3813180000	1509487200
2017-11-02	2579.459961	2581.110107	2566.169922	2579.850098	2579.850098	4048270000	1509573600
2017-11-03	2581.929932	2588.419922	2576.77002	2587.840088	2587.840088	3567710000	1509660000
2017-11-06	2587.469971	2593.379883	2585.659912	2591.129883	2591.129883	3539080000	1509919200
2017-11-07	2592.110107	2597.02002	2584.350098	2590.639893	2590.639893	3809650000	1510005600
2017-11-08	2588.709961	2595.469971	2585.02002	2594.379883	2594.379883	3899360000	1510092000
2017-11-09	2584	2586.5	2566.330078	2584.620117	2584.620117	3831610000	1510178400
...	...	...	...	...	...	...	...

The code that adds the timestamp to Pandas dataframe can be used without defining the date column which to convert into timestamps. If this date column is not defined, it starts looping columns and tries to convert them to timestamps. If it successfully converts the tested column to timestamps, the tested column will be used as date column. In addition, if the date string format is not defined, Pandas `to_datetime` function tries to find the right format automatically (See Figure 12. Timestamp adding function code). If one wants to use as short a command as possible to add timestamp to CSV file, only `-t` option, input and output paths are needed (See Figure 13. Shortest timestamp adding command). `-t` option is added to indicate to the program that timestamp adding operation is to be performed.

```
def add_timestamp_column_to_csv(path, date_column=None,
timestamp_column="timestamp", datestring_format=None):
    """
    datestring_format %d.%m.%Y parses for example date 26.01.2017
    """

    try:
        df = pd.read_csv(path)
    except UnicodeDecodeError:
        df = pd.read_csv(path, encoding="latin-1")

    if not date_column:
```

```
        # Try to find date column if not defined
        success = False
        for col in df.columns:
            try:
                df[timestamp_column] = df[col].apply(lambda x:
pd.to_datetime(x, format=datestring_format).timestamp())
                print("Used column '{}' to create
timestamps.".format(col))
                success = True
                break
            except:
                continue

        if not success:
            print("Could not find date column.")
            exit()

        return df

    df[timestamp_column] = df[date_column].apply(lambda x:
pd.to_datetime(x, format=datestring_format).timestamp())
    return df
```

Figure 12. Timestamp adding function code

```
$ python3 dataset_tool.py -t -c ../dataset/GSPC.csv -o
../dataset/GSPC_timestamp.csv
```

Figure 13. Shortest timestamp adding command

3.2.2 Dataset combining methods

When timestamps have been added to few datasets, all those datasets can be combined by time. There are different options offered by this tool, for example, time accuracy can be chosen: `year`, `month`, `day`, `hour`, `minute` and `second`. If using, for example, time precision `hour`, then all datapoints in certain feature inside the same hour will be combined by calculating the average of those values by default. There are also other options in data combining. This is only when one value needs to be made from many values; however, if there is no value at all, then `forward fill` can be used to use the value from the second most precise time precision available. For example, if value is needed for every hour in a day (24 hours) and only one value is available for one day, then this one value is used for all hours in the day (See Table 4. Forward fill example).

Table 4. Forward fill example

	A	B	C	D	E	F
1	Only one variable for one day			One variable filled for whole day		
2	Date	Value		Date	Value	
3	2017010100	5		2017010100	5	
4	2017010101			2017010101	5	
5	2017010102			2017010102	5	
6	2017010103			2017010103	5	
7	2017010104			2017010104	5	
8	2017010105			2017010105	5	
9	2017010106			2017010106	5	
10	2017010107			2017010107	5	
11	2017010108			2017010108	5	
12	2017010109			2017010109	5	
13	2017010110			2017010110	5	
14	2017010111			2017010111	5	
15	2017010112			2017010112	5	
16	2017010113			2017010113	5	
17	2017010114			2017010114	5	
18	2017010115			2017010115	5	
19	2017010116			2017010116	5	
20	2017010117			2017010117	5	
21	2017010118			2017010118	5	
22	2017010119			2017010119	5	
23	2017010120			2017010120	5	
24	2017010121			2017010121	5	
25	2017010122			2017010122	5	
26	2017010123			2017010123	5	
27	2017010124			2017010124	5	
28	2017010200	8		2017010200	8	
29	2017010201			2017010201	8	
30	2017010202			2017010202	8	
31	2017010203			2017010203	8	
32	2017010204			2017010204	8	
33	2017010205			2017010205	8	
34	2017010206			2017010206	8	
35	2017010207			2017010207	8	
36	2017010208			2017010208	8	
37	2017010209			2017010209	8	
38	2017010210			2017010210	8	
39	2017010211			2017010211	8	
40	2017010212			2017010212	8	
41	2017010213			2017010213	8	
42	2017010214			2017010214	8	
43	2017010215			2017010215	8	
44	2017010216			2017010216	8	
45	2017010217			2017010217	8	
46	2017010218			2017010218	8	
47	2017010219			2017010219	8	
48	2017010220			2017010220	8	
49	2017010221			2017010221	8	
50	2017010222			2017010222	8	
51	2017010223			2017010223	8	
52	2017010224			2017010224	8	

Forward fill can be conducted with one Pandas class DataFrame method `fillna()`, where the method needs to be defined as `ffill`. Example usage of `forward fill` in Python is as follows: `df = df.fillna(method='ffill')`.

For traffic incidents analysis dataset, data that has been downloaded in chapter 3.1 needs to be combined. There is a great deal of different data, many of them have different time precisions, for example traffic incidents are reported for every day but weather information can be very accurate, even for every second. With this combining tool, those datasets can be combined to one dataset that has hour accuracy by

filling the traffic incident data that is reported only per day and calculating the averages or sum from weather data and other data (See Figure 14. Combiner usage).

```
Combiner(csv_paths=["../dataset/lam_dataset.csv",
                    "../dataset/roadweather_dataset.csv",
                    "../dataset/helsinki-vantaa.csv",
                    "../dataset/tieliikenneonnettomuudet_2017_onnettomuus.csv",
                    "../dataset/tieliikenneonnettomuudet_2017_osallinen.csv",
                    "../dataset/tieliikenneonnettomuudet_2017_hlo.csv",
                    "../dataset/tilannehuone_count_20170101-20180101.csv",
                    "../dataset/sunrise_sunset2017.csv"],
        prefixes=["lam", "rw", "fmihv", "tloon", "tloos", "tlohl", "th", "ss"], time_precision="hour",
        output_path="../dataset/combined_hour.csv",
        combine_methods={"Kuolleet": "sum", "Loukkaant": "sum", "speed": "sum"}).combine_by_timestamp_mem_eff()
```

Figure 14. Combiner usage

3.2.3 Combining method options

Mean is the default combining method. When many values need to be compressed into one value, the average value is calculated (See Table 5. Mean method).

Table 5. Mean method

Hour data			Day data	
time	var1		time	var1
2017010100	5		20170101	5.285714286
2017010101	6			
2017010102	6			
2017010103	5			
2017010104	4			
2017010105	3			
2017010106	8			
2017010107	6			
2017010108	7			
2017010109	8			
2017010110	3			
2017010111	5			
2017010112	2			
2017010113	6			

Sum method is simple as it just sums all values together into one compressed value (See Table 6. Sum method).

Table 6. Sum method

Hour data			Day data	
time	var1		time	var1
2017010100	5		20170101	74
2017010101	6			
2017010102	6			
2017010103	5			
2017010104	4			
2017010105	3			
2017010106	8			
2017010107	6			
2017010108	7			
2017010109	8			
2017010110	3			
2017010111	5			
2017010112	2			
2017010113	6			

Datapoints method counts the number of datapoints. For example, if there is LAM data with traffic speeds per one vehicle, average speed of all vehicles and additionally, amount of traffic will be gained (See Table 7. Datapoints method).

Table 7. Datapoints method

Hour data			Day data		
time	var1		time	var1	var1_datapoints
2017010100	5		20170101	5.285714286	14
2017010101	6				
2017010102	6				
2017010103	5				
2017010104	4				
2017010105	3				
2017010106	8				
2017010107	6				
2017010108	7				
2017010109	8				
2017010110	3				
2017010111	5				
2017010112	2				
2017010113	6				

Min method selects the smallest value (See Table 8. Min method).

Table 8. Min method

Hour data			Day data	
time	var1		time	var1
2017010100	5		20170101	2
2017010101	6			
2017010102	6			
2017010103	5			
2017010104	4			
2017010105	3			
2017010106	8			
2017010107	6			
2017010108	7			
2017010109	8			
2017010110	3			
2017010111	5			
2017010112	2			
2017010113	6			

Max method selects the biggest value (See Table 9. Max method).

Table 9. Max method

Hour data			Day data	
time	var1		time	var1
2017010100	5		20170101	8
2017010101	6			
2017010102	6			
2017010103	5			
2017010104	4			
2017010105	3			
2017010106	8			
2017010107	6			
2017010108	7			
2017010109	8			
2017010110	3			
2017010111	5			
2017010112	2			
2017010113	6			

Standard deviation method calculates standard deviation from datapoints in the selected time accuracy (See Table 10. Standard deviation method).

Table 10. Standard deviation method

Hour data			Day data	
time	var1		time	var1
2017010100	5		20170101	1.815682598
2017010101	6			
2017010102	6			
2017010103	5			
2017010104	4			
2017010105	3			
2017010106	8			
2017010107	6			
2017010108	7			
2017010109	8			
2017010110	3			
2017010111	5			
2017010112	2			
2017010113	6			

Categorical method can be used on columns that do not have numerical data. It does `one hot encoding` on the selected column, where it creates as many new columns as there are different values (categories) in a column. Only one column of new columns will have value “1”. For cases where there are many different values for the specified time accuracy, the most common value will be selected (See Table 11. Categorical method).

Table 11. Categorical method

Hour data			Day data			
time	var1		time	categorical_var1_bus	categorical_var1_car	categorical_var1_motorcycle
2017010100	bus		20170101	1	0	0
2017010101	bus					
2017010102	car					
2017010103	motorcycle					
2017010104	car					
2017010105	car					
2017010106	motorcycle					
2017010107	bus					
2017010108	motorcycle					
2017010109	bus					
2017010110	bus					
2017010111	car					
2017010112	bus					
2017010113	bus					

Categorical_sum method is almost the same as `categorical` method, however, it does not do `one hot encoding`. It sums all values in a column and inserts them in `categoricalsum` columns (See Table 12. Categorical_sum method).

Table 12. Categorical_sum method

Hour data			Day data			
time	var1		time	categoricalsum_var1_bus	categoricalsum_var1_car	categoricalsum_var1_motorcycle
2017010100	bus		20170101	7	4	3
2017010101	bus					
2017010102	car					
2017010103	motorcycle					
2017010104	car					
2017010105	car					
2017010106	motorcycle					
2017010107	bus					
2017010108	motorcycle					
2017010109	bus					
2017010110	bus					
2017010111	car					
2017010112	bus					
2017010113	bus					

All those methods are defined in one Python class, from which they can be used. In the Figure 15 there is the implemented code that does have functions for each method. Each function behaves similarly and has similar output where there is dictionary data structure used and depending on methods that is used, different key is used in the dictionary and wanted value is calculated for that key. By inserting Pandas dataframes one by one for certain time frame, one can calculate wanted values for certain feature and time frame in the dataframe, and use received dictionaries to collect values for that certain time frame and append CSV file one time frame at a time. In the combining methods code implementation in Figure 15, only non-categorical methods are included, because categorical method implementations have some bugs at the time of writing this thesis.

```
class CombiningMethods:

    def __init__(self, categorical_columns, categorical_sum_columns,
combine_methods):
        self.categorical_columns = categorical_columns
        self.categorical_sum_columns = categorical_sum_columns
        self.combine_methods = combine_methods

        self.value_dict = {}


    def combine_mean(self, df, column, id_column):
        value = df[column].apply(float).sum() / len(df[column].dropna())
        self.value_dict[id_column + "_mean"] = value
        return {id_column + "_mean": value}


    def combine_sum(self, df, column, id_column):
        value = df[column].apply(float).sum()
        self.value_dict[id_column + "_sum"] = value
```

```
        return {id_column + "_sum": value}


    def combine_min(self, df, column, id_column):
        value = df[column].apply(float).min()
        self.value_dict[id_column + "_min"] = value
        return {id_column + "_min": value}


    def combine_max(self, df, column, id_column):
        value = df[column].apply(float).max()
        self.value_dict[id_column + "_max"] = value
        return {id_column + "_max": value}


    def combine_std(self, df, column, id_column):
        value = df[column].apply(float).std()
        self.value_dict[id_column + "_std"] = value
        return {id_column + "_std": value}


    def combine_datapoints(self, df, column, id_column):
        datapoints_id_column = id_column = id_column + "_" +
"datapoints"
        datapoints_value = len(df[column].dropna().values)
        self.value_dict[datapoints_id_column] = datapoints_value
        return {datapoints_id_column: datapoints_value}
```

Figure 15. Combining methods code

3.2.4 Memory efficiency

The largest dataset of the used datasets is `lam_dataset.csv` (5.4 GB) and the second largest is `roadweather_dataset.csv` (1.6 GB) for one year of data. Combining many large datasets consumes a great amount of memory. This became a problem, because 16 GB ram is not nearly enough for a system to handle this much data concurrently when new data needs to be formed by calculating and arranging data to a new dataset. Memory efficient function is implemented to perform dataset combining by creating temporary files. Those temporary files include data for one dataset and for the defined time accuracy, so one temporary file includes data for one final datapoint for every feature in dataset. These final datapoints are then calculated by using selected combining method for all those values in one temporary file.

In the temporary file for time accuracy hour 18, day 17, month 01 and year 2017 for dataset `lam_dataset.csv` we have 35000~ rows of values that includes latitude,

longitude and speed (See Table 13. lam_dataset.csv example rows). Coordinates are not that relevant in this case; however, speed could be useful. By calculating the average of speed from those 35000~ values the average speed of vehicles is gained for every hour. Because of the large amount of data, single speedings are lost in mass, and the average speeds do not variate very much. Nevertheless, when calculating sum, something that tells the sum of amount of traffic and total speed is gained.

Table 13. lam_dataset.csv example rows

	A	B	C	D
1	latitude	longitude	timestamp	speed
2	60.24	24.99	2017111718	70
3	60.24	24.99	2017111718	84
4	60.24	24.99	2017111718	71
5	60.24	24.99	2017111718	86
6	60.24	24.99	2017111718	71
7	60.24	24.99	2017111718	83
8	60.24	24.99	2017111718	89

When this large amount of data has been “compressed”, where for example one row represents 35000~ rows, all “compressed” and calculated data can be loaded to memory for sorting and saving into one final dataset.

The code creating the temporary files creates a stream from CSV file, where it loads smaller parts of CSV to memory at once, so it will not eat up all memory. It sorts those smaller parts of CSV to smaller CSV files, where one smaller CSV file contains data for defined time accuracy. (See Figure 16. Temporary file creating code)

```
def _create_temp_files_by_time_accuracy(self):

    for path in tqdm(self.csv_paths, desc="Creating temporary files"):
        df = pd.read_csv(path, chunksize=100000)

        dataset_filename = path.split(os.path.sep)[-1]

        for part_df in df:

            if len(part_df.index) == 0:
                print("Empty dataframe. Skipping: {}".format(path))
                continue

            part_df[self.timestamp_column_name] =
part_df[self.timestamp_column_name].apply(self.timestamp_to_datestring)
```

```
            region = ""
            for region, df_region in
part_df.groupby(self.timestamp_column_name):
                time_accuracy_temp_path = os.path.join(self.temp_dir,
region + "_" + dataset_filename)

                if os.path.exists(time_accuracy_temp_path):
                    existing_columns =
pd.read_csv(time_accuracy_temp_path).columns
                    with open(time_accuracy_temp_path, "a") as f:
                        df_region[existing_columns].to_csv(f,
header=False, index=False)
                else:
                    df_region.to_csv(time_accuracy_temp_path,
index=False)
```

Figure 16. Temporary file creating code

In the process of combining 15.2 GB of data, where the biggest dataset was size of 12 GB, the highest memory usage from dataset combiner program was 112 MB. Without memory efficient dataset combining function, the highest memory usage would be tens of gigabytes, and would require a powerful computer with plenty of memory.

4 Regression (predicting continual variable)

4.1 Dataset

In this chapter, there is used one-year worth of combined data (2017), where for first dataset most frequent feature is datapoint per hour and for other per day, so two frequencies in datasets were used. In some features there are very high frequency data (for example, 1 per minute), so features with more frequent data than day or hour are compressed with data combining methods that are described in chapter 3.2.3. In first, more frequent dataset, which has datapoint per hour, target features that one wants to predict must be forward filled, because those features had datapoint only for day. Forward fill is done by using one day's datapoint on every hour of the same day, so it could be possible to extract more information out of those features that had more frequent data at the beginning.

4.2 Feature engineering

Feature engineering is needed in data-analysis to help models to gain a better understanding of data. For example, here one has the day length presented in format `hour:minute:second` and by converting it to the most precise time unit it has, one gets integer number of seconds, which is easier for neural network to understand, because `0` will be no light in a day at all and `10000` will be `2 hours` and `46 minutes` of light in a day.

While some data must be compressed into one data point because there is data for day time precision, it is best to get as much metrics from one feature as possible, because from e.g. 35000 data points compressed into 1, a great amount of describing features may be lost. Here all combining methods of dataset combiner are utilized to gain as much insight to a single data point as possible, so one feature with precise time accuracy (as frequent as value per second) is compressed to time precision of day, and one very frequent column of data is now six columns of day precision data. These six columns or features are metrics from the original very frequent data. In this case, there is the feature `air temperature`, and it could be interesting to know the minimum and maximum temperatures for the day, where only the mean temperature would not provide enough information. Low enough minimum temperature might cause road surface to freeze, make road surface slippery and thereafter cause traffic incidents because of unpredicted slipperiness of road surface. (Table 14: Compressed feature(s)).

Table 14: Compressed feature(s)

timestamp	fmihvIlman lämpötila (degC)_datapoints	fmihvIlman lämpötila (degC)_max	fmihvIlman lämpötila (degC)_mean	fmihvIlman lämpötila (degC)_min	fmihvIlman lämpötila (degC)_std	fmihvIlman lämpötila (degC)_sum
20150102	144.0	5.4	3.482638888888889	2.4	0.6785762409933868	501.5
20150103	144.0	2.5	0.9784722222222224	-0.9	0.9098653703872854	140.9
20150104	144.0	-0.9	-2.629166666666667	-5.4	1.1852753361044943	-378.6

20150105	144.0	-5.5	-10.692361111111113	-14.7	2.5547454777530096	-1539.7000000000005
20150106	144.0	-5.5	-13.679861111111109	-16.7	2.6936490927959484	-1969.9
20150107	144.0	-2.4	-3.3673611111111112	-5.5	0.9369516008062008	-484.9

4.3 Feature selection

4.3.1 Fit line between two variables

By manually plotting two variables, one in x-axis and another on y-axis, one can see how one variable affects another one. With LOWESS smoothing regression line one can see here that with increasing mean air temperature the amount of deaths caused by traffic incidents also increases (See Figure 17. Air temperature vs amount of deaths). Similarities can also be seen when comparing mean air temperature with the amount of injuries (See Figure 18. Air temperature vs amount of injuries) and amount of incidents (See Figure 19. Air temperature vs amount of incidents).

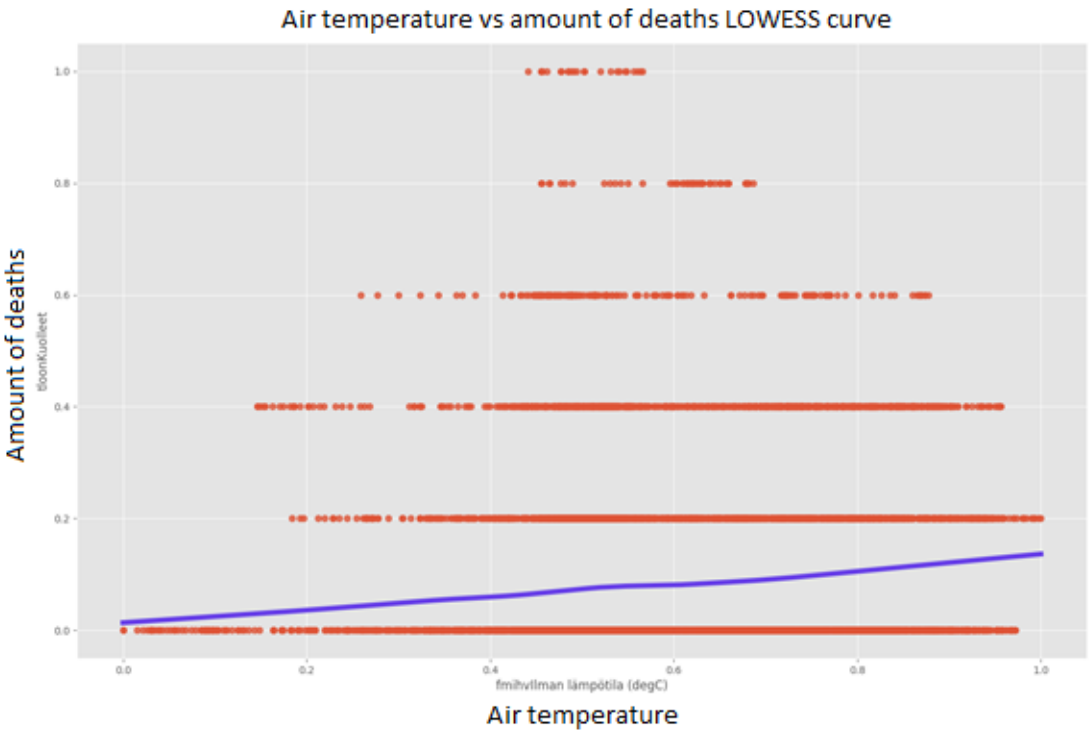


Figure 17. Air temperature vs amount of deaths

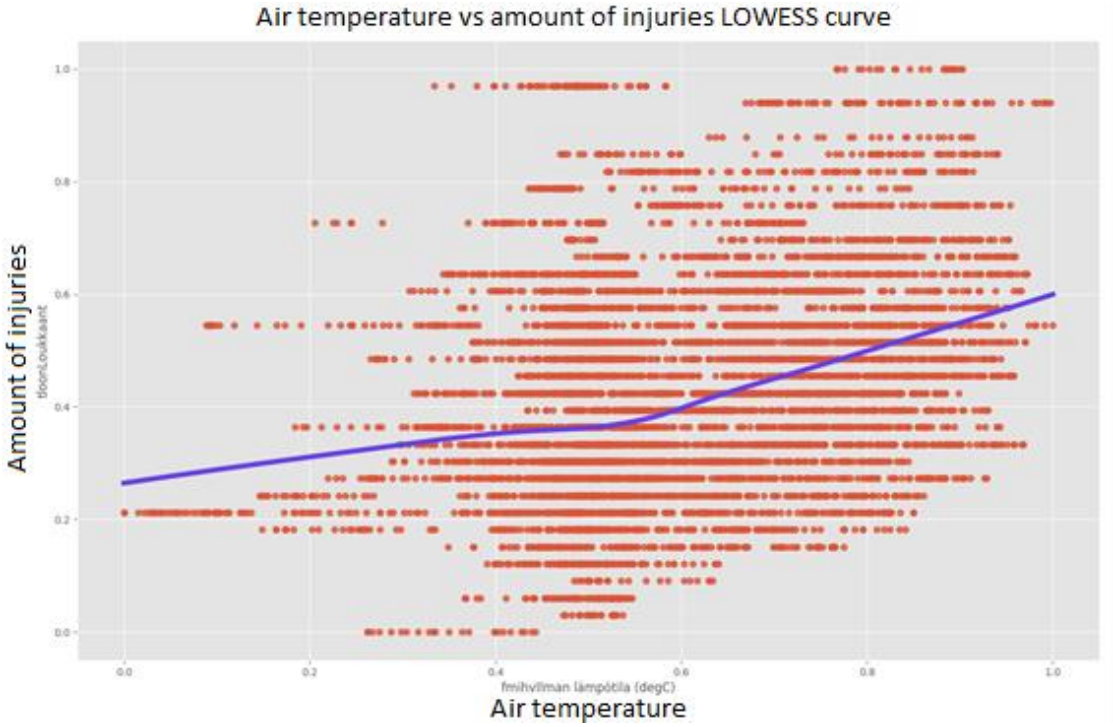


Figure 18. Air temperature vs amount of injuries

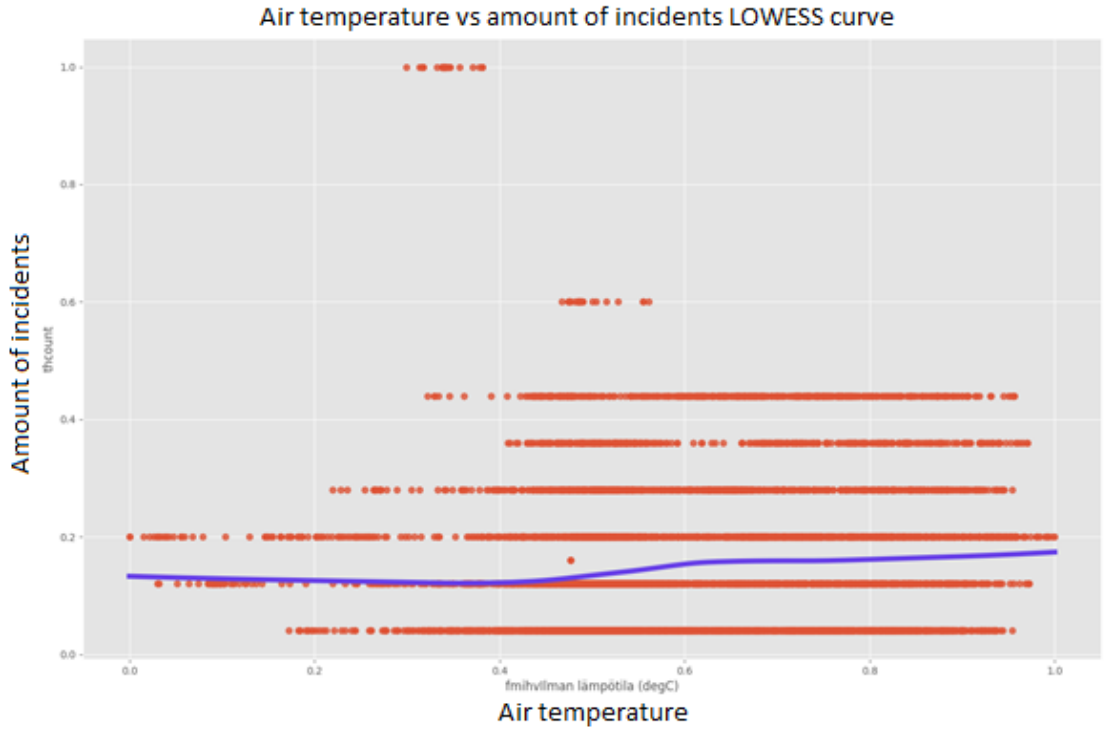


Figure 19. Air temperature vs amount of incidents

These three variables are to be predicted in traffic incidents: the amount of deaths, amount of injuries and amount of traffic incidents. Seems like the mean air temperature would be suitable as one variable to predict from. Another variable that would have somewhat the same effect is the mean road surface friction, because generally

with increasing air temperature the road surface will become dry and friction will increase. This correlation can be seen between the air temperature and road surface friction by plotting those variables and using LOWESS smoothing (See Figure 20. Air temperature vs road surface friction).

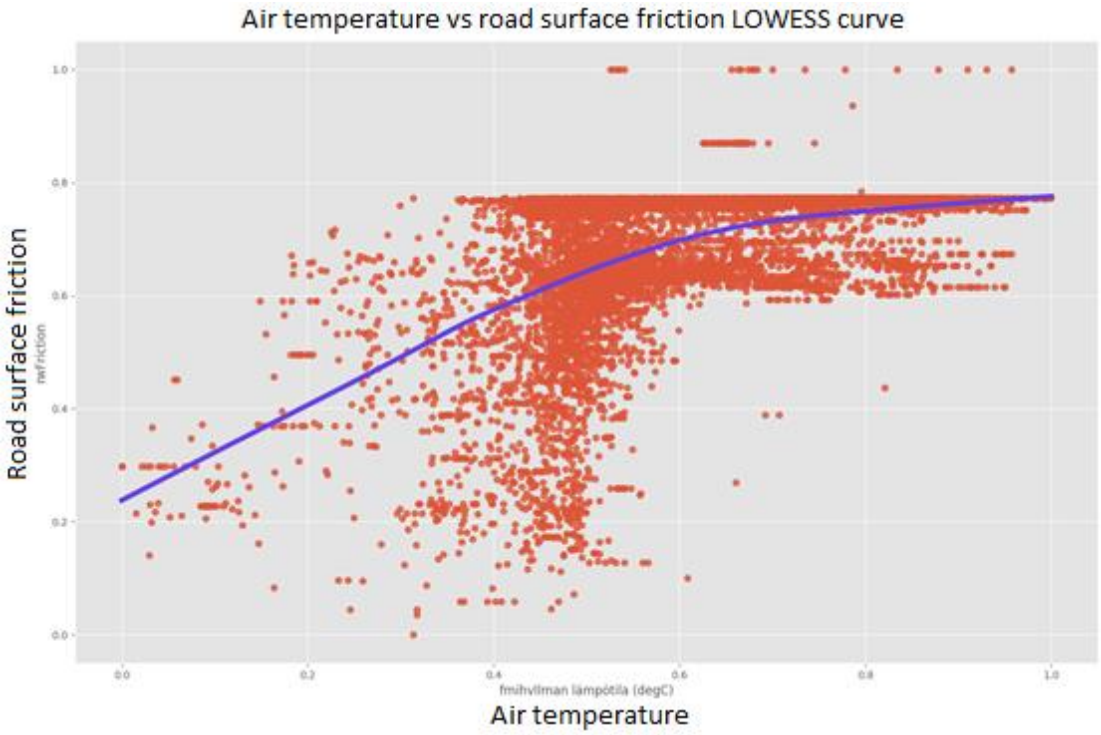


Figure 20. Air temperature vs road surface friction

Since increasing friction and air temperature mean that there are more deaths in traffic incidents, this means that more people die in traffic incidents in summer than in winter. This can be seen by comparing the amount of deaths during months. This shows that from the beginning of the year, the amount of deaths starts slowly to increase towards summer and after summer, the amount of deaths starts to slowly decrease (See Figure 21. Months vs amount of deaths).

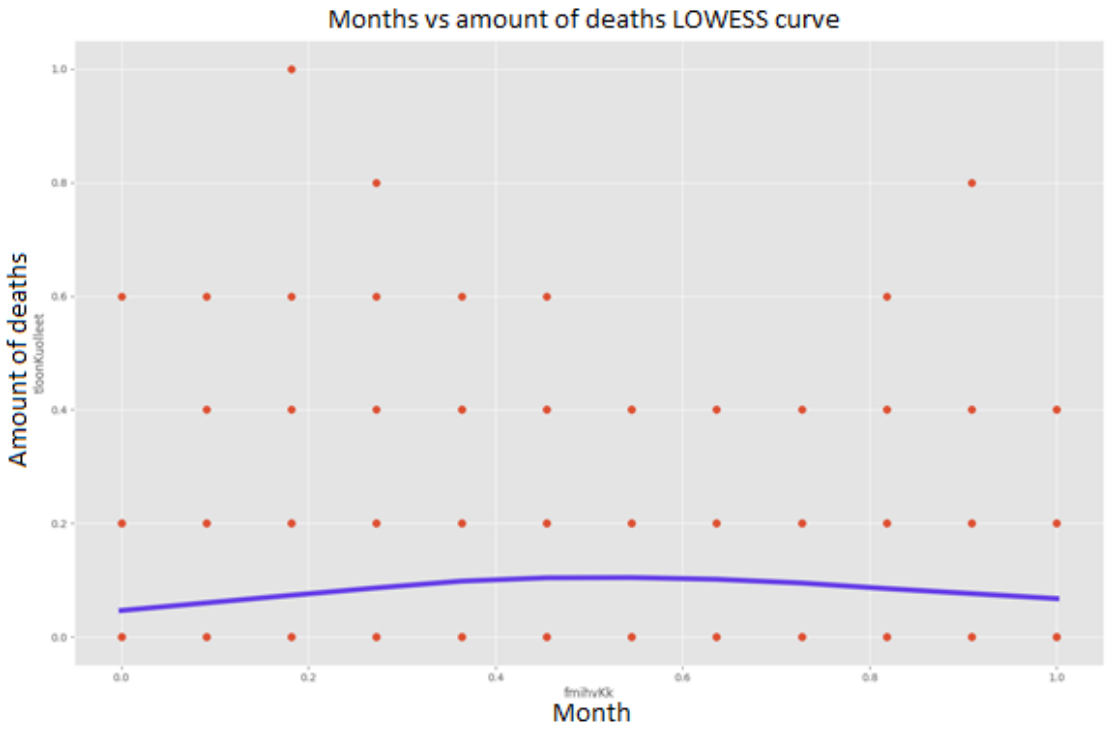


Figure 21. Months vs amount of deaths

4.3.2 Feature importance

With tree models, one can evaluate the importance of features when we want to predict certain variables. When using algorithmic feature importance/selection methods, one needs to remember that if there are no descriptive features at all, there is a chance for misleading results, and one should be sceptic when examining feature importance results that are provided by algorithm.

4.4 Neural networks

For LSTM model data needs to be prepared in a way that it has time series data from which to predict the output variables. For dataset where there is hour data, one can e.g. define the time window of 8, then there are 8 hours of data from which to predict next hour. It is useful to use LSTM model when there are features that can be based on other features in a certain time window; however, not instantly. For example, if air temperature is low and humidity is high at night, then the chances are that in the morning the road is slippery, and because the road is slippery, traffic incidents

are more likely to happen. In the example of input and output data, red color indicates input values from which to predict the green colored output values (See Table 15. LSTM data example).

Table 15. LSTM data example

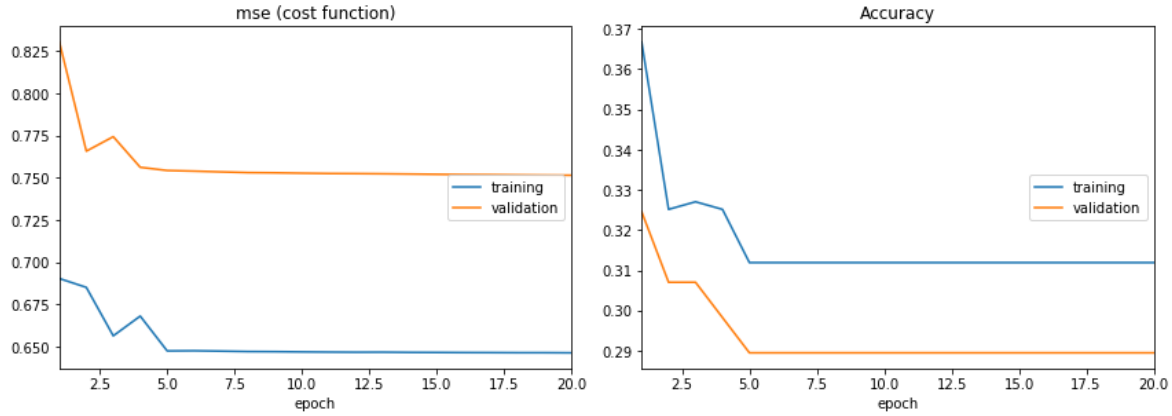
input data x (predict from)	output data y (to predict)				
fmihvPilvien määrä (1/8)	fmihvIlmanpaine (msl) (hPa)	fmihvSuhteellinen kosteus (%)	tloonKuolleet	tloonLoukkaant	thcount
0	998.2	87.1666666666667	0	0	1
0	998.233333333334	91.3333333333333	0	0	1
1	997.95	91.5	0	0	1
6	997.783333333333	93.8333333333333	0	0	1
2.66666666666667	997.75	95.1666666666667	0	0	1
5.83333333333333	997.583333333333	92.3333333333333	0	0	1
6.16666666666667	997.45	93.5	0	0	1
5.16666666666667	997.233333333333	95.8333333333333	0	0	1
7.83333333333333	997.316666666667	96.3333333333333	0	0	1
7.5	997.533333333333	97	0	0	1

A dense model is simpler, because it is used to predict the output variables from certain input variables from the same datapoint; hence, if time series data is used, no time series is considered in the dense model. Only one row at a time is considered (See Table 16. Dense data example).

Table 16. Dense data example

input data x (predict from)	output data y (to predict)				
fmihvPilvien määrä (1/8)	fmihvIlmanpaine (msl) (hPa)	fmihvSuhteellinen kosteus (%)	tloonKuolleet	tloonLoukkaant	thcount
0	998.2	87.1666666666667	0	0	1
0	998.233333333334	91.3333333333333	0	0	1
1	997.95	91.5	0	0	1
6	997.783333333333	93.8333333333333	0	0	1
2.66666666666667	997.75	95.1666666666667	0	0	1
5.83333333333333	997.583333333333	92.3333333333333	0	0	1
6.16666666666667	997.45	93.5	0	0	1
5.16666666666667	997.233333333333	95.8333333333333	0	0	1
7.83333333333333	997.316666666667	96.3333333333333	0	0	1
7.5	997.533333333333	97	0	0	1

The attempt in predicting amount of traffic incidents with neural networks did not work, because data was not descriptive enough. Features that were used when trying to predict amount of traffic incidents, were changed many times and different ones were tested, but mainly part of weather data or whole weather data was used when predicting one feature, amount of traffic incidents. Most of the time learning process for neural network is similar as illustrated in Figure 22, where cost and accuracy are plotted for 20 epochs. Chapter 2.9 explains how cost function and accuracy results should look like for optimal prediction results and when it can be verified that a neural network model has learned something.




```
mse (cost function):
training   (min:    0.646, max:    0.690, cur:    0.646)
validation (min:    0.751, max:    0.830, cur:    0.751)

Accuracy:
training   (min:    0.312, max:    0.367, cur:    0.312)
validation (min:    0.289, max:    0.325, cur:    0.289)
```

Figure 22. Common cost and accuracy result

4.5 Regressors

4.5.1 Random Forest Regressor

Random Forest Regressor (`sklearn.ensemble.RandomForestRegressor`) gives a good variation of predictions. In these predictions, air temperature is used as the input variable and the amount of injuries in traffic incidents as the output variable. The predictions are evaluated from test data that has not been trained on a model. With the used parameters it does not learn to predict values near minimum and maximum values and the predictions distribute a near average (See Figure 23. Random Forest Regressor prediction distribution histogram). Predictions distribution can be examined more closely by plotting the target values and predicted values (See Figure 24. Random Forest Regressor prediction distribution scatter).

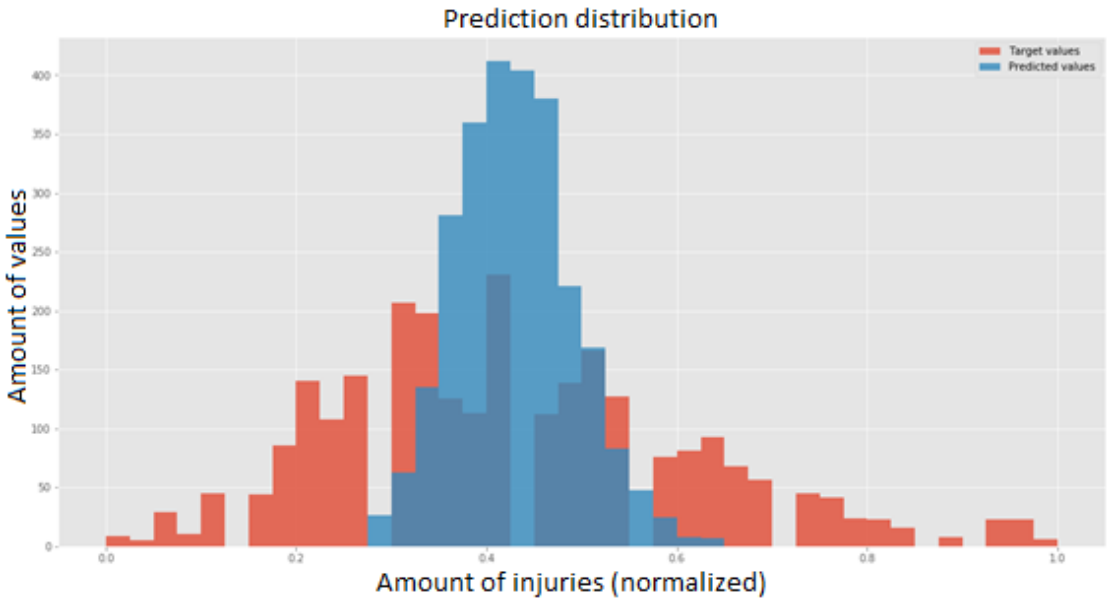


Figure 23. Random Forest Regressor prediction distribution histogram

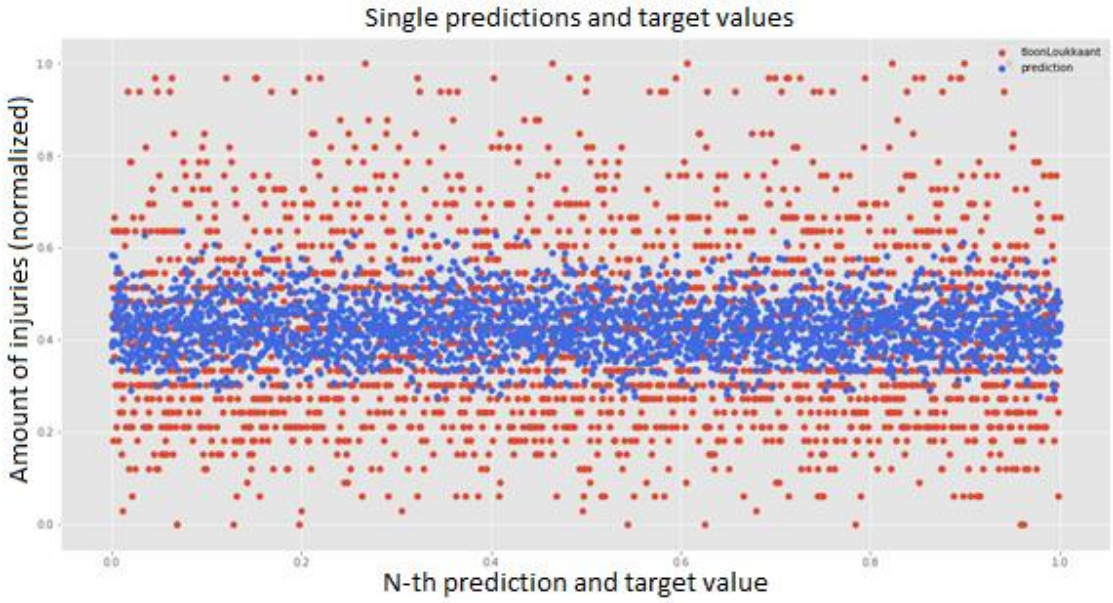


Figure 24. Random Forest Regressor prediction distribution scatter

This is a good result for a simple regressor; however, the goal here is to learn the values near boundaries and see if some anomaly values can be detected by using more features and learn if a certain group of those features affects predictions differently compared to some other group.

4.5.2 Decision Tree Regressor

Test data that has not be trained on a model is to be predicted from; hence, overfitting should not be possible. With Decision Tree Regressor (`Sklearn.tree.DecisionTreeRegressor`) using default parameters one gets much wider distributed predictions (See Figure 25. Decision Tree Regressor prediction distribution). The distribution of predicted values matching target values is unexpected, because prediction accuracy was very low. This means that when comparing a single prediction to the target value, there was a high inaccuracy and the regressor was not able to learn anything.

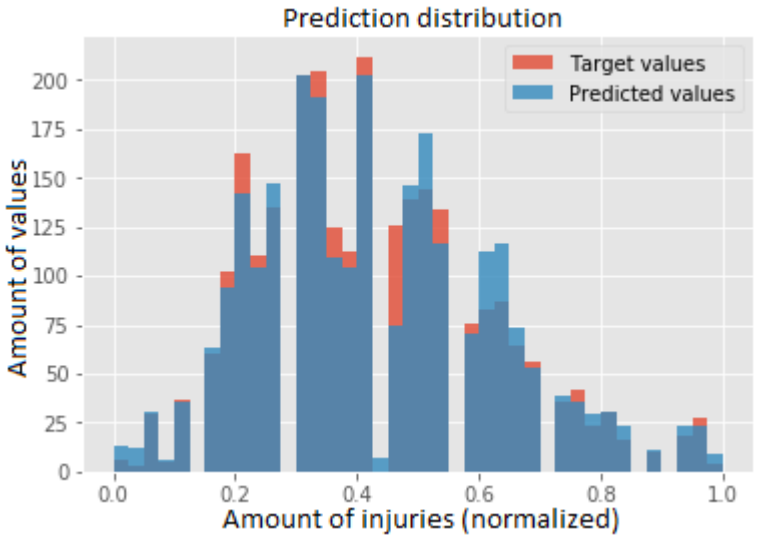


Figure 25. Decision Tree Regressor prediction distribution

4.6 ReLU zero predictions (dying ReLU)

When using `ReLU` as activation, the model is more prone to learn zero values. If the dataset has plenty of zero values for a certain feature, there is a risk that the model will only predict zero for that feature (See Figure 26. Example Keras model with dying ReLU risk).

```
model = Sequential()
model.add(LSTM(dataset["output_dim"],
input_shape=dataset["input_shape"],
               activation='relu',
recurrent_activation='hard_sigmoid'))

rmsprop = optimizers.RMSprop(lr=0.001)
model.compile(loss="mse", optimizer=rmsprop)
```

Figure 26. Example Keras model with dying ReLU risk

Plotting the distribution of predicted values and target values shows that in three features there is one that has too many zero values and the model learned to predict only zero for that feature (See Figure 27: One feature with dying ReLU problem).

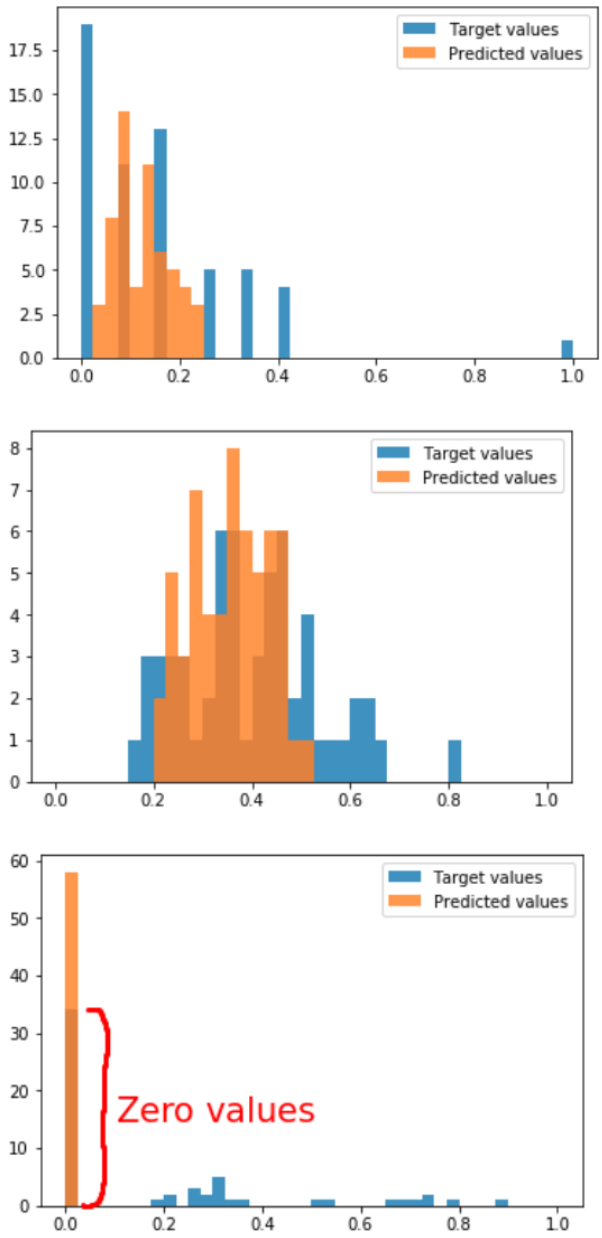


Figure 27: One feature with dying ReLU problem

This is called `dying ReLU problem` and it occurs when ReLU neuron dies and never activates a datapoint again. (What is the “dying ReLU” problem in neural networks? 2015.)

5 Binary classification (predicting categorical variable)

5.1 Dataset

In this chapter, there is used three-year worth of combined data (2015-2017), where most frequent feature is datapoint per day. It would have been possible to combine

dataset to have datapoint per hour, but features that one wants to predict, in most frequent case, have only datapoint per day, so other features with more frequent data are compressed with data combining methods that are described in chapter 3.2.3.

5.2 Feature engineering

After failed attempts to make regressors and neural networks learn the amount of traffic incidents, amount of deaths and amount of injuries from data, the decision was to simplify the problem and make it a binary classification problem by converting the number of incidents to binary classes. The logic was if there were no traffic incidents, the binary class is 0 and if there are 1 or more traffic incidents, then the binary class is 1. This resulted in roughly balanced classes on the dataset, which means that the model that learns from this data will not learn the other class that much than the other.

For binary classification, all compression methods were used for one feature. From one single feature, the following features were calculated: the amount of datapoints, maximum value of all occurred values, minimum value of all occurred values, mean value, standard deviation and sum of all values. While a great deal of data had to be compressed for certain time accuracy (day), there was plenty of metrics that includes much information of that compressed data; hence, compressing all that data does not have that big an impact on analysis performance.

5.3 Time accuracy

Time accuracy (or frequency) of day was used, because target values were represented as number of incidents per day. There is no information about precise time of day when certain incident happened.

5.4 Feature selection

In binary classification, an effective way to find descriptive features is to loop through all features, split a feature by binary class and plot values on those two “bins” on the same graph as histogram. This shows how values distribute for both of

those binary classes. Cases where those two histograms overlap as little as possible, there is a very descriptive feature. In this case, all features overlap so much that none of them could be used for predicting if traffic incident(s) happen or does not happen. When comparing feature `road surface friction` to traffic incident binary class `1 for happen` or `0 for not happen`, one can plot `road surface friction` values as histogram to visualize distribution of those values, then one can see how much the distribution for `happen` and `not happen` differs from each other (Figure 28. Road surface friction distribution). In the figure, one can see that it does not differ, and one cannot use road surface friction feature for binary classification model, because nothing is learnable from this feature (Figure 28. Road surface friction distribution).

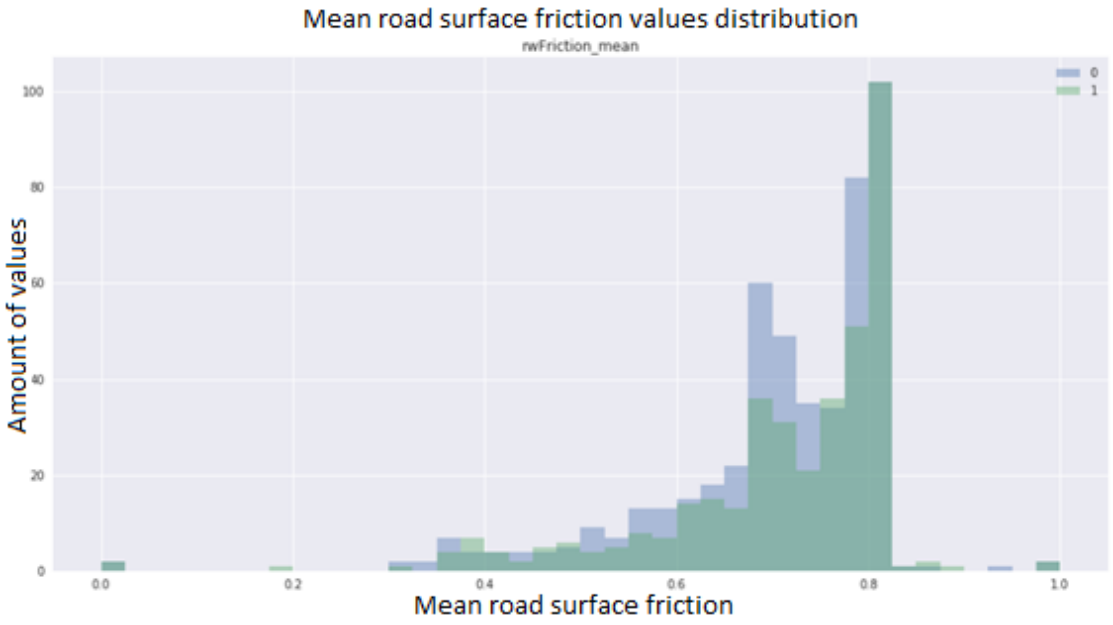


Figure 28. Road surface friction distribution

In chapter 4.3.1 air temperature seemed to have a slight correlation with the amount of injuries and deaths, and the number of month also had correlated. It seems like more incidents happen in summer, however, there are still plenty of days where incidents do not happen, where feature values are similar to the ones where incidents do happen. When comparing `air temperature` to traffic incident `happen` and `not happen`, it can be seen that there is slightly more of `not happen` (blue color); nevertheless, the general shape of distribution is the same (Figure 29: Air temperature distribution). This means that neither `air temperature` can be used for training binary classification model.

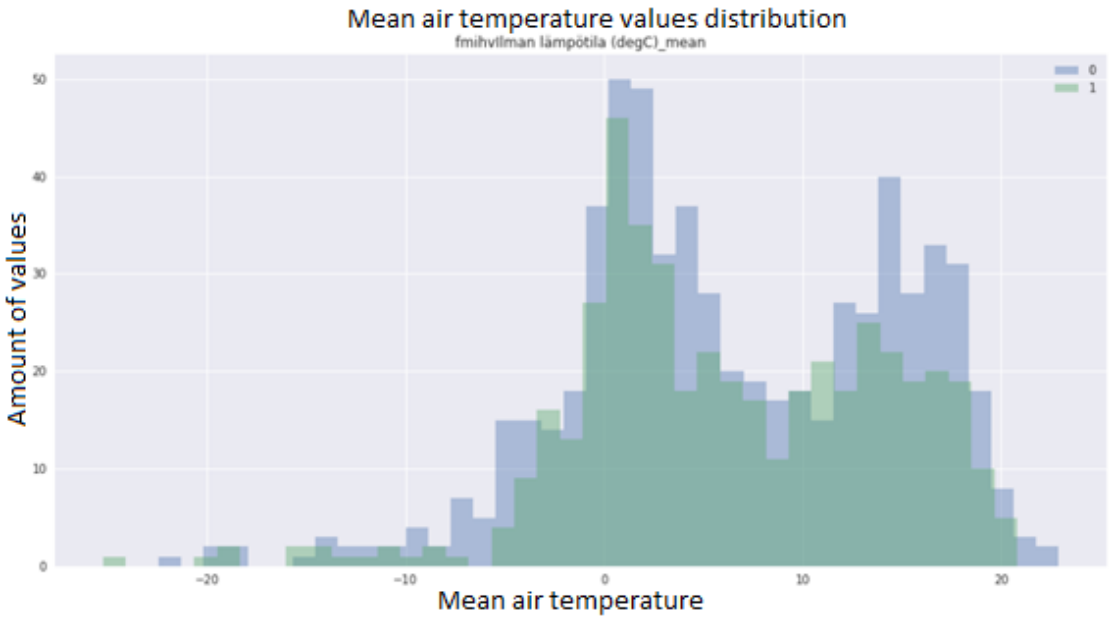


Figure 29: Air temperature distribution

All ~780 features included in the dataset show similar patterns, which means that none of the features there are can be used for predicting traffic incidents.

6 Occasions for traffic incidents

Since traffic incidents are not predictable with day accuracy data from available open data sources, one can examine occasions where traffic incidents happen more often, so one can evaluate from a certain feature a rough probability for a traffic incident to happen. With more precise data for traffic incidents, the situation could be different if time available for a single traffic incident was recorded. However, it seems like traffic incidents depend a great deal on humane behavior, which as well is not easily predictable.

Generally, it seems that more deaths and injuries in traffic incidents happen more often in summer, when it is lighter and warmer as shown in chapter 4.3.1. This could be due to slipperiness in winter that makes drivers drive slower, which reduces the amount of deaths and injuries, however, not necessarily the amount of traffic incidents.

When comparing if incidents `happened (1)` or `not happened (0)`, there is an interesting outlier in the minimum road surface friction values. The following Figure 30 represents the distribution of values for road surface friction separately for

days where incidents `happened` and `not happened`. There are generally more days where incidents do not happen, so if at some value range there is more of `happened` than `not happened`, it should be considered an unusual occasion, where traffic incidents are more likely to happen. It seems that traffic incidents happen more often if the minimum road surface friction is between values 0.75 – 0.8. Comparing the difference of outlier `happened` and `not happened` values to the total amount of values plotted in the figure, one can calculate the classical probability of 1.3% that the randomly chosen day is the day, when one or more traffic incident happens while the minimum road surface friction for the day is between 0.75 and 0.8. (See Figure 30: Road surface minimum friction outlier)

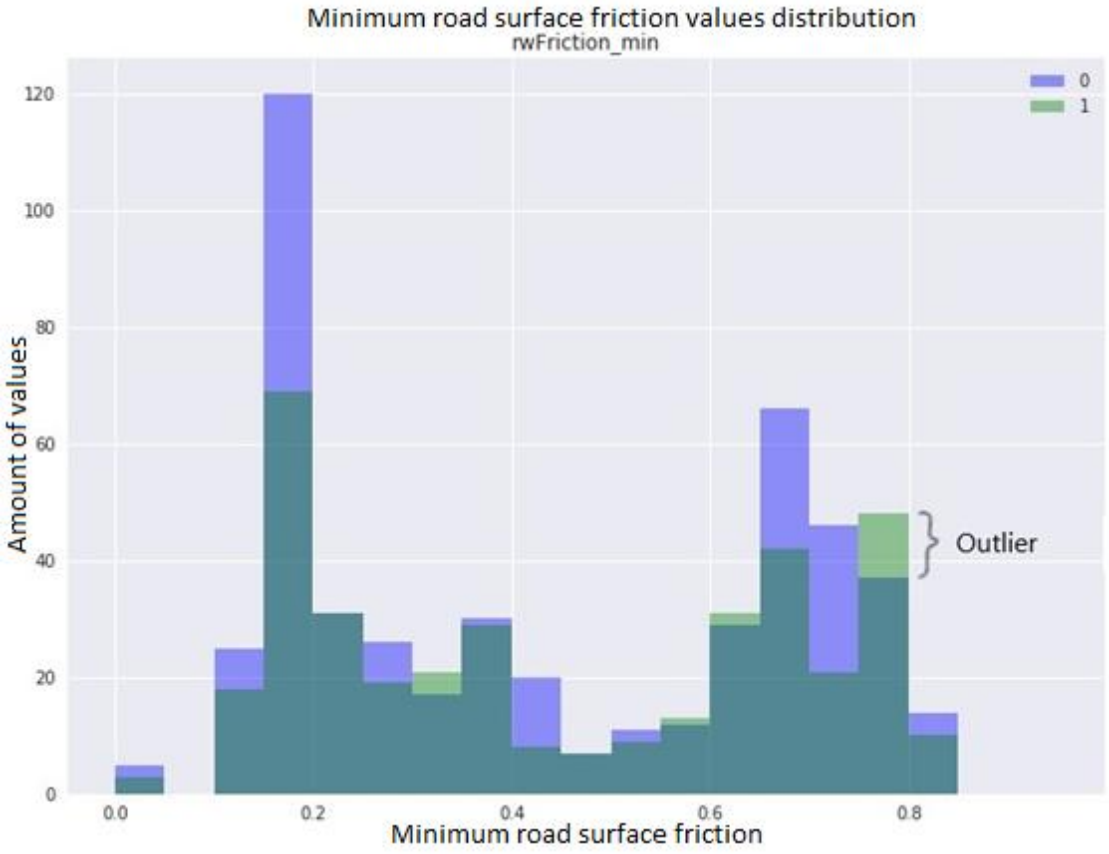


Figure 30: Road surface minimum friction outlier

If one continues examining the occasions on other features, where minimum road surface friction is between 0.75 and 0.8, one can find occasions causing the minimum road surface friction to be 0.75 – 0.8. Examining an obvious feature, air temperature distribution, air temperature is much more often over 10 Celsius degrees than under 10 Celsius degrees, when the minimum road surface friction is between 0.75 – 0.8.

The figure contains only air temperature data where the minimum road surface friction is between 0.75 – 0.8. This examination is one factor more that suggests that in summer more traffic incidents happen. (See Figure 31: Air temperature, where minimum road surface friction is between 0.75 and 0.8)

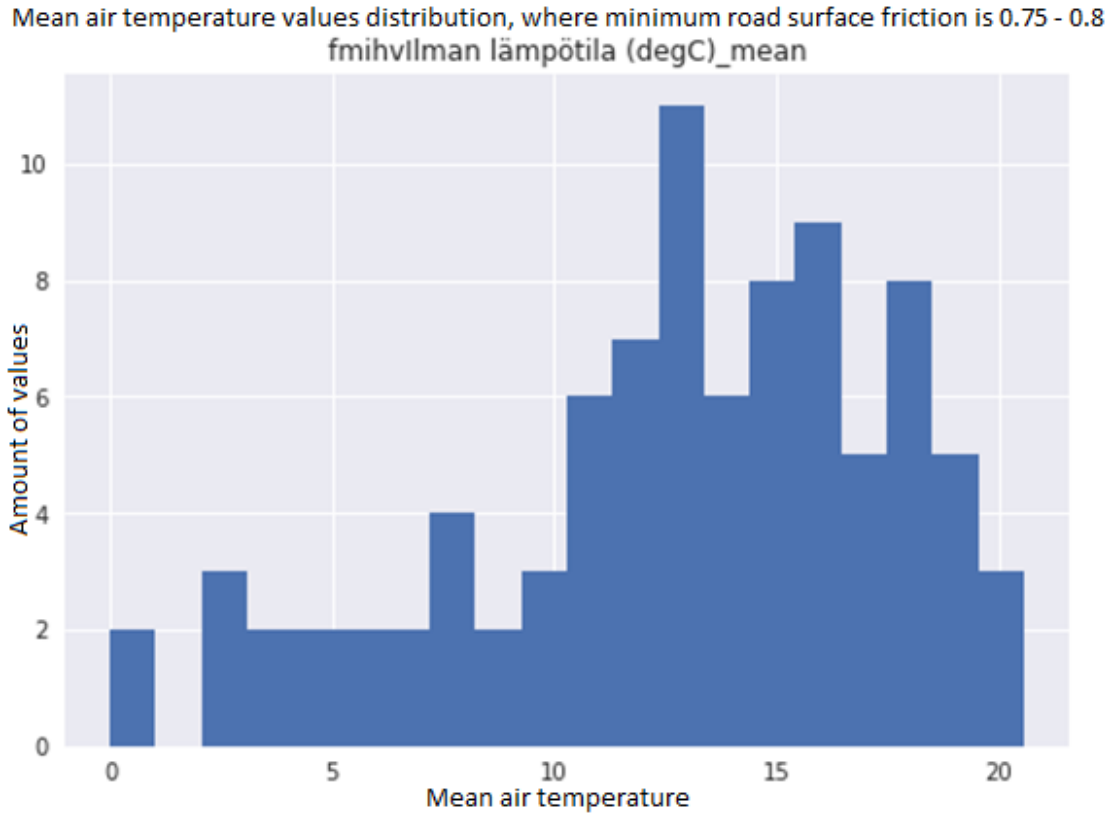


Figure 31: Air temperature, where minimum road surface friction is between 0.75 and 0.8

When examining feature `rwFriction2` maximum value for the day, one can see that when there is more `happened incident(s)` rather than `not happened incidents` for certain road surface friction, everytime road surface friction is above 0.3, and never below. This suggests that incidents tend to happen when there is a good friction on the road and driving conditions are optimal. There are arrows in Figure 32 pointing at these occasions. (See Figure 32: Road surface maximum friction occasions)

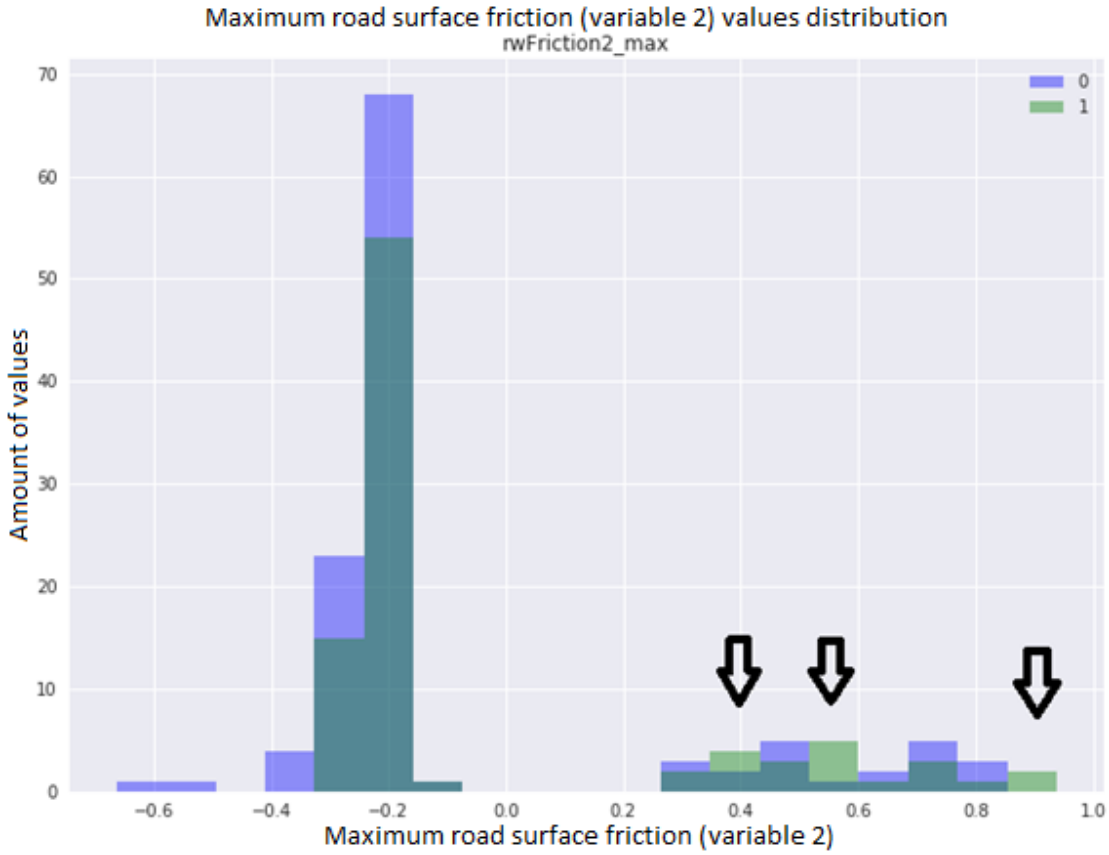


Figure 32: Road surface maximum friction occasions

There can be many reasons why more deadly traffic incidents happen with optimal driving conditions. Motorcycles can also affect statistics, because more deaths begin to happen just before summer and the amount of deaths begins to decrease after summer. If a motorcycle is involved in an incident, one could make the conclusion that a death is more likely to happen than when there is a four-wheel vehicle involved in a traffic incident. While the amount of traffic incidents remain approximately the same in dependent of the time of year, one hypothesis is that the amount of four-wheel vehicle involved traffic incidents decreases in summer; however, the amount of motorcycles involved in traffic incidents increases rapidly in summer.

7 Conclusion

Even though the data features from the used open data sources could not be used for predicting traffic incidents, plenty of work has been done to figure this out, and

numerous attempts were made with different predictive models and differently engineered data. It is not all for nothing, since the biggest deed was made for the dataset combining tool, which can now be used for other data analysis projects that need combining of time series data. There was much problem solving included in developing the dataset combining tool, such as memory consumption, which became a problem, when the datasets size went up to gigabytes. Because of big datasets, combining would not be possible with too low memory (author's computer had 16GB, which was not enough). Nor would it be possible without memory efficient combining function, which reads dataset in streams with Pandas Python library, and appends temporary files on disk where data is "sorted" and stored (instead of memory). Sorted data can be handled afterwards one by one, and when all calculations are performed, and temporary files become one row of CSV data, then all rows can be concatenated to a final combined dataset.

There were many attempts to predict the amount of traffic incidents, deaths and injuries caused by traffic incidents with neural networks. However, the data was not descriptive enough for those features that was to be predicted; hence, neural networks failed to make any reasonable predictions, and the predictions were only slightly more accurate than what could be gained with classical probability or "rolling a dice".

After the failed attempts to predict the amount of traffic incidents, deaths and injuries, the problem was simplified to predicting only if incident(s) happened or did not happen on a certain day. This seemed to be a promising approach, because there were two possible outcomes 0 or 1, where 0 was `not happened` and 1 was `happened`. When this new feature engineering was implemented in the dataset, it turned out that those two possible outcomes were relatively balanced. In the dataset with 3 years of data with time accuracy (or frequency) of day, there were 634 of `not happened` and 463 of `happened`. Hence the decision was made to not balance the dataset any further to avoid losing data. In the phase of feature selection, it was noticed that there is no feature describing the outcome well enough to make any reasonable predictions. However, because there were 634 (58 %) of `not happened` and 463 (42 %) of `happened`, there was one obvious possibility left to examine: occasions where traffic incidents happen more often. If an occasion exists where there

is more of `happened` than `not happened`, it must be an uncommon occasion, because there is that much more of `not happened`. Many of those occasions exist where some of them might not mean nothing, and only few examples were selected, which supports a hypothesis where in summer there happen more of deadly traffic incidents.

In the end, most of the time goes to preparing data (Warden 2018).

Garbage in, garbage out

References

Amatriain, X. 2016. What are hyperparameters in machine learning?. Accessed on 29.1.2019. Retrieved from https://www.quora.com/What-are-hyperparameters-in-machine-learning

Brownlee, J. 2014. An Introduction to Feature Selection. Accessed on 19.11.2018. Retrieved from https://machinelearningmastery.com/an-introduction-to-feature-selection/

Brownlee, J. 2018. A Gentle Introduction to k-fold Cross-Validation. Accessed on 28.11.2018. Retrieved from https://machinelearningmastery.com/k-fold-cross-validation/

Cleveland, W. 1979. Robust Locally Weighted Regression and Smoothing Scatterplots. Accessed on 14.1.2019. Retrieved from https://pdfs.semanticscholar.org/414e/5d1f5a75e2327d99b5bbb93f2e4e241c5acc.pdf

Could the equation of the curve provided by LOESS be obtained?. 2017. StackExchange community question. Accessed on 11.1.2019. Retrieved from https://stats.stackexchange.com/questions/264231/could-the-equation-of-the-curve-provided-by-loess-be-obtained

Credit Card Fraud Detection. 2018. Kaggle dataset. Accessed on 13.1.2019. Retrieved from https://www.kaggle.com/mlg-ulb/creditcardfraud

Dayananda, S. 2018. What is the main difference between classification problems and regression problems in machine learning?. Accessed on 29.1.2019. Retrieved from https://www.quora.com/What-is-the-main-difference-between-classification-problems-and-regression-problems-in-machine-learning

European Regional Development Fund funded project abstract. N.d. Project funding application. Accessed on 7.2.2019. Retrieved from https://www.eura2014.fi/rrtiepa/projekti.php?projektikoodi=A73893

Feature Selection. N.d. Scikit learn documentation. Accessed on 19.11.2018. Retrieved from https://scikit-learn.org/stable/modules/feature_selection.html

Ghosh, P. 2018. How is Bad Data Crippling Your Data Analytics?. Accessed on 3.12.2018. Retrieved from http://www.dataversity.net/bad-data-crippling-data-analytics/

How is direction of weight change determined by Gradient Descent algorithm. 2018. StackExchange community question. Accessed on 29.1.2019. Retrieved from https://ai.stackexchange.com/questions/5670/how-is-direction-of-weight-change-determined-by-gradient-descent-algorithm

In depth skewed data classif. (93% recall acc now). 2016. Jupyter notebook kernel for Kaggle dataset. Accessed on 13.1.2019. Retrieved from https://www.kaggle.com/joparga3/in-depth-skewed-data-classif-93-recall-acc-now

Liberman, N. 2017. Decision Trees and Random Forests. Accessed on 4.12.2018. Retrieved from https://towardsdatascience.com/decision-trees-and-random-forests-df0c3123f991

McCallister, J. N.d. Pearson Correlation Coefficient: Formula, Example & Significance. Accessed on 12.1.2019. Retrieved from https://study.com/academy/lesson/pearson-correlation-coefficient-formula-example-significance.html

Overfitting in Machine Learning: What It Is and How to Prevent It. 2017. Article on EliteDataScience. Accessed on 28.11.2018. Retrieved from https://elitedatascience.com/overfitting-in-machine-learning

Patel, N. N.d. What is Data Quality and How Do You Measure It for Best Results?. Accessed on 14.1.2019. Retrieved from https://neilpatel.com/blog/data-quality/

Pearson Correlations – Quick Introduction. N.d. Spss tutorials. Accessed on 13.11.2018. Retrieved from https://www.spss-tutorials.com/pearson-correlation-coefficient/

Predicting Fraud with TensorFlow. 2017. Jupyter notebook kernel for Kaggle dataset. Accessed on 14.1.2019. Retrieved from https://www.kaggle.com/currie32/predicting-fraud-with-tensorflow

Raschka, S. 2014. About Feature Scaling and Normalization. Accessed on 21.1.2019. Retrieved from https://sebastianraschka.com/Articles/2014_about_feature_scaling.html

Ray, S. 2018. Improve Your Model Performance using Cross Validation (in Python and R). Accessed on 28.11.2018. Retrieved from https://www.analyticsvidhya.com/blog/2018/05/improve-model-performance-cross-validation-in-python-r/

Redman, T. 2018. If Your Data Is Bad, Your Machine Learning Tools Are Useless. Accessed on 3.12.2018. Retrieved from https://hbr.org/2018/04/if-your-data-is-bad-your-machine-learning-tools-are-useless

Siganos, D. & Stergiou, C. N.d. Neural Networks. Accessed on 21.1.2019. Retrieved from https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html

Standard Deviation. N.d. Statistical guide website. Accessed on 21.1.2019. Retrieved from https://statistics.laerd.com/statistical-guides/measures-of-spread-standard-deviation.php

Stephanie. 2013. Lowess Smoothing in Statistics: What is it?. Accessed on 13.11.2018. Retrieved from https://www.statisticshowto.datasciencecentral.com/lowess-smoothing/

The 5 Levels of Machine Learning Iteration. 2017. Article on EliteDataScience. Accessed on 28.11.2018. Retrieved from https://elitedatascience.com/machine-learning-iteration

Warden, P. 2018. Why you need to improve your training data, and how to do it. Accessed on 14.1.2019. Retrieved from https://petewarden.com/2018/05/28/why-you-need-to-improve-your-training-data-and-how-to-do-it/

What is the “dying ReLU” problem in neural networks?. 2015. StackExchange community question. Accessed on 13.11.2018. Retrieved from https://datascience.stackexchange.com/questions/5706/what-is-the-dying-relu-problem-in-neural-networks

Zheng, A. 2015. Evaluating machine Learning Models. Accessed on 29.1.2019. Retrieved from https://www.oreilly.com/ideas/evaluating-machine-learning-models/page/3/evaluation-metrics

Overfitting. N.d. Investopedia documentation. Accessed on 28.11.2018. Retrieved from https://www.investopedia.com/terms/o/overfitting.asp

Bourke, P. 1999. Interpolation methods. Accessed on 29.1.2019. Retrieved from http://paulbourke.net/miscellaneous/interpolation/

Wilder-James, E. 2016. Breaking Down Data Silos. Accessed on 3.12.2018. Retrieved from https://hbr.org/2016/12/breaking-down-data-silos

Sehra, C. 2018. Decision Trees Explained Easily. Accessed on 10.12.2018. Retrieved from https://medium.com/@chiragsehra42/decision-trees-explained-easily-28f23241248