University of Ljubljana
Faculty of Computer and Information Science

Jakob Šalej

AIoT and edge computing of sensor data

MASTER'S THESIS

THE 2nd CYCLE MASTER'S STUDY PROGRAMME
COMPUTER AND INFORMATION SCIENCE
Track: Computer and Information Science

Supervisor: doc. dr. Mira Trebar
Co-supervisor:

Ljubljana, 2020

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jakob Šalej

AIoT in robna obdelava senzorskih podatkov

magistrsko delo

magistrski program druge stopnje
računalništvo in informatika
Smer: Računalništvo in Informatika

Mentor: doc. dr. Mira Trebar

Somentor:

Ljubljana, 2020

Acknowledgments

Worth mentioning in the acknowledgment is everyone who contributed to your thesis.

Jakob Šalej, 2020

To all the flowers of this world.

"The only reason for time is so that everything doesn't happen at once."

— Albert Einstein

Contents

List of used acronmys

acronym	**meaning**
CA	classification accuracy
DBMS	database management system
SVM	support vector machine
...	...

Abstract

Title: AIoT and edge computing of sensor data

This sample document presents an approach to typesetting your BSc thesis using LaTeX. A proper abstract should contain around 100 words which makes this one way too short. A good abstract contains: (1) a short description of the tackled problem, (2) a short description of your approach to solving the problem, and (3) (the most successful) result or contribution in your thesis.

Keywords

internet of things, machine learning, sensor data, edge computing, anomaly detection

Povzetek

Naslov: AIoT in robna obdelava senzorskih podatkov

V vzorcu je predstavljen postopek priprave magistrskega dela z uporabo okolja LaTeX. Vaš povzetek mora sicer vsebovati približno 100 besed, ta tukaj je odločno prekratek. Dober povzetek vključuje: (1) kratek opis obravnavanega problema, (2) kratek opis vašega pristopa za reševanje tega problema in (3) (najbolj uspešen) rezultat ali prispevek magistrske naloge.

Ključne besede

računalnik, računalnik, računalnik

Razširjeni povzetek

To je primer razširjenega povzetka v slovenščini, ki je obvezen za naloge pisane v angleščini. Razširjeni povzetek mora vsebovati vse glavne elemente dela napisanega v angleščini skupaj s kratkim uvodom in povzetkom glavnih elementov metode, glavnih eksperimentalnih rezultatov in glavnih ugotovitev. Razširjeni povzetek naj bo strukturiran v podpoglavja (spodaj je naveden le okvirni primer in je nezavezujoč). Čez palec navadno razširjeni povzetek nanese okoli 10 odstotkov obsega celotnega dela.

I Kratek pregled sorodnih del

II Predlagana metoda

III Eksperimentalna evaluacija

IV Sklep

poljuben tekst poljuben tekst

Chapter 1

Introduction

Artificial Intelligence (AI) is becoming more and more important in the IoT (Internet of Things) field. Machine Learning (ML) methods are very complex and usually run on big datasets in the cloud. With edge devices growing more powerful, new solutions that enable use of AI on sensor nodes are emerging. Today, the vast majority of the IoT devices send their raw sensor data to the cloud, where it gets stored, processed and analysed [1]. As this leads to high network usage, privacy concerns and slow data processing, moving computing away from the cloud closer to the network edge and even directly to the edge devices can prove effective [2].

Our work will focus on possibilities of data processing on IoT devices using machine learning methods. We propose a low-powered system that is able to execute both learning and prediction locally. Since these systems are under-powered and can run on batteries alone, machine learning algorithms will be evaluated based on their performance with limited resources available. Finally, it will be analyzed if and where can edge computing sufficiently replace cloud computing.

Chapter 2

Related Work

The new category of smart AIoT (Artificial Intelligence + IoT) devices represents systems that are able to locally compute data streams from their sensors in real time, using machine learning [3]. This further enhances capabilities of applications, such as route optimization, accident prevention and road anomalies in the field of smart transportation.

Cloud, fog, edge and mist computing are the latest paradigms in IoT [1]. While fog computing transfers data processing from cloud closer to the devices via fog nodes that are placed close to IoT source nodes, edge and mist computing usually represent data processing on sensor devices themselves.

Important benefits of the edge computing are increased users' privacy and lower bandwidth consumption [4]. By processing data locally, sharing users' data with a cloud server is avoided. Instead of sending raw data, only computed information is transmitted, saving network bandwidth and keeping users' information at the edge, hidden behind mediators.

When moving computing from the cloud to the edges of the network, performance capabilities of edge nodes suddenly becomes important and new challenges arise when deploying applications. One potential solution is the use of container virtualization, thus decoupling hardware resources from software and allowing packaged software to be executed everywhere [5].

Another challenge on edge network devices is deep learning with focus on data processing capabilities of low-power sensor systems [6]. Small deep neural network (DNN) models, suitable to run on those systems can be further optimized with methods such as parameter quantization and prunning.

Different machine learning models can be used to predict attacks and anomalies on the IoT systems accurately [7]. Models' performance using accuracy, precision, recall, f1 score and area under the Receiver Operating Characteristic Curve is evaluated and presented as a comparative study on the open data set.

2.1 Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches

In the article [7], Hasan et al. used machine learning algorithms to predict anomalies and attacks on IoT systems, such as Denial of Service, Data Type Probing, Malicious Control, Malicious Operation, Scan, Spying and Wrong Setup. Used machine learning algorithms were Logistic Regression (LR), Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF) and Artificial Neural Network (ANN). Based on employed evaluation metrics such as accuracy, precision, recall, f1 score, and area under the Receiver Operating Characteristic (ROC) Curve, RF performed the best with textit99.4% accuracy on unseen data.

Hasan et al. implemented ML algorithms in Python, using Numpy, Scikit-learn [8], Pandas [9] and Keras [10] libraries and frameworks. Figure 2.1 shows the work process. Used parameters for each algorithm were not specified.

2.1.1 Data Collection

Open source dataset [7] from web platform textitKaggle was used for the analysis. It contains IoT traffic traces captured in IoT environment DS2OS

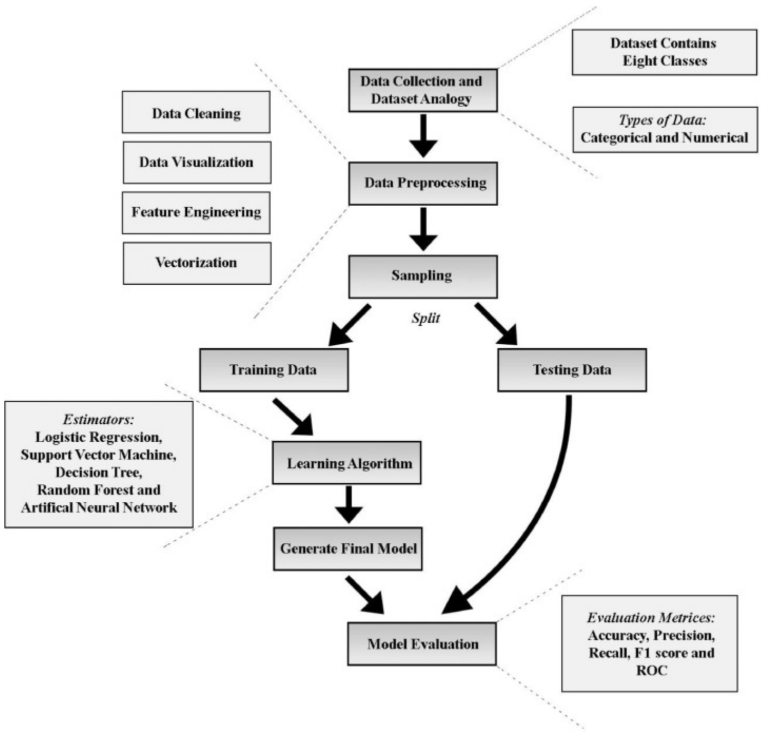
Dataset Contains Eight Classes
Data Collection and Dataset Analogy
Data Cleaning
Types of Data: Categorical and Numerical
Data Visualization
Data Preprocessing
Feature Engineering
Vectorization
Sampling
Split
Training Data
Testing Data
Estimators: Logistic Regression, Support Vector Machine, Decision Tree, Random Forest and Artifical Neural Network
Learning Algorithm
Generate Final Model
Evaluation Metrices: Accuracy, Precision, Recall, F1 score and ROC
Model Evaluation

Figure 2.1: Overall framework for attack and anomaly detection algorithm.

and includes data from different types of devices such as light controllers, thermometers, movement sensors, washing machines, batteries, thermostats, smart doors and smart phones. Each entry has info about the source device, destination device, when and what operation was performed, as well as a *normality level*.

Dataset has 357,952 samples with 13 features. Of those, 347,935 are examples of normal (NL) data, while 10,017 represent anomalous data. There are 5780 samples with Denial of Service (DoS) attack, 342 of Data Type Probing (DP), 889 of Malicious Control (MC), 805 of Malicious Operation (MO), 1547 of Scan (SC), 532 of Spying (SP) and 122 examples of Wrong Setup (WS).

2.1.2 Data Preprocessing

Data preprocessing was performed on the dataset, as in [7]. The following steps were applied:

- 13 samples with corrupted data (unreadable field values) removed,
- all *NaN* values from column **accessedNodeType** replaced with *Malicious*,
- all non-numeric values in column **value** replaced with numeric representations,
- all missing values in **value** column filled with *0*,
- **timestamp** column removed from the dataset (irrelevant),
- label encoding used on all columns except **values**.

2.1.3 Sampling

Data was randomly split into training and testing set, 80% vs 20%. Training set was further split with five different sizes (20%, 40%, 60%, 80%, 100% of training set). Samples were chosen randomly. Testing set was fixed in size.

2.1.4 Learning Algorithms

Logistic Regression, Support Vector Machine, Decision Tree and Random Forest were implemented with *scikit-learn*, using default model parameter values. ANN was implemented using Python Deep Learning library *Keras*.

Logistic Regression

Logistic regression (LR) is a linear model for classification. In the used scikit-learn implementation, *l2* regularization is applied by default. Default solver for the optimization problem is *lbfgs* [11].

Support Vector Machine

Support Vector Machines (SVMs) are supervised learning models used for classification and regression. Because of classification and to ensure fast performance, Linear Support Vector Classification (LinearSVC) implementation was used. By default, it uses *rbf* kernel and *l2* regularization with the strength of 1.0 [12].

Decision Tree

Decision Tree (DT) is a non-parametric supervised learning method for classification. In the used scikit-learn implementation, default criterion for measuring the quality of a split is *Gini impurity* [13].

Random Forest

Random Forest (RF) is one of the ensemble methods which combine the predictions of several base estimators to improve robustness of the estimator. Each tree in the ensemble is built from a sample drawn with replacement from the training set. By default, there are 100 trees in the scikit-learn implementation of the algorithm, with *Gini impurity* as a default measure of split's quality [14].

Artificial Neural Network

Artificial Neural Network (ANN) is a circuit of connected neurons that each deliver outputs based on their inputs and used predefined activation functions [15]. Keras with Tensorflow backend was used for ANN (11 input nodes, hidden layer with 32 nodes and *relu* activation, 8 output nodes with *softmax* activation). Selected optimizer was *Adam optimizer.* Loss function was *sparse categorical crossentropy.*

2.1.5 Model Evaluation

To evaluate the performance and to compare the results to [7], *accuracy* was used as a main performance metric. In addition, standard deviation (STD) of a 5-fold cross-validation on a training set was calculated. Also used were *balanced accuracy* (takes into account dataset imbalance), *precision*, *recall* and *f1 score* on both train and test sets.

Data was split into train (80%) and test (20%) set. Five-fold cross-validation was implemented on training set to determine the effect of model's selected parameters. For training set scores, averages of five runs were used for each metric. This was additionally repeated three times, each time with randomly sampled training set.

Results for testing set were from performance on yet unseen data. Besides evaluations as *accuracy*, *balanced accuracy*, *precision*, *recall* and *f1 score*, confusion matrix and area under the Receiver Operating Characteristic (ROC) graphs were generated.

As can be seen in the next section, the results between models were close. Measuring time required to train a model and time required to calculate predictions was an attempt to look at the performance from a different perspective that could show difference between models.

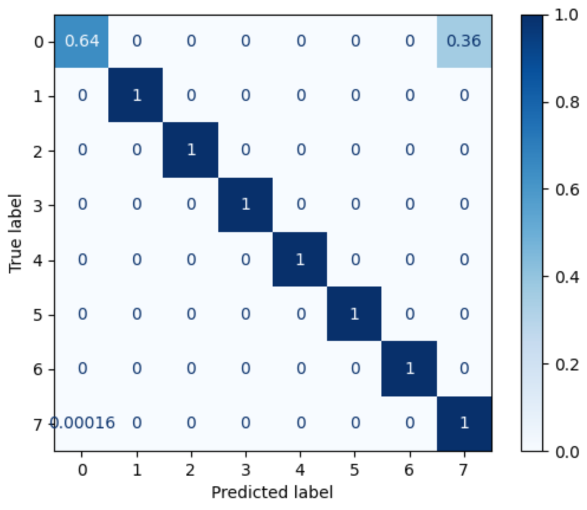

Figure 2.2: Confusion Matrix for DT.

2.1.6 Results

Performance of algorithms on both training and testing set can be seen in **??** and **??** and in Table 2.1. The best performing algorithms based on selected performance metrics are DT and RF. Confusion matrix for DT can be seen in 2.2 and AUC - ROC graph in 2.4. Confusion matrix for RF can be seen in 2.3 and AUC - ROC graph in 2.5.

Since results between DT and RF are almost the same, one can assume that given the high number of samples in dataset, DT is reasonably stable. As RF is just ensemble of many DTs, if those are all stable and return similar results, then results from ensemble do not deviate highly from those of only one tree.

2.1.7 Performance Results

To get an idea of computational requirements of algorithms, their execution speed was measured. All performance tests were done on a plugged in Dell XPS 13 (9350) laptop with 8 GB of LPDDR3 RAM and i7 6500u CPU (2

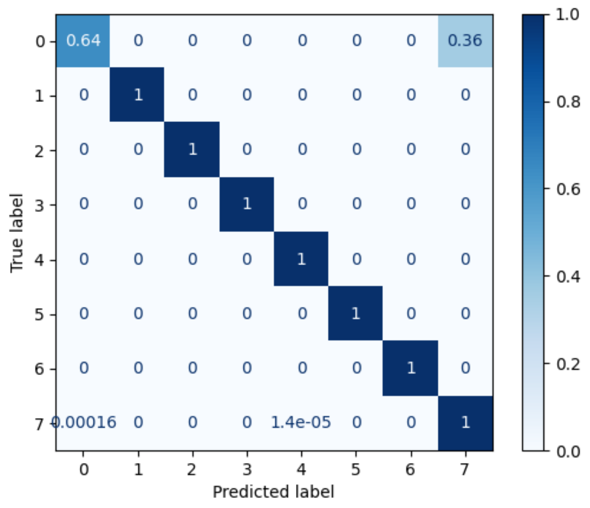

Figure 2.3: Confusion Matrix for RF.

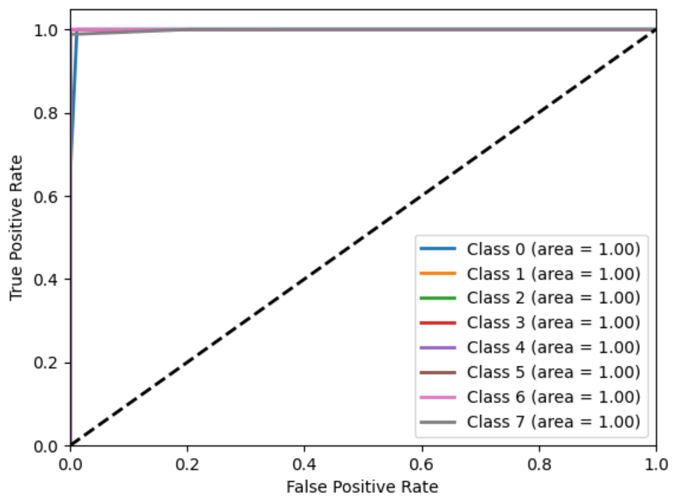

Figure 2.4: AUC - ROC Curve for DT.

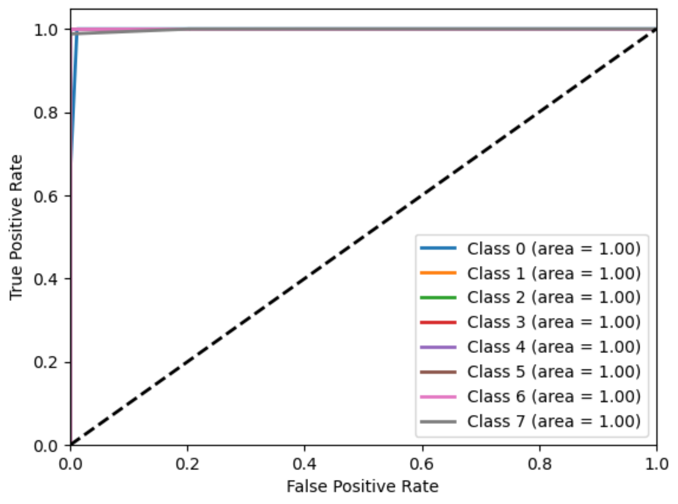

Figure 2.5: AUC - ROC Curve for RF.

cores, 4 threads) [16]. Each measurement was repeated 5 times and the lowest value was selected. These results are not meant to be absolute performance scores, but to allow comparing relative execution speed of selected algorithms.

In 2.6 it can be seen how long it takes for algorithms to train the model on a given data (training set). All five training sizes were used to see how results scale with the increased dataset size. Figure 2.7 shows how long it takes a trained model to predict the outputs of a testing set.

DT's relative simplicity allows it to comfortably win in required training time. As expected, prediction only takes a fraction of training time, with only RF taking significantly more time than other algorithms. Because of lower computational and architectural complexity and superb accuracy, DT is the best performing model on this dataset.

Performance Results on Raspberry Pi

Training and prediction times when running on the Raspberry Pi 4 with 1 GB of RAM can be seen in 2.8 and 2.9. Overall, the low-powered device was

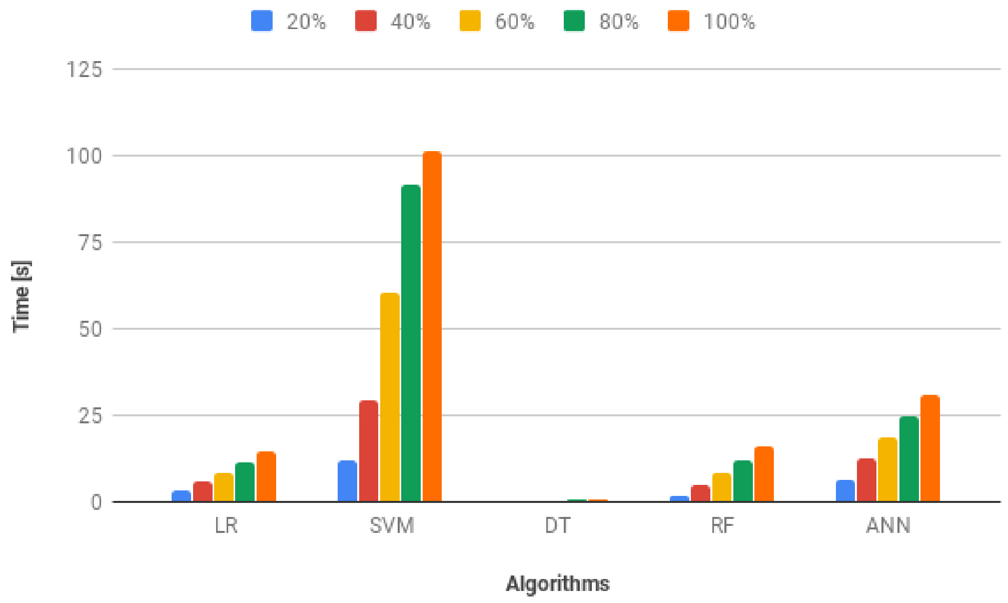

Figure 2.6: Time spent training the model by each algorithm with different training set sizes (% of all training data).

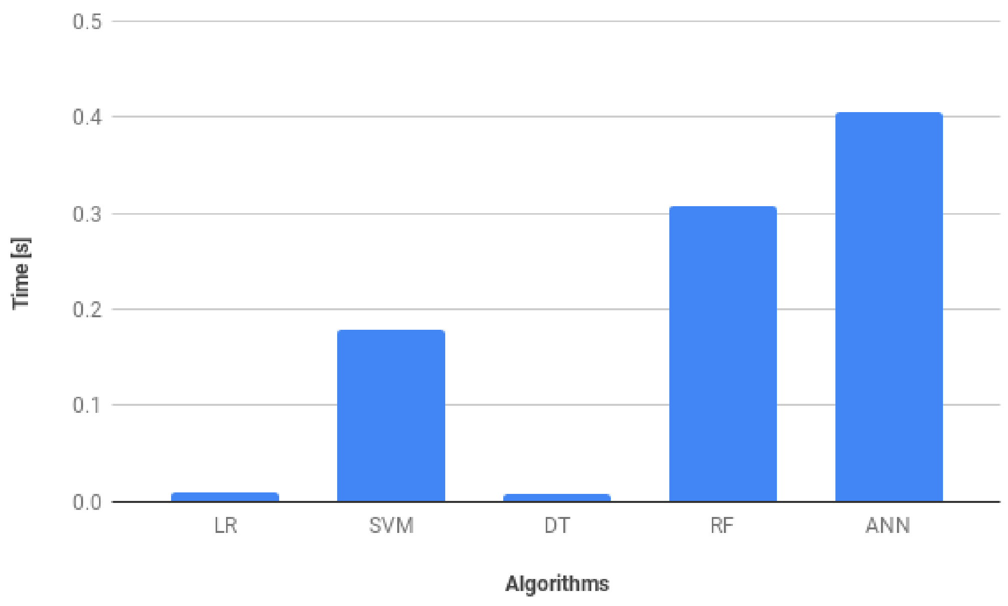

Figure 2.7: Time spent calculating predictions on testing set by each algorithm.

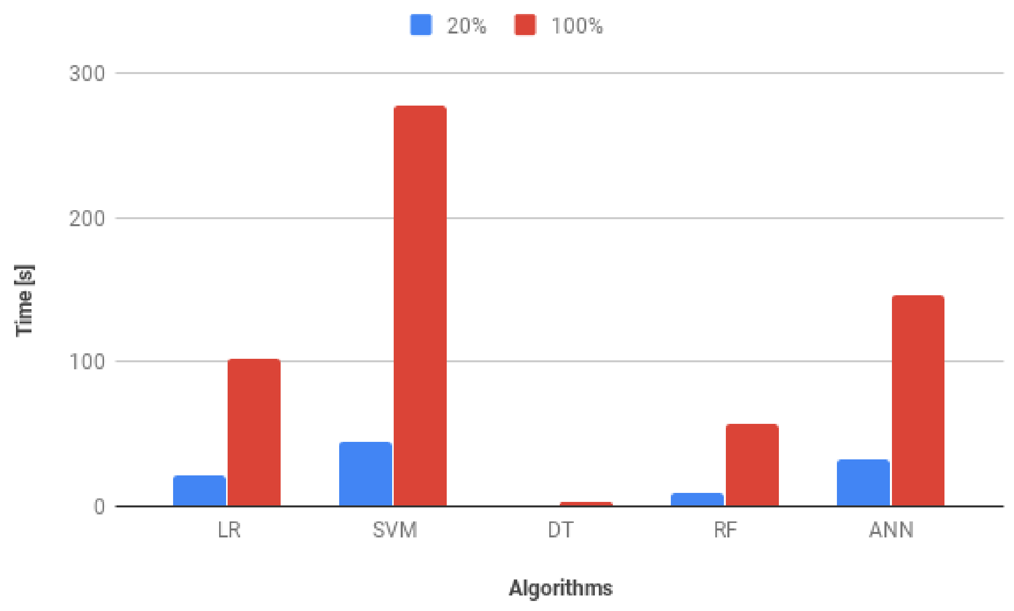

Figure 2.8: Time spent training the model by each algorithm with 20% and 100% of training data on Raspberry Pi.

approximately 5-times slower than laptop system, as shown in 2.10 and 2.11. Nevertheless, DT remains performing very fast.

2.1.8 Conclusion

When compared to the article [7], looking at graphs it is clear that results do not match perfectly. This makes sense, as models in [7] and our models differ in settings of parameters. Since no details of implementation were given, we could only assume the best settings through trial and error.

On the other hand, our best performing algorithms do match the upper bound of achieved accuracy in [7]. This means we were successfully able to repeat their results and even improve them for some algorithms.

Although it might look like a great result given the level of achieved accuracy overall, there is no denying that database is heavily imbalanced.

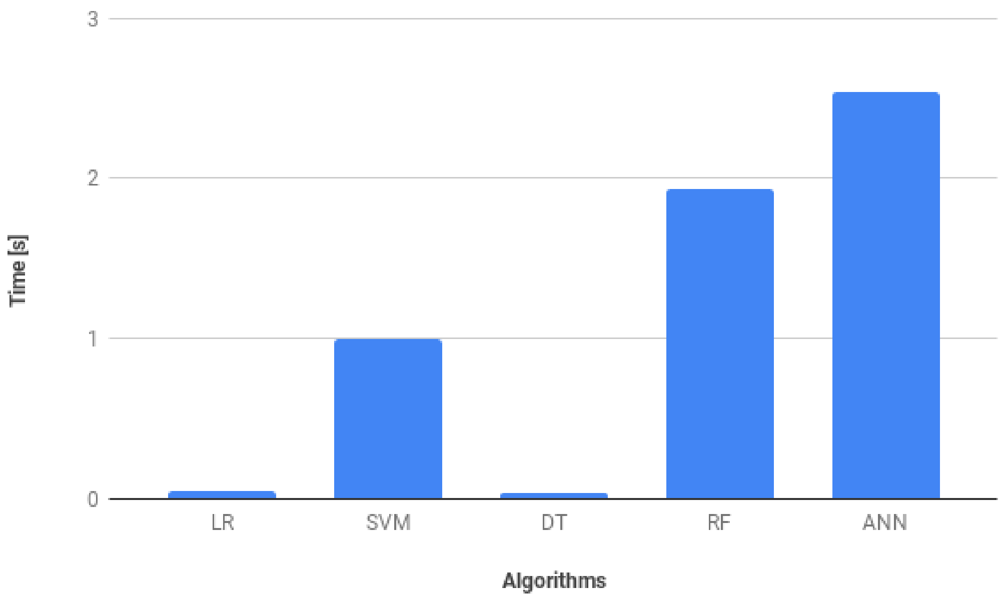

Figure 2.9: Time spent calculating predictions on testing set by each algorithm on Raspberry Pi.

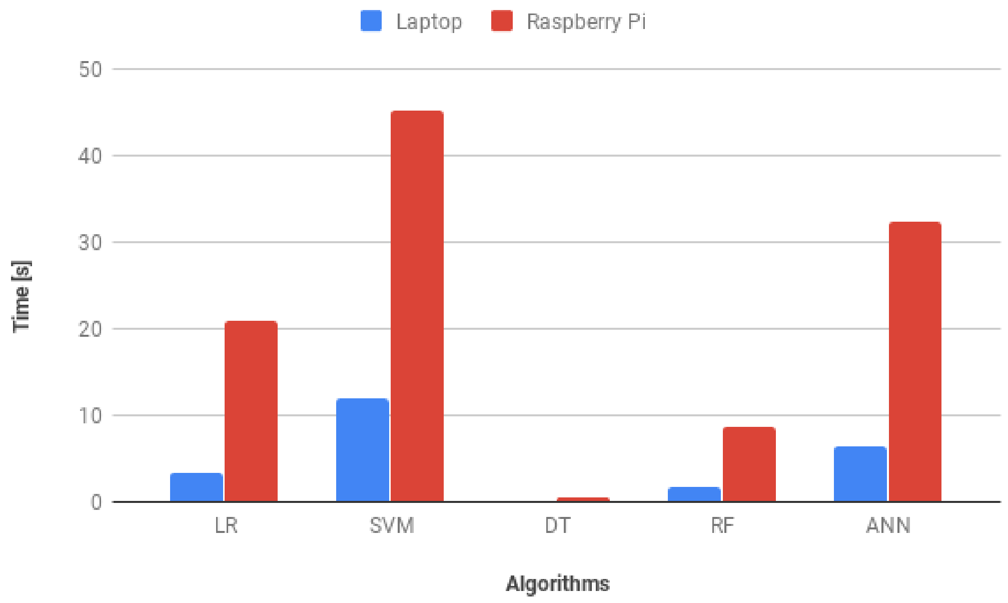

Figure 2.10: Time spent training the model with identical training set (20% of all training set) on Raspberry Pi and XPS 13 laptop system.

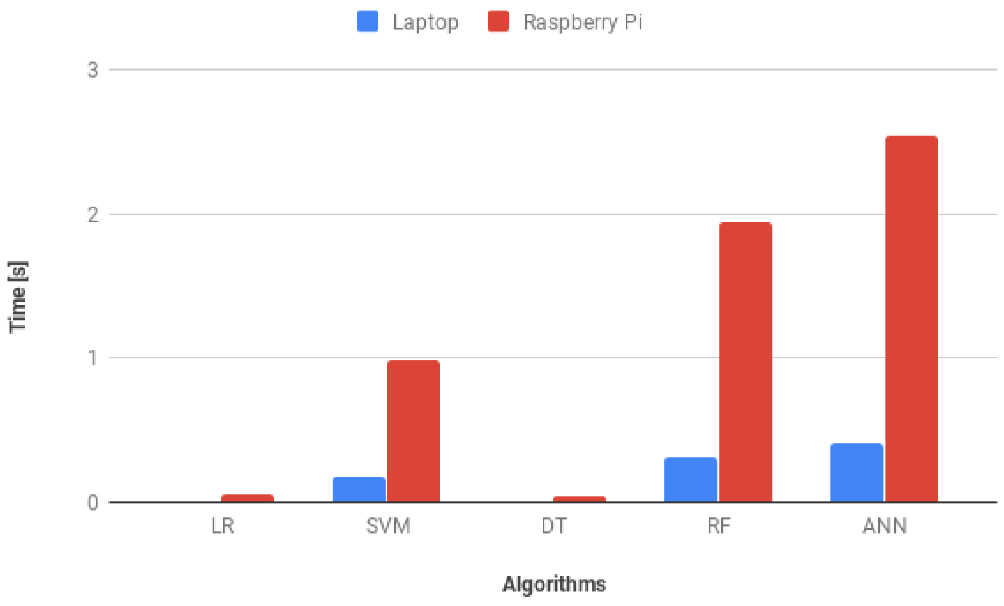

Figure 2.11: Time spent calculating predictions on identical testing set on Raspberry Pi and XPS 13 laptop system.

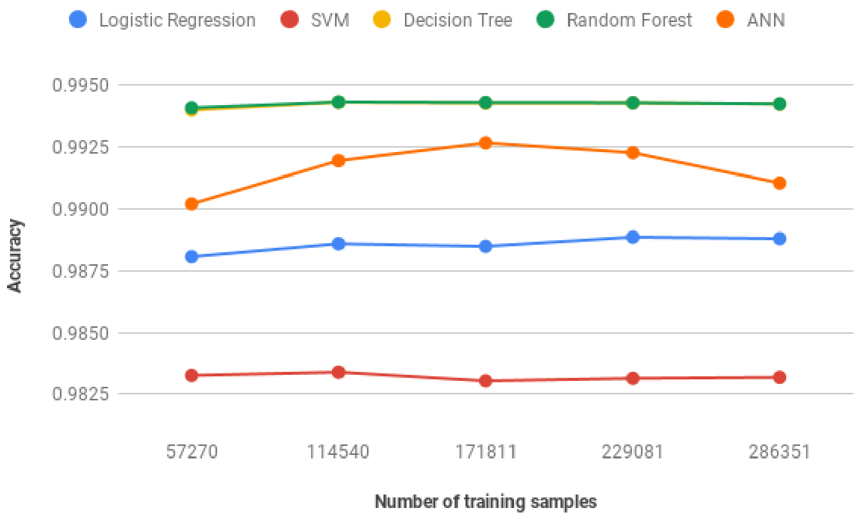

Figure 2.12: Accuracy of algorithms on training sets (5-fold cross validation).

Table 2.1: Performance on training and testing sets

Training	LR	SVM	DT	RF	ANN
Accuracy	0.9886	0.9943	0.9943	0.9943	0.9935
STD	0.0003	0.0002	0.0003	0.0002	0.0002
Balanced accuracy	0.6539	0.9573	0.9573	0.9573	0.9041
Precision	0.9866	0.9942	0.9942	0.9942	0.9934
Recall	0.9886	0.9943	0.9943	0.9943	0.9935
F1	0.9867	0.9937	0.9937	0.9937	0.9929
Testing	LR	SVM	DT	RF	ANN
Accuracy	0.9883	0.9939	0.9939	0.9939	0.9933
Balanced accuracy	0.9596	0.9567	0.9618	0.9578	0.9574
Precision	0.9865	0.9938	0.9938	0.9938	0.9932
Recall	0.9883	0.9939	0.9939	0.9939	0.9933
F1	0.9864	0.9933	0.9933	0.9933	0.9927

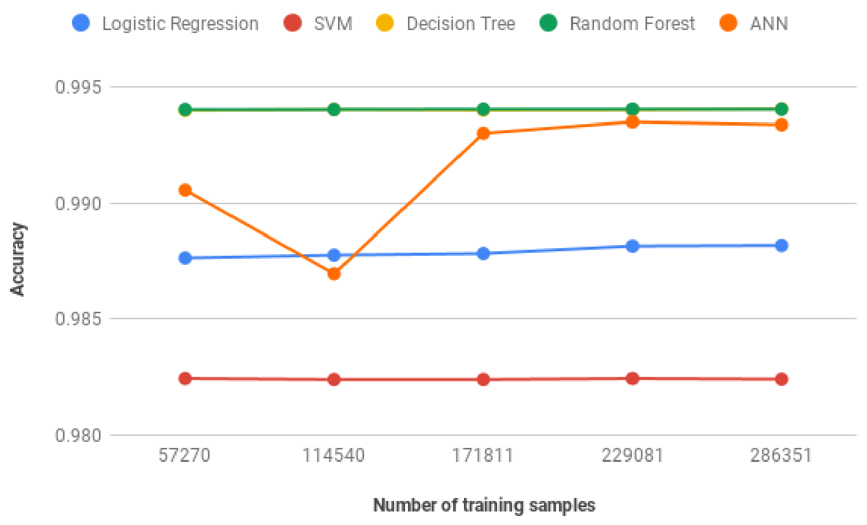

Figure 2.13: Accuracy of algorithms on a fixed sized testing set.

Majority classifier skews the results as other classes are under represented. This does make one thing possible though - optimization through reducing the complexity of training set, since lots of samples might be redundant. That way, training set could be shrunk without a hit to accuracy.

Chapter 3

Plovke: slike in tabele

Chapter 4

Razno

Chapter 5

Kaj pa literatura?

Kot smo omenili že v uvodu, je pravi način za citiranje literature uporaba BibTeXa [?]. Programski paket LaTeXje prvotno predstavljen v priročniku [?] in je v resnici nadgradnja sistema TeX avtorja Donalda Knutha, znanega po denimo, če izpustim njegovo umetnost programiranja, Knuth-Bendixovem algoritmu [?].

Vsem raziskovalcem s področja računalništva pa svetujem v branje mnenje L. Fortnowa [?].

Chapter 6

Sklepne ugotovitve

Izbira LaTeX ali ne LaTeX je seveda prepuščena vam samim. Res je, da so prvi koraki v LaTeXu težavni. Ta dokument naj vam služi kot začetna opora pri hoji.

Appendix A

Title of the appendix 1

Example of the appendix.

Bibliography

[1] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, J. P. Jue, All one needs to know about fog computing and related edge computing paradigms: A complete survey, Journal of Systems Architecture 98 (2019) 289 – 330.

[2] J. Pan, J. McElhannon, Future edge cloud and edge computing for internet of things applications, IEEE Internet of Things Journal 5 (1) (2018) 439–449.

[3] F. Zantalis, G. Koulouras, S. Karabetsos, D. Kandris, A review of machine learning and iot in smart transportation, Future Internet 11 (04 2019).

[4] A. Salem, T. Nadeem, Lamen: Leveraging resources on anonymous mobile edge nodes, in: Proceedings of the Eighth Wireless of the Students, by the Students, and for the Students Workshop, ACM, 2016, pp. 15–17.

[5] R. Morabito, Virtualization on internet of things edge devices with container technologies: A performance evaluation, IEEE Access 5 (2017) 8835–8850.

[6] J. Chen, X. Ran, Deep learning with edge computing: A review, Proceedings of the IEEE PP (2019) 1–20.

[7] M. Hasan, M. M. Islam, M. I. I. Zarif, M. Hashem, Attack and anomaly detection in iot sensors in iot sites using machine learning approaches, Internet of Things 7 (2019) 100059.

[8] scikit-learn: machine learning in python — scikit-learn 0.22.2 documentation, https://scikit-learn.org/stable/, (Accessed on 04/13/2020).

[9] pandas - python data analysis library, https://pandas.pydata.org/, (Accessed on 04/13/2020).

[10] Home - keras documentation, https://keras.io/, (Accessed on 04/13/2020).

[11] 1.1. linear models — scikit-learn 0.22.2 documentation, https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression, (Accessed on 04/13/2020).

[12] sklearn.svm.svc — scikit-learn 0.22.2 documentation, https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html, (Accessed on 04/13/2020).

[13] sklearn.tree.decisiontreeclassifier — scikit-learn 0.22.2 documentation, https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html, (Accessed on 04/13/2020).

[14] 1.11. ensemble methods — scikit-learn 0.22.2 documentation, https://scikit-learn.org/stable/modules/ensemble.html#forest, (Accessed on 04/14/2020).

[15] S.-C. Wang, Artificial Neural Network, Springer US, Boston, MA, 2003, pp. 81–100. doi:10.1007/978-1-4615-0377-4_5.
URL https://doi.org/10.1007/978-1-4615-0377-4_5

[16] Xps 13 9350 specifications, https://downloads.dell.com/manuals/all-products/esuprt_laptop/esuprt_xps_laptop/xps-13-9350-laptop_reference%20guide_en-us.pdf, (Accessed on 05/25/2020).