Abstract

Every year, excavation projects cause costly infrastructure damage that affects convenience and livelihoods of many people in Norway. The purpose of this report is to explore the possibility of predicting the outcome of new projects using machine learning. Depending on the accuracy and probability that a project leads to strike damage, safety measures can be implemented to decrease that risk, with the goal that a machine learning model will serve as a risk assessment tool for future projects. For it to be possible to train machine learning models, Geomatikk AS has provided a huge backlog of metadata about past excavation projects and their outcomes.

Sammendrag

Graveskader forårsaker hvert år kostbare skader på infrastruktur som påvirker levebrødet og skaper ulemper for mange rundt om i Norge. Formålet med denne rapporten er å utforske muligheten til å forutsi utfallet til fremtidige prosjekter ved bruk av maskinlæring. Avhengig av nøyaktighet og sannsynlighet for at et prosjekt vil føre til graveskader, vil det kunne implementeres sikkerhetstiltak for å minske risikoen, dette med et mål om at en maskinlæringsmodell vil kunne brukes som et risikovurdering verktøy for graveskader i fremtiden. For å kunne trene maskinlæringsmodellene har Geomatikk AS gikk tilgang til en stor mengde metadata fra tidligere gravesprosjekter med graveskadeutfall.

Preface

This project is a collaboration between the authors of this article and Geomatikk AS, in an effort to reduce the amount of strike damage that occurs. Geomatikk provides solutions for utility companies, contractors, and public authorities that help coordinate excavation projects. This is done to maximize the safety of contractors and mitigate infrastructure damage. Geomatikk's core product is a comprehensive mapping of all underground infrastructure in the countries it operates. This underpins an end-to-end technology platform that manages these pre-dig checks and complementary workflows such as site inspection, damage resolution, and network monitoring.

The reason for wanting to reduce this is that every time there is an accident, there are huge costs to society. This can be a financial cost or the cost of losing access to critical infrastructure such as electricity, the internet, or water.

The authors thank our advisor Donn Morrison at NTNU for providing feedback and guidance throughout the project.

We would also like to thank Geomatikk for providing this project and giving us access to their office space and resources. We especially want to thank Tor Andreas Aasebø and Espen Bragerhaug at Geomatikk for being helpful in answering questions and discussing the data set and problems along the way.

Trondheim, 20.05.2022

Cato Bakken

Victoria Blichfeldt

Mahmoud Hasan Shawish

Assignment text

Arbeidstittel: Maskinlæring innenfor graveskadeforebygging

Hensikten med oppgaven:
Vi ønsker å se hvilke muligheter maskinlæring kan gi oss for å forutse sjansen for at en graveskade vil skje ifm et graveprosjekt.

Kort beskrivelse av oppgaveforslag:
Som landsdekkende aktør innen gravemeldingstjenste i Norge, Sverige og Finland sitter vi med gode data på graveprosjekter og hvilke prosjekter som har gitt skade på eksisterende infrastruktur - graveskade. Vi ønsker å se om vi ved hjelp av maskinlæring kan bruke disse dataene til å forutse om et graveprosjekt vil føre til graveskade og på den måten sette inn tiltak som gjør at sjansen for at det blri graveskader.

Utfyllende kommentarer til hva oppgaven gjelder:
Geomatikkgruppen håndterer cirka 400 000 gravehenvendelser i Norge, Sverige og Finland årlig. Vi mottar informasjon om graveprosjektet fra entreprenør og gir entreprenører informasjon om hvilken infrastruktur som ligger i bakken. Informasjonen gies både ved utsending av infrastrukturkart og ved fysisk opplytting av infrastrukturen i arbeidsområdet.
En graveskade er en skade på eksisterende infrastruktur ifm graving. Dette kan føre til bortfall av strøm, vann eller bredbånd hos sluttbrukere.
Dette påfører oss som samfunn betydelige kostnader hvert år. Graveskader lagres også i vårt system og er knyttet til en gravehenvendelser. Dette er gode testsett for utprøving av AI-modeller.
Det er mye metadata knyttet til en gravehenvendelse. Eksempler på det er tid, utfører, kommentarfelt, type infrastruktur i området, størrelse osv. Vi ønsker å se om man ved bruk av AI kan finne om det er noen sammenheng mellom metadata og sjansen for at infrastruktur blir skadet ifm prosjektet.
Det er også aktuelt å bruke eksterne kilder som f. eks været på skadedagen.

Contents

List of Figures

List of Tables

1 Introduction

Strike damages put a huge toll on the community, so there is a massive incentive to avoid them. Geomatikk have done internal analysis of their data, however they have not managed to reach any conclusions. This is why our assignment is to figure out a way to use machine learning to predict whether a project will have an accident or not.

Machine learning has and is being used frequently in analyzing and interpreting data. It's especially efficient when the amount of data is large. This is why machine learning was chosen as a tool to solve this problem, as there are huge amounts of data available.

Due to the nature of the project where false positives don't really matter, they will be treated as permissible.

This paper will be taking a look at which machine learning models can be used to predict strike damages and which models are most suited to this task.

1.1 Research Question

The title of the Bachelor Thesis is "Machine learning for prevention of strike damage" with the purpose of researching and developing a ML model that will help to assess the risk of excavation projects. In our thesis, we will look at how and what techniques to use to preprocess the data set and then create a ML model to classify the outcome. Some of the challenges that we have to face in preprocessing are how to handle a dataset with mixed data types, imbalanced data, missing values, and huge amounts of data. However, the challenges during the development of the model are to determine which model is the most suitable for our objective and data and which evaluation metrics to use. Choosing evaluation metrics will be done with consideration of Geomatikk's recommendation to use recall as an evaluation metric. This led the team to formulate the following research question:

How can machine learning be used for risk assessment of excavation projects?

1.2 Report Structure

We will begin by presenting general information about machine learning and its intricacies, followed by the methodology used to work on the project. Then we will discuss the project and present our results and conclusion.

This report is structured into the following chapters:

- **Introduction** - This chapter introduces the project. It gives a description of the research question of this paper, acronyms and abbreviations and a glossary.
- **Background and Theory** - This chapter explains relevant theory and gives background to the following chapters.
- **Method and Technology** - This chapter describes the process and methodology behind the planning and execution of the experiments. Furthermore, it gives an overview over choices of technology for this project.
- **Results** - In this chapter the results from the experiments are presented.
- **Discussion** - This chapter discuss the results against the research question.
- **Conclusion and Future work** - In this chapter a conclusion for the work of this paper is presented and it provides suggestions for future work to be done related to this project.
- **Broader impact** - This chapter goes into the environmental, social and ethical impacts of this project.

1.3 Acronyms and Abbreviations

AI = artificial intelligence
ANN = Artificial Neural Network
API = application programming interface
AUC = Area Under The Curve
CUDA = compute unified device architecture
CPU = central processing unit
EMK = Emergency Medical Communication
FP = False Positive
FN = False Negative
GPU = graphics processing unit
KNN = K-Nearest Neighbours
KPI = key performance indicator
ML = machine learning
PR = precision-recall SVM = Support Vector Machine
TP = True Positive
TN = True Negative
WSL = Windows Subsystem for Linux

1.4 Glossary

Strike damage[1] = accidental damage on any type of underground infrastructure, for example, electric cable, water pipe, gas pipe or sewage lines.

[1] source - Private communication with Geomatikk

2 Background and Theory

This chapter will present the theoretical concepts and knowledge behind the methods and techniques that are normally used for similar objectives. Some of the methods presented below were used in this thesis, while others were not, either due to the lack of time or due to being irrelevant in our particular case.

2.1 Artificial Intelligence

According to Stanford professor John McCarthy, [1] one of the founders of the discipline of artificial intelligence, AI is the field of making intelligent machines. and since modern machines run on commands from programs, then AI is by extension, the field of making intelligent computer programs.
Intelligence originally was a trait exclusively for living creatures, but with AI development machines are getting constantly more intelligent and capable. AI is offering improvements and even breakthroughs in many sectors, ranging from medicine[2] to autonomous driving.[3]

While some AI systems are completely based on human reasoning. As in a programmer writing an algorithm to the tune of if X, do y. Others are made to self-learn and improve, which is the field of Machine Learning.

2.2 Machine Learning

Machine learning is the field of building systems that improve themselves with experience. Machine learning is a part of AI, but unless the AI program reliably learns from experience, it is not ML. Professor Tom M. Mitchell from Carnegie Mellon University gave a more formal definition:[4, p. 1]
"...we say that a machine learns with respect to a particular task T, performance metric P, and type of experience E, if the system reliably improves its performance P at task T, following experience E."

In some fields ML is already the best way to develop software. Some of these fields are:

- Big level of complexity: When the solution requires algorithms too complex for a human to realistically make, such as speech recognition and classifying images. It is easy for a human to understand speech, but it is hard to write an algorithm that does that reliably.

- constant adaptation: When software needs to constantly adapt to new information, such as a recommendation system that recommends based on changing spending habits and changing interests. Or spam email filters that have to flag new types of spam email. An email spam filter

from 2010 might have a hard time flagging spam email it has not seen before, like spam about bitcoins.

ML have four main branches as shown in the figure below:

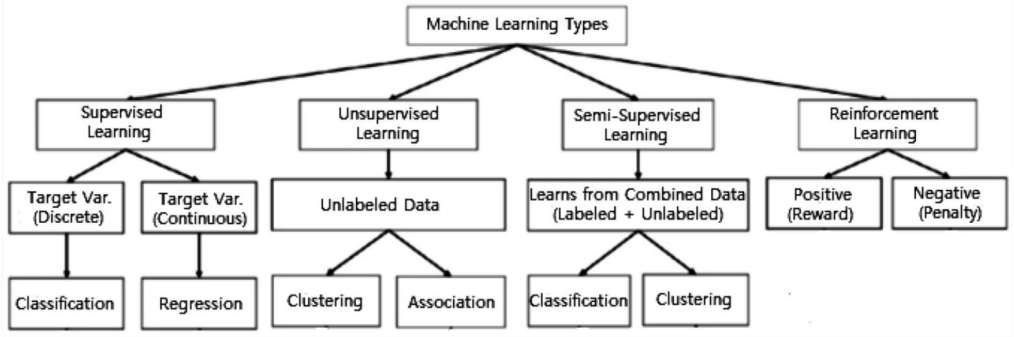


Figure 1: Machine learning branches[5]

2.2.1 Supervised Learning

Supervised learning is typically the task of machine learning to learn a function that maps an input to an output based on sample input-output pairs[5]. Supervised learning algorithms will analyze the training data and then produce an inferred function which is used for mapping new examples. It is the most important methodology in machine learning.

2.2.2 Unsupervised Learning

Unsupervised learning analyzes unlabeled datasets without the need for human interference, i.e., a data-driven process[5]. In unsupervised learning there is no specific output provided, instead one will try to infer some underlying structure from the inputs[6].

2.2.3 Semi-Supervised

Semi-supervised learning can be defined as a hybridization of the above-mentioned supervised and unsupervised methods, as it operates on both labeled and unlabeled data[5]. Semi-supervised learning algorithms will attempt to improve the performance in one the two tasks by utilizing data generally associated with the other[6].

2.2.4 Reinforcement Learning

Reinforcement learning is a type of machine learning algorithm that enables software agents and machines to automatically evaluate the optimal behaviour in a particular context or environment to improve its efficiency[5].

2.3 Machine Learning Models

2.3.1 Random Forest

Random forest is a machine learning method for classification and regression. It is a computationally efficient technique that operates quickly over large datasets[7]. A random forest is comprised of random trees which are drawn at random from a set of possible trees with random attributes at each node. Random trees can be generated efficiently and combined into large sets leading to generally accurate models[7].

Algorithm 1 Basic Random Forest[8, p. 200]

Select the number of models to build, m
for $i = 1\ to\ m$ **do**
  Generate a bootstrap sample of the original data
  Train a tree model on this sample
  for $eachsplit$ **do**
    Randomly select $k(< P)$ of the original predictors
    Select the best predictor among the k predictors and partition the data
  end for
  Use typical tree model stopping criteria to determine when a tree is complete (but do not prune)
end for

2.3.2 Support Vector Machine

A support vector machine is a computer algorithm that learns by example to assign labels to objects[9]. This machine learning algorithm is mostly used for pattern recognition. The SVM algorithm is a supervised type of machine learning algorithm which takes a set of training examples which each belong to one of many categories and builds a model that predicts the category of new examples[10].

2.3.3 K-Nearest Neighbors

The k-nearest neighbours algorithm is a supervised learning method. It is used for classification and regression. In K-NN, a database is searched for the most similar elements to the given query, and the similarity is defined by a distance function[11].

2.3.4 XGBoost

XGBoost(eXtreme Gradient Boosting) is an open-source library which provides regularizing gradient boosting. XGBoost is an implementation of a generalized boosting algorithm that has become extremely popular due to its excellent predictive performance[12]. The drawbacks to XGBoost is that it can be very time consuming to run.

2.3.4.1 Gradient Boosting

Gradient boosting is a very important tool when it comes to supervised learning which provides state of the art performance on classification, regression and ranking tasks[12].

2.3.5 Artificial Neural Network

Artificial neural networks are very popular nowadays and are usually used in deep learning projects. They are trained to solve a particular task. Neurons are organized into groups called layers and connected to each other precisely to form a network[13]. If the number of layers is high, then the ANN is defined as deep.

2.4 Cost-Sensitive Learning

Cost-Sensitive learning is a type of learning in data mining that takes the misclassification costs into consideration[14]. This approach is very common in datasets where the class distributions of data are highly imbalanced.

2.5 Evaluation Metrics

Evaluation metrics help measure the performance of a ML model. There are multiple evaluation metrics to choose from, and the choice of which to use when assessing an ML solution depends on the choice of KPIs.

2.5.1 Confusion matrix

The confusion matrix is one of the most popular methods to measure the efficacy of a classification problem[15]. It is a two-dimensional table visualizing the classification performance of a classifier as seen in Figure 2. It is indexed in one dimension by the true or observed class of an object and by the prediction of the class in the other.

True positives are defined as positive samples that are correctly classified. A true negative occurs when a sample of a negative class is correctly classified by the model. For classification models with two classes, it makes two types of errors. They are false positives and false negatives. False positives, also known as type 1 errors, occur when samples of a negative class have

incorrectly been classified as positive. False negatives, or type 2 errors as they also are known as, occur when a sample of the positive class has been wrongly classified as negative[16].

Predicted	Observed	
	Event	Nonevent
Event	TP	FP
Nonevent	FN	TN

Figure 2: Confusion matrix for binary classification with ”event” and ”non-event”. The cells in the table indicate the number of true positives (TP), false positive (FP), true negative (TN) and false negative (FN)[8]

The evaluation metrics described below are all based on the values from the confusion matrix.

2.5.2 Accuracy score

Accuracy is defined by

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{1}$$

and is a representation of how many predictions were made correctly[15].

2.5.3 Precision

Precision is defined as

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

and is the ratio of true positives divided by the total number of positives predicted[16].

2.5.4 Recall

Recall is defined as of the total of all positive events, how many were correctly classified, and is given by the formula

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

[15]

2.5.5 F1-Score

F1-Score is a combination of precision and recall and solves the dilemma of having to choose the KPI (i.e., higher recall or higher precision) when comparing models[15]. It is defined as

$$F1\ Score = \frac{2 * (Precision * Recall)}{Precision + Recall} \tag{4}$$

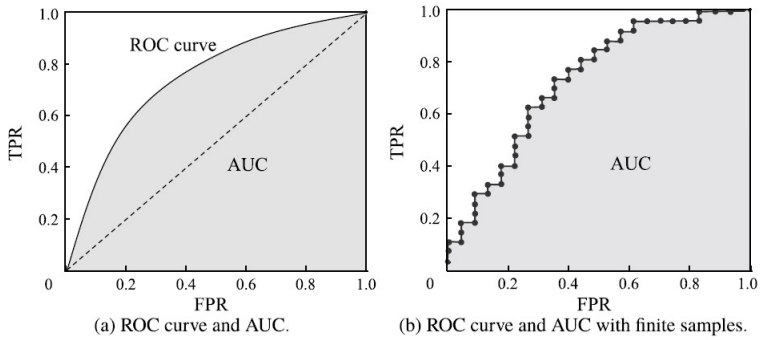
and is the harmonic mean of precision and recall, which is a kind of numerical average. This means that a model with a high F1-Score has both good precision and good recall[16].

2.5.6 ROC & AUC

Verdhan et al.[15] define the receiver operating characteristic or ROC as a plot between TPR (true positive rate) and FPR (false positive rate), see Figure 3a for an example, where TPR is the same as recall 3, and FPR is calculated using the respective formula

$$FPR = \frac{FP}{TN + FP} \tag{5}$$

In real life, there are finite pairs of TPR and FPR coordinates to plot the ROC curve, and might not look as smooth as Figure3a. As the curve is only an approximation, it will realistically look more like the plot in Figure3b.



(a) ROC curve and AUC. (b) ROC curve and AUC with finite samples.

Figure 3: Example of graphs illustrating ROC curve and AUC[17, p. 38]

We can generally say that a model is better than another if it totally encloses the other model's ROC curve. in the case where there are intersections between the curves, no model is generally better than the other. A way of evaluating the performance of two different models when the ROC curve is overlapping is by using the Area Under the ROC curve (AUC). It is illustrated in Figure3a by the shaded area under the ROC curve. AUC is approximated

by the formula stated in [17] as

$$AUC = \frac{1}{2}\sum_{i=1}^{m-1}(x_{i+1} - x_i)(y_i + y_{i+1}) \tag{6}$$

where $x_1 = 0$ and $x_m = 1$, and is calculating the sum of areas between each step along the curve as illustrated in Figure 3b. [17, pp. 36–38]

2.6 Precision-Recall AUC

Precision-Recall AUC is the calculated area under a curve, where the curve is defined by a trade-off between different aspects of performance as the threshold applied to the models predictions varies[18]. PR-AUC varies on a scale from zero to one with random performance equal to sample prevalence in the focal dataset[18].

2.7 Cross-validation

Cross-validation is a data resampling technique used to access the generalization ability of predictive models and to prevent overfitting [19]. The technique works by splitting the data into K stratified subsets. then it trains the model k times using k-1 subsets and tests on the last subset. Each subset gets used as a test set once. the average of the k testing scores is used as the evaluation result.

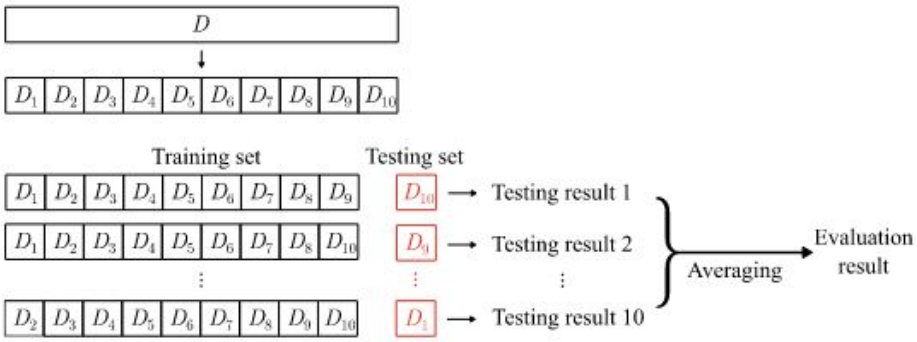


Figure 4: Illustration showing data splitting for 10 fold cross validation.[17, p. 29]

2.8 Re-sampling techniques

When you are dealing with an imbalanced dataset, your results are likely to be inaccurate. Which is why there are multiple re-sampling methods available. There are three methods of re-sampling, you have oversampling, undersampling and methods which combine the two.

2.8.1 Oversampling

Oversampling is generally the most frequently used re-sampling method. Oversampling works by raising the weight of the minority class by either replicating minority class samples, or creating new ones. There are multiple different oversampling methods. Here we will go over two of the most popular ones.

- **RandomOverSampler:** This method will increase the size of the dataset by replicating existing samples. The main point of this method is that it does not create new samples, and the variety of the samples does not change [20].
- **SMOTE:** This is a method that increases the number of minority samples by generating new samples. The algorithm takes samples from the feature space for each target class and its nearest neighbours, and then takes the features from target case and its neighbours and combines them into new samples [20]. New samples generated from this method are not copies of existing samples.

2.8.2 Undersampling

Undersampling works by removing samples from the majority class until the dataset is balanced enough. When a dataset is very imbalanced, its usually realistic to assume that many samples are redundant. However there is always a risk of removing relevant data [21].

- **RandomUnderSampler:** This method will reduce the size of the dataset by removing samples from the majority class by random. This is the risky part of using undersampling as you may lose critical data.
- **EditedNearestNeighbours:** This method applies a nearest-neighbours algorithm and edits the dataset by removing samples that do not agree ”enough” with their neighbourhood [22].

2.8.3 Combined Sampling

Oversampling and undersampling each have different advantages and disadvantages. By combining the two, you can gain the benefits and drawbacks of both methods [20].

- **SMOTEENN** This is one of the most used resampling methods which combines SMOTE as the oversampling model, and ENN(EditedNearestNeighbour) as the undersampling model.

2.9 Feature engineering

Feature engineering is the process of converting raw observation into useful features to be used in ML. It uses statistical techniques to achieve it’s goals.

2.9.1 Dummy Variables Encoding

It is an encoding technique which maps each unique category to a vector that has binary values 1 or 0. The vector contains the 1 if the instance is a part of this category and 0 if it is not.[23]

2.9.2 Mean Target Encoding

It is an encoding method where each category **C** in the feature **X** is replaced by the a ratio **R** calculated with regards to the target **Y**. In our case the target feature is binary $Y \epsilon [0, 1]$. The ratio for binary targets is the number of positive instances of to the total number of instances of the specified category. [24]

$$R = \frac{number\ of\ instances\ where\ X = C\ and\ Y = 1}{number\ of\ instances\ where\ X = C} \tag{7}$$

3 Method and Technology

This chapter is based on the theory presented in Chapter 2 and provides an overview of the methodology behind the experiments conducted. It also covers the choice of technologies used to execute the experiments.

3.1 Process

In this section, we describe the process prior to the implementation of machine learning models and the resulting experiments.

3.1.1 Data

The data used in this project were provided by Geomatikk AS and contained a large amount of raw data from more than 400 000 previous inquiries. It contained features with categorical and numerical data.

The dataset contained sensitive information and prevented us from bringing it outside of their offices. Due to the current situation regarding Covid-19 restrictions at the beginning of the project we were unable to access the real dataset. Therefore, they also provided a smaller demo dataset containing fake data that replicated the format of the real dataset. We used this to familiarize ourselves with the structure of the data and the data types. Throughout the early phases of the project, we continuously discussed the dataset with Geomatikk and we ended up with iterating through a few different versions of the dataset as things were discovered while analyzing the data.

Short summary of the features:

- **henvendelse_id**: a unique id for each inquiry.

henvendelse_id	pavist_dato	graveskade_dato	hastesak
1	03.04.2019 00:00		USANN
2	27.06.2021 11:22	28.06.2021 15.20	SANN
3	07.07.2021 12:10		SANN
4	07.07.2021 12:13	15.07.2021 10.15	SANN
5	07.07.2021 12:22		SANN

gravedybde	kommune_id	organisasjon_id	blindarbeid
2	319	348	USANN
10	64	12020	USANN
0.75	72	14206	USANN
3	72	14207	USANN
3	64	673	USANN

grunnboring	berorte_fagomraader	areal
USANN	{”(f,f,t,f,f,f,f,f,f,f,f)”}	400
USANN	{”(t,f,f,f,f,f,f,f,f,f,f)”}	50
USANN	{”(t,f,f,f,f,f,f,f,f,f,f)”}	85
USANN	{”(t,f,f,f,f,f,f,f,f,f,f)”}	150
USANN	{”(t,f,f,f,f,f,f,f,f,f,f)”}	1300

polygon
POLYGON((7041827.20 270321.45, 7041817.10 270360.15, 7041799.85 270334.20, 7041827.20 270321.45))
POLYGON((7041827.20 270321.45, 7041817.10 270360.15, 7041799.85 270334.20, 7041827.20 270321.45))
POLYGON((7041827.20 270321.45, 7041817.10 270360.15, 7041799.85 270334.20, 7041827.20 270321.45))
POLYGON((7041827.20 270321.45, 7041817.10 270360.15, 7041799.85 270334.20, 7041827.20 270321.45))
POLYGON((7041827.20 270321.45, 7041817.10 270360.15, 7041799.85 270334.20, 7041827.20 270321.45))

har_graveskade	paaviser	har_maskinkontroll
0	1	0
1	1	0
0	2	1
1	3	0
0	4	0

skadepunkt	netteier_ids
	{3010,3020,3030,3040,3050}
POINT(270266.5 7041828.325)	{7340}
	{7340}
POINT(270266.5 7041828.325)	{3010}
	{7340}

Table 1: Example of the dataset with columns and their values. This table only contains dummy data

- **paavist_dato**: the date and time of when the Geomatikk field agent went on a site inspection.
- **graveskade_dato**: the date and time of when the strike damage occurred.
- **hastesak**: Boolean shows whether the project had to be addressed quickly.
- **gavedypde**: the depth of the excavation/drilling.
- **kommune_id**: the id of municipality where the project takes place.
- **organisasjon_id**: the id of the contractor which submitted the project.
- **blindarbeid**: Boolean shows when the project have drilling under the ground.
- **grunnboring**: Boolean shows if the project is a drilling project.
- **berorte_fagomraader**: a list of Boolean that shows which infrastructure type the project is working on. The list is: VA_Nett, El_nett, Tele_fiber_kabel_tv, Fjernvarme, Grunnboring, Annet, Gassledning, Vei, Gjerde, Autovern, Bygg_anlegg, Drenering.
- **areal**: the total area of the project site.
- **polygon**: UTM Zoone 33N coordinates that shows a polygon representation of the project site.
- **har_gravskade**: Boolean that shows whether there was a strike damage.
- **paaviser**: id of Geomatikk field agent that went on site inspection.
- **har_maskinkontroll**: Boolean that shows whether the project will use machines that utilize coordinates for extra precision.
- **skadepunkt**: the UTM Zoone 33N coordinate of where the strike damage occurred.
- **netteier_ids**: a list of ids of the infrastructure within a certain radius of the project site.

3.1.2 Research

Following the acquisition of the data, we continued to look for relevant research that we could use as a base for the experiments. Geomatikk informed us that no previous research had been done in this field[2]. However, we did our own literary search and were unable to find any research directly related to the use of machine learning to make risk assessments on the chance of

[2]Private communication with Geomatikk

strike damage. We therefore had to resolve to research within related fields, with the same type of problem to solve, e.g. disease prediction or fraudulent credit card prediction. The research phase was concentrated around the beginning of the project, but some research was done later, if new topics of relevance was discovered.

3.1.3 Data Analysis

Due to the large amount of raw data, and to get better knowledge of it, conducting a data analysis was necessary. The analysis provided insight into the distribution of the values in the different columns. This gave us knowledge about what numerical features had to be standardized or normalized. It also provided information about which features contained categorical data and the distribution of it. This helped in the decision of whether to have one hot encoding or if some other feature engineering should be applied. During the analysis, it was discovered that the target classes were very heavily imbalanced, with a distribution of 97.68% of inquires with out strike damage and 2.32% with strike damage. The distribution is illustrated in Figure 5. This resulted in having to take more consideration when picking out models we wanted to implement in the experiments, and having to explore different options into how dealing with imbalances datasets.

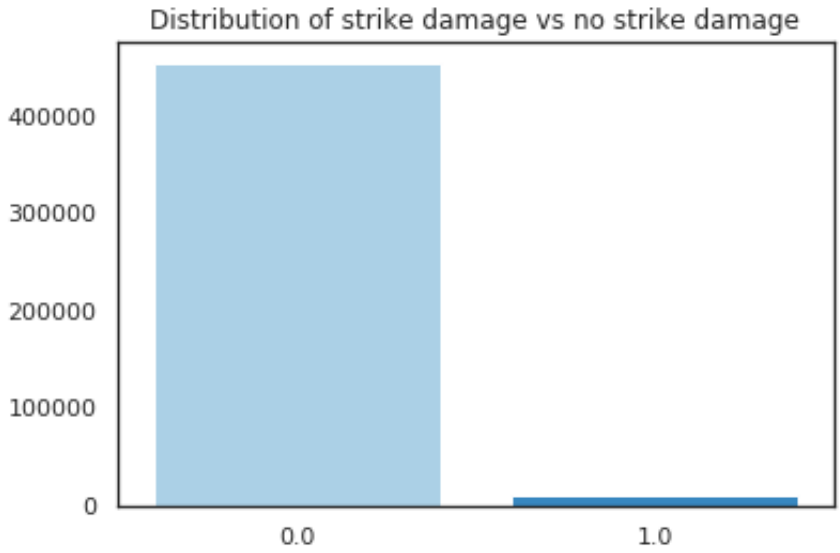


Figure 5: Distribution of target classes

Table 1 illustrates an excerpt of how the raw dataset look like. It is a dataset containing fabricated data, but with the same structure and data formats

as the original dataset. Each row contains information about an inquiry received by Geomatikk, where ”har_graveskade” is the target class, and the other columns are potential features.

Figure 6 shows the distribution of area sizes per inquiry. From this figure we can see the existence of outliers, and that the area feature should be scaled.

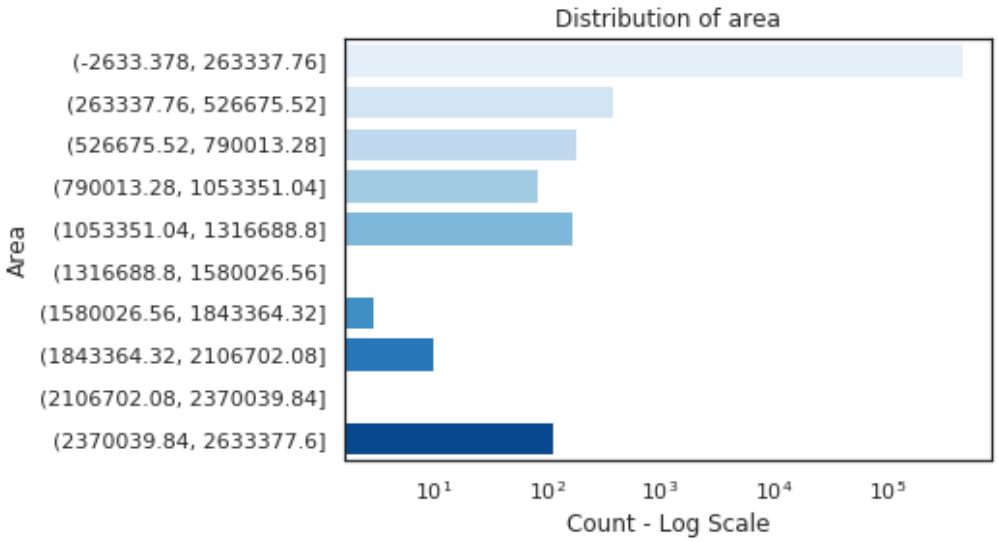


Figure 6: Distribution of site area sizes.

Figures 7, 8, and 9 show the disruption inquires per field agents, municipality, and organisation/contractor respectively. the figures show what is expected, that inquires are not distributed evenly. This will have an implication on which encoding technique we use for these categorical features.

Figure 10 shows the distribution of the number of infrastructures around project sites. It might prove useful to make a feature of this information.

After further analysis, it was discovered that some of the features were not useful for our case. These feature were *henvendelse_id*, *gravskade_dato*, and *polygon*. *henvendelse_id* is a unique id for each project. *gravskade_dato* and *polygon* where not used. It was also discovered that *grunnboring* was redundant as it was also available in *berorte_fagomraader*.

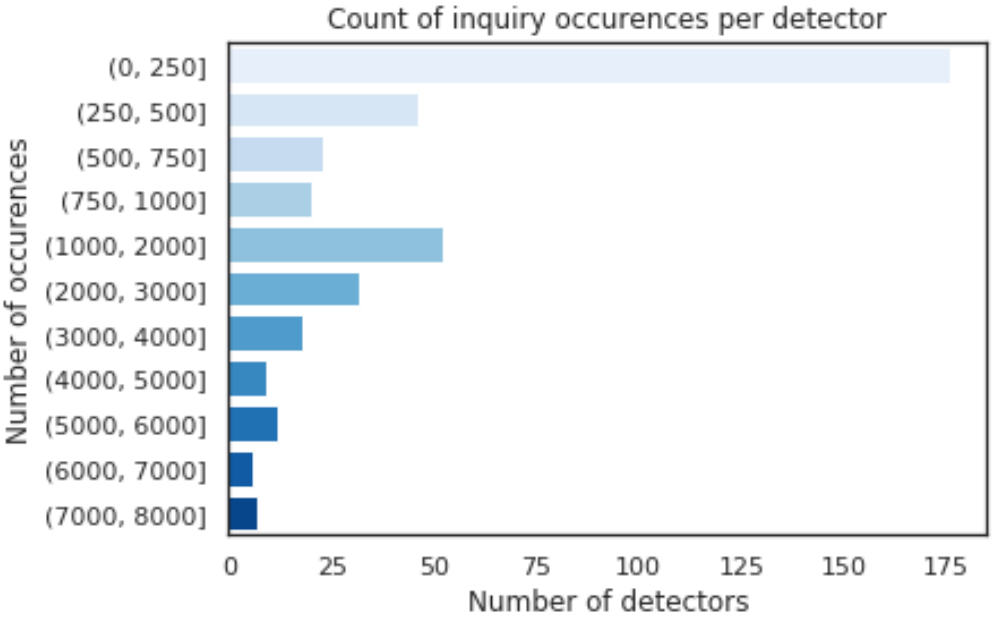


Figure 7: Distribution inquires per field agent.

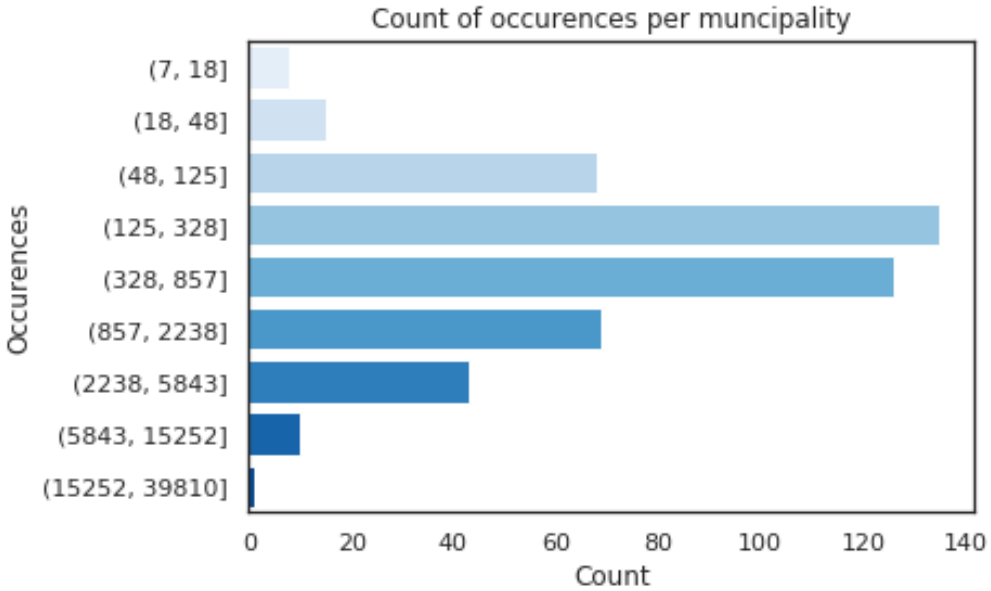


Figure 8: Distribution of inquires per municipality.

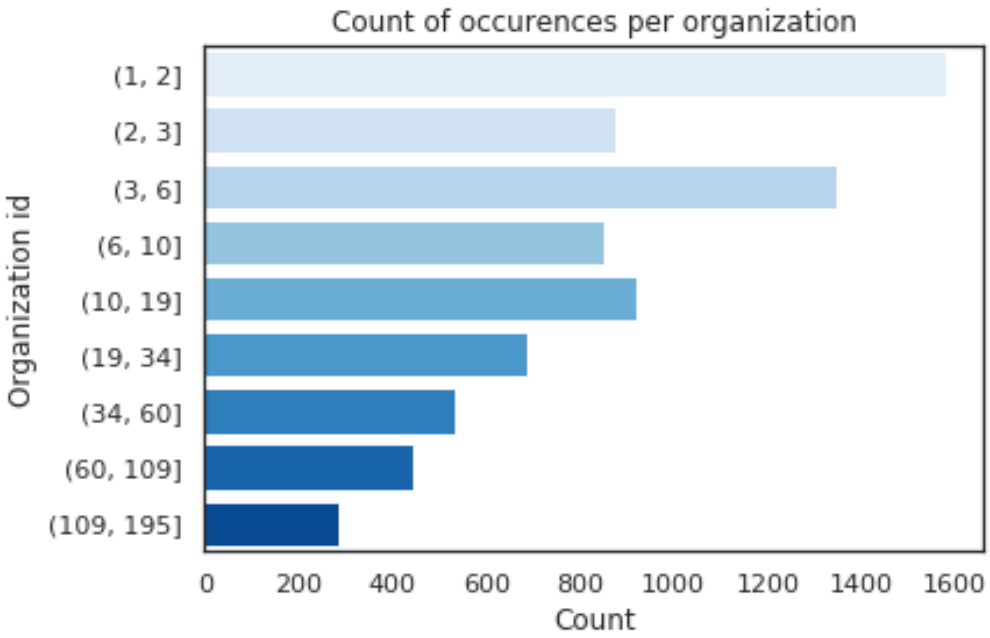


Figure 9: Distribution of inquires per organisation.

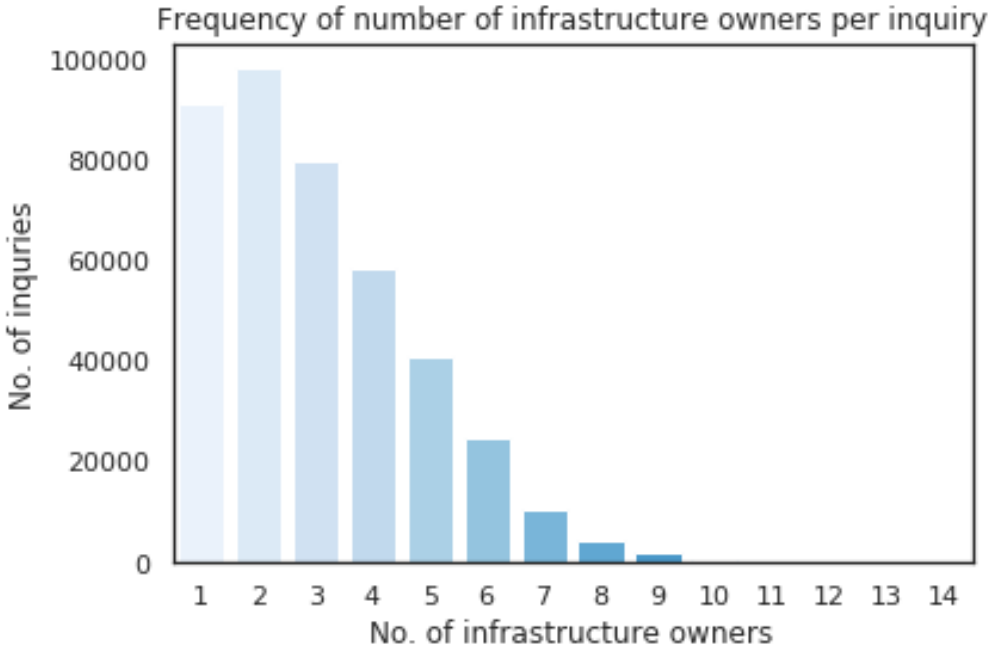


Figure 10: disruption of number of infrastructures around project sites.

3.2 Execution

This chapter will explain how the experiments were implemented in this project. It will also explain the scientific method that was followed.

Figure 11 shows a simplified depiction of the process steps throughout this project.

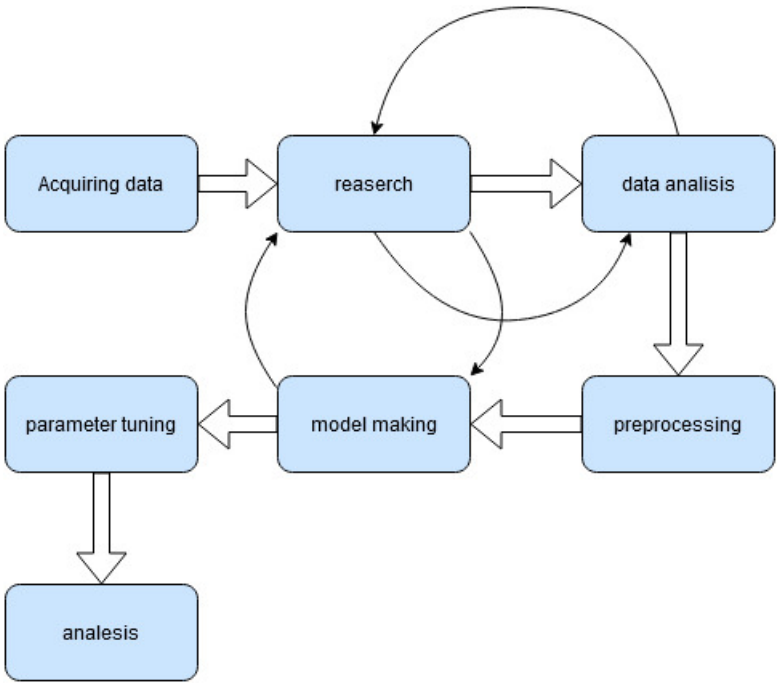


Figure 11: Simplified overview of the process steps throughout the project.

3.2.1 Experimentation Process

During the planning phase, the team chose to follow the hypothetico-deductive method illustrated in figure 12 during the execution phase.

The hypothetico-deductive model is used to scientifically explore a research question. The model follows the steps below:

- Make an observation.
- Use the observation to make a research question
- Formulate a hypothesis built on the observation and research question. Where the hypothesis should be falsifiable.
- Deduce predictions based on the hypothesis.

- Perform experiments around the predictions.
- Collect the result of the experiments.
- Evaluate the hypothesis in light of the results obtained.

This model is usually run in cycles where after every evaluation you return to the hypothesis step, make a new prediction and run tests to try and falsify the prediction.

The experimentation made during the project aimed to utilize the Hypotheico-deductive method to gain empirical support for our hypothesis. The method was helpful in structuring the experiments, especially since we had a broad research question around *"how"* to use machine learning in this special case. We tried to formulate hypotheses around the different ML models and preprocessing techniques, and then falsify them. The experiments were conducted and the results were logged and evaluated in order to come to a conclusion in the end.

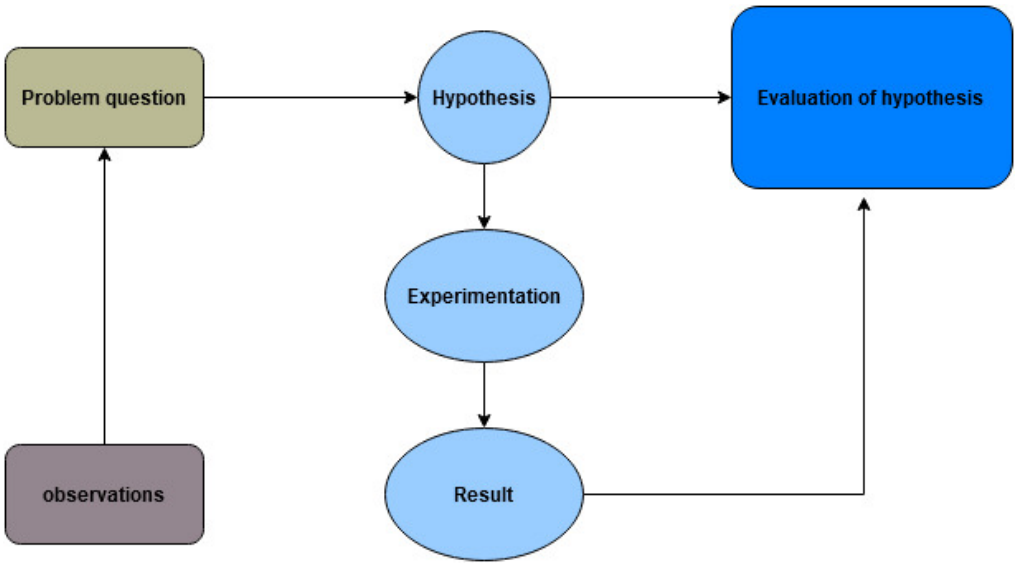


Figure 12: Hypothetico-deductive model Illustration. [25]

3.2.2 Data Prepossessing

As stated in subsection 3.1.3 we had a big dataset containing a backlog of project inquiries. The dataset had a variety of features. see table 1.

Features *henvendelse_id*, *gravskade_dato*, and *polygon*, and the redundant feature *grunnboring* were dropped from the dataset. There were also a relatively small number of instances that had missing values. These instances were also dropped.

Feature extraction and engineering techniques were implemented to extract some useful features and improve them:

- From feature *pavist_dato*, 19 binary features were extracted. 7 representing which day of the week the field agent inspected the site, and 12 represented which month the inspection took place.
- From feature *netteier_ids* a new continuous feature was extracted. the feature represented the number of infrastructures within a certain radius from the project site.
- Feature *berorte_fagomraader* was split into 12 binary features representing the 12 infrastructure types.
- Four continuous feature representing count of inquiries for each category were extracted from the categorical features *kommune_id*, *organisasjon_id*, *paaviser*, and *netteier_ids*. see table 2 which shows where they were used.
- Binary features *hastesak*, *blindarbeid*, and *grunnboring* which were represented in strings *"SANN"/"USANN"* were replaced by number representation *"0"/"1"*.
- Feature encoding: Both Target Mean Encoding and Dummy Variable Encoding were used on the categorical variables in the dataset. See table 2.
- Scaling: scaling technique was used on feature *areal*. The count features see table 2 were scaled as well.

Resampling techniques were used to mitigate the effect of data imbalance in the dataset. SMOTEEN Combined sampling technique resampled the training set of each dataset. The resampled training set was fed to the model.

Due to the high number of unique values in the four categorical features *kommune_id*, *organisasjon_id*, *paaviser*, and *netteier_ids*. We decided to make four datasets. Each one has different variation of dummy variable encoding and/or mean target encoding. see table 2.

3.2.3 Training Models

Six ML model types were used in this project. The start was mainly with simpler models k-nearest neighbors, and support vector machine. Then we experimented with progressively more complex models. Mainly random forest, gradient boosted trees, and neural network.

Each model's hyper-parameters were tuned to get the best result possible.

Feature	**Dataset**			
	EXCLUDED	RATIO	TOTAL_RATIO	KOMMUNE_DUMMIES
netteier_id	DVE	DVE	MTE & C	DVE
kommune_id	dropped	MTE & C	MTE & C	DVE
paaviser	dropped	MTE & C	MTE & C	MTE & C
organisasjons_id	dropped	MTE & C	MTE & C	MTE & C
Feature count	407	413	46	887

Table 2: Table of the datasets used in the project. Included is the encoding and the total number of features. Where: **DVE**: Dummy variable encoding. **MTE**: Mean target encoding. **C**: Count of inquiries for each category. **dropped**: The feature was dropped from the dataset.

Cost-sensitive training was implemented for the models that supported it. along with cost-insensitive training with resampled data. Table [3] shows the model count. Each ML model type except neural network was trained on four parameter combinations for each dataset. The aim was to get the best possible result for each of the evaluation metrics. Neural network on the other hand only changed the structure depending on the shape of the input feature vector. This lead to neural network only having 8 models, one for each dataset.

3.2.3.1 Hyperparameter tuning

Parameter tuning was implemented by supplying a list of values for each hyper-parameter and then iterating over parameter value combinations. All the models were trained on every combination of hyperparameters. Throughout the tuning process all increases in evaluation metric scores were logged along with the hyper-parameters that lead to the increase. The results were logged and evaluated afterward. Parameter tuning was done for both cost-sensitive and cost-insensitive learning.

3.2.3.2 Cost-sensitive training

Cost-sensitive training was used with XGBoost classifier, XGBoost random forest, and neural network model. This training was done with the original non-resampled datasets. The models take the class weight as a parameter. the class weight parameter was also included in the hyper-parameter tuning phase. the results were logged and evaluated. The support vector machine model supported cos-sensitive learning, but we ran into issues because of hardware limitations. Thus, SVC cost-sensitive learning was not included in this report.

models	Dataset		No. of models for every evaluation metric on every dataset
	Resampled	Not resampled	
KNN	4	0	16
Random Forest	4	0	16
SVC linear	4	0	16
XGBoost Classifier	4	4	32
XGBoost Random Forest	4	4	32
Neural Network	4	4	8
Total no. of mdoels			120

Table 3: table showing the total number of models, their types and the dataset they were trained on.

3.2.4 Analyzing model performance

Model performance evaluation began with evaluating best scores for all the model from hyperparameter tuning phase. Analyzing these scores was done manually afterward. We went with Geomatikk's recommended evaluation metric "recall" as the main metric to focus on, along with other useful evaluation metrics such as "precision", "f1", "ROC AUC", and "accuracy scores". After evaluating the best scores, we built a total number of 120 model each with different hyper-parameters or structure. See table 3.2.3.

3.3 Roles and Work Distribution

Shawish and Blichfeldt did all the work related to the coding in this project. They also contributed to most section of this report. Bakken contributed with parts of documentation throughout this project. In this paper he provided to sections of the theory chapter and some minor contributions elsewhere.

3.4 Choice of technology

In this section we will provide an overview of the technologies used to implement the experiments described in Chapter 4

3.4.1 Hardware

We were first provided with a stationary computer, but it was quickly replaced by a HP EliteBook 840 G6 laptop, as the Razor Core X was only compatible with Thunderbolt 3, and the stationary did not have one.

The laptop had 512 GB of disk space and contained at first 16GB of ram, but we ran out of memory trying to run the models, and was later provided another 16GB, in total 32GB.

3.4.1.1 CUDA-compatible GPU

Geomatikk provided a Razor Core X with a CUDA-compatible NVIDIA GeForce GTX 1070 GPU. This made it possible to write code using libraries, like Rapids, that make it run in parallel on the GPU instead of the CPU, thus drastically increasing the processing speed of the ML models.

3.4.2 Operating system

The team was able to choose the operating system on the laptop provided by Geomatikk. In the beginning the team decided to go for Windows 11 as it was the one most used by all. Later on, after encountering issues with running the Rapids-library in Windows and on WSL, the team decided to migrate to Ubuntu 20.04.

3.4.3 Programming language and libraries

The team chose Python 3 as programming language as it is widely used in machine learning and has an extensive set of libraries related to the field of ML. It was also convenient that it is easy to use, and the team had prior knowledge.

3.4.3.1 Pandas

Pandas[3] is an open-source library that provides simple and straightforward data structures and data analysis tools. It was used to preform data preprocessing and data manipulations on the dataset.

3.4.3.2 Rapids

Rapids[4] is a collection of open-source software libraries that is compatible with CUDA. The Rapids CuML and CuDF API, which we mainly used, mirrors the scikit-learn and Pandas libraries. This made it very convenient to use, as we were already familiar with them.

3.4.3.3 XGBoost

XGBoost[5] is a library that implements ML algorithms under the Gradient Boosting framework, and provides optimized distributed gradient boosting. This library is also compatible with CUDA-capable GPUs.

[3] Pandas - https://pandas.pydata.org/docs/index.html#
[4] Rapids - https://rapids.ai/index.html
[5] XGBoost - https://xgboost.readthedocs.io/en/stable/

3.4.3.4 TensorFlow

TensorFlow[6] is an open source end-to-end platform for developing and training ML models. It provides a variety of tools, ready-to-use datasets and libraries. TensorFlow was chosen to develop the neural network as members of the team had previous experience using it.

3.4.3.5 Keras

Keras[7] is a library that runs on top of TensorFlow, providing a deep learning API. It was developed with an emphasis on enabling fast experimentation and being simple, flexible, and powerful to use. This API was chosen by the team as it is simple to use and some of the members had previous experience using it.

3.4.3.6 Scikit-learn

Scikit-learn[8] is an open-source machine learning library. It was used to get metrics from the ML models and to optimize parameters of all but the neural network model.

3.4.3.7 Imbalanced-learn

Imbalanced-learn[9] is a free library, providing tools for handling classification with imbalanced classes. The library is relying on and is fully compatible with scikit-learn. It was used to run different re-sampling techniques on the dataset.

3.4.3.8 Jupyter Notebook

Jupyter Notebook[10] enables the user to combine code with rich text, and was useful for us to visualize and comment on statistics from analysing the dataset.

3.4.3.9 Mathplotlib

Mathplotlib[11] is a library for visualizing different plots, static and animated. This is a well-known and free library and was used to visualize results from the data analysis, to better view statistics of the raw data.

3.4.3.10 Seaborn

Seaborn[12] is a Python library for statistical data visualization and is based on mathplotlib. It is easy to use because of its high-level interface and makes it straightforward to produce visually pleasing graphs without too much work.

[6] TensorFlow - https://www.tensorflow.org/
[7] Keras - https://www.tensorflow.org/api_docs/python/tf/keras
[8] Scikit-learn - https://scikit-learn.org/stable/
[9] Imbalanced-learn - https://imbalanced-learn.org/stable/
[10] Jupyter - https://jupyter.org/
[11] Mathplotlib - https://matplotlib.org/
[12] Seaborn - https://seaborn.pydata.org/index.html

3.4.4 Hyperparameters and Structure

Model structure and hyperparameters are separated into eight tables, two for each dataset, and four figures which show the neural network model structure for each dataset. Tables 13, 14, 15, and 16 show hyper-parameters for the resampled datasets models. Tables 17, 18, 19, and 20 show hyper-parameters for the *non* resampled datasets models. Figures 21, 22, 23, and 24 show the neural network structure for the 4 datasets.

Dataset: Resampled Excluded

models		hyper-parameters
KNN	F1	{n_neighbors': 5}
	Rec	{n_neighbors': 30}
	Pre	{n_neighbors': 20}
	ROC	{n_neighbors': 30}
Random Forest	F1	{max_depth': 19, 'max_features': 'auto', 'max_samples': 0.4, 'n_estimators': 250, 'split_criterion': 0}
	Rec	{max_depth': 7, 'max_features': 'log2', 'max_samples': 1.0, 'n_estimators': 400, 'split_criterion': 0}
	Pre	{max_depth': 19, 'max_features': 'auto', 'max_samples': 0.6000000000000001, 'n_estimators': 400, 'split_criterion': 0}
	ROC	{max_depth': 16, 'max_features': 'auto', 'max_samples': 0.8, 'n_estimators': 100, 'split_criterion': 0}
SVC linear	F1	{C': 1, 'kernel': 'linear'}
	Rec	{C': 10, 'kernel': 'linear'}
	Pre	{C': 1, 'kernel': 'linear'}
	ROC	{C': 100, 'kernel': 'linear'}
XGBoost Classifier	F1	{colsample_bytree': 1, 'eval_metric': 'logloss', 'learning_rate': 0.2, 'max_depth': 6, 'min_child_weight': 5, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	Rec	{colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 3, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	Pre	{colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.3, 'max_depth': 10, 'min_child_weight': 3, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	ROC	{colsample_bytree': 1, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 5, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
XGBoost Random Forest	F1	{colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 256, 'max_depth': 10, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	Rec	{colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 128, 'max_depth': 7, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	Pre	{colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 256, 'max_depth': 10, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	ROC	{colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 256, 'max_depth': 10, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
Neural Network		Learning rate : decaying on rate (1e-3 * 10 ** (epoch / 30)), Early stopping: monitor "val_loss" mode "min" patience "5", batch size: "2048", epochs: "25"

Figure 13: hyper-parameters for resampled EXCLUDED dataset models.

Dataset: Resampled TOTAL_RATIO

models		hyper-parameters
KNN	F1	{n_neighbors': 5}
	Rec	{n_neighbors': 30}
	Pre	{n_neighbors': 20}
	ROC	{n_neighbors': 30}
Random Forest	F1	{'max_depth': 19, 'max_features': 'auto', 'max_samples': 0.6, 'n_estimators': 400, 'split_criterion': 0}
	Rec	{'max_depth': 7, 'max_features': 'auto', 'max_samples': 0.6000000000000001, 'n_estimators': 100, 'split_criterion': 1}
	Pre	{'max_depth': 19, 'max_features': 'auto', 'max_samples': 1.0, 'n_estimators': 250, 'split_criterion': 0}
	ROC	{'max_depth': 10, 'max_features': 'auto', 'max_samples': 0.2, 'n_estimators': 100, 'split_criterion': 0}
SVC linear	F1	{C': 1, 'kernel': 'linear'}
	Rec	{C': 100, 'kernel': 'linear'}
	Pre	{C': 1, 'kernel': 'linear'}
	ROC	{C': 1, 'kernel': 'linear'}
XGBoost Classifier	F1	{'colsample_bytree': 1, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 5, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	Rec	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 5, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	Pre	{'colsample_bytree': 1, 'eval_metric': 'logloss', 'learning_rate': 0.3, 'max_depth': 6, 'min_child_weight': 1, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	ROC	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 5, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
XGBoost Random Forest	F1	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 128, 'max_depth': 10, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	Rec	{'colsample_bynode': 0.75, 'eval_metric': 'logloss', 'max_bin': 256, 'max_depth': 7, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.9, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	Pre	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 128, 'max_depth': 10, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	ROC	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 128, 'max_depth': 7, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
Neural Network		Learning rate : decaying on rate (1e-3 * 10 ** (epoch / 30)), Early stopping: monitor "val_loss" mode "min" patience "5", batch size: "2048", epochs: "25"

Figure 14: hyper-parameters for resampled TOTAL_RATIO dataset models.

Dataset: Resampled RATIO		
models		**hyper-parameters**
KNN	F1	{n_neighbors': 5}
	Rec	{n_neighbors': 30}
	Pre	{n_neighbors': 20}
	ROC	{n_neighbors': 30}
Random Forest	F1	{'max_depth': 19, 'max_features': 'auto', 'max_samples': 0.4, 'n_estimators': 250, 'split_criterion': 0}
	Rec	{'max_depth': 7, 'max_features': 'log2', 'max_samples': 0.2, 'n_estimators': 250, 'split_criterion': 1}
	Pre	{'max_depth': 19, 'max_features': 'auto', 'max_samples': 0.4, 'n_estimators': 250, 'split_criterion': 0}
	ROC	{'max_depth': 10, 'max_features': 'auto', 'max_samples': 0.6000000000000001, 'n_estimators': 400, 'split_criterion': 0}
SVC linear	F1	{C': 1, 'kernel': 'linear'}
	Rec	{C': 100, 'kernel': 'linear'}
	Pre	{C': 0.1, 'kernel': 'linear'}
	ROC	{C': 1, 'kernel': 'linear'}
XGBoost Classifier	F1	{'colsample_bytree': 1, 'eval_metric': 'logloss', 'learning_rate': 0.3, 'max_depth': 10, 'min_child_weight': 3, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	Rec	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 5, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	Pre	{'colsample_bytree': 1, 'eval_metric': 'logloss', 'learning_rate': 0.2, 'max_depth': 10, 'min_child_weight': 1, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	ROC	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 5, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
XGBoost Random Forest	F1	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 128, 'max_depth': 10, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	Rec	{'colsample_bynode': 0.75, 'eval_metric': 'logloss', 'max_bin': 256, 'max_depth': 7, 'n_estimators': 400, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	Pre	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 128, 'max_depth': 10, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	ROC	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 128, 'max_depth': 7, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'subsample': 0.9, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
Neural Network		Learning rate : decaying on rate (1e-3 * 10 ** (epoch / 30)), Early stopping: monitor "val_loss" mode "min" patience "5", batch size: "2048", epochs: "25"

Figure 15: hyper-parameters for resampled RATIO dataset models.

Dataset: Resampled RATIO

models		hyper-parameters
KNN	F1	{n_neighbors': 5}
	Rec	{n_neighbors': 10}
	Pre	{n_neighbors': 10}
	ROC	{n_neighbors': 10}
Random Forest	F1	{'max_depth': 19, 'max_features': 'auto', 'max_samples': 0.4, 'n_estimators': 400, 'split_criterion': 0}
	Rec	{'max_depth': 7, 'max_features': 'log2', 'max_samples': 1.0, 'n_estimators': 100, 'split_criterion': 1}
	Pre	{'max_depth': 19, 'max_features': 'auto', 'max_samples': 0.2, 'n_estimators': 100, 'split_criterion': 0}
	ROC	{'max_depth': 10, 'max_features': 'auto', 'max_samples': 0.6000000000000001, 'n_estimators': 400, 'split_criterion': 0}
SVC linear	F1	{C': 1, 'kernel': 'linear'}
	Rec	{C': 10, 'kernel': 'linear'}
	Pre	{C': 1, 'kernel': 'linear'}
	ROC	{C': 100, 'kernel': 'linear'}
XGBoost Classifier	F1	{'colsample_bytree': 1, 'eval_metric': 'logloss', 'learning_rate': 0.3, 'max_depth': 8, 'min_child_weight': 5, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	Rec	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 5, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	Pre	{'colsample_bytree': 1, 'eval_metric': 'logloss', 'learning_rate': 0.3, 'max_depth': 10, 'min_child_weight': 5, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	ROC	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 3, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
XGBoost Random Forest	F1	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 128, 'max_depth': 10, 'n_estimators': 400, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	Rec	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 256, 'max_depth': 7, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.9, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	Pre	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 128, 'max_depth': 10, 'n_estimators': 400, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
	ROC	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 256, 'max_depth': 7, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}
Neural Network		Learning rate : decaying on rate (1e-3 * 10 ** (epoch / 30)), Early stopping: monitor "val_loss" mode "min" patience "5", batch size: "2048", epochs: "25"

Figure 16: hyper-parameters for resampled KOMMUNE_DUMMIES dataset models.

Dataset: NON Resampled EXCLUDED

models		hyper-parameters	no. CV-folds
XGBoost Classifier	F1	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.3, 'max_depth': 10, 'min_child_weight': 1, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'scale_pos_weight': 43, 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	Rec	{'colsample_bytree': 1, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 3, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'scale_pos_weight': 80, 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	Pre	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 5, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'scale_pos_weight': 1, 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	ROC	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 5, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'scale_pos_weight': 43, 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
XGBoost Random Forest	F1	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 256, 'max_depth': 10, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'scale_pos_weight': 43, 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	Rec	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 256, 'max_depth': 7, 'n_estimators': 400, 'predictor': 'gpu_predictor', 'scale_pos_weight': 80, 'subsample': 0.9, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	Pre	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 128, 'max_depth': 10, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'scale_pos_weight': 43, 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	ROC	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 256, 'max_depth': 10, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'scale_pos_weight': 43, 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
Neural Network		Learning rate : decaying on rate (1e-3 * 10 ** (epoch / 30)), Early stopping: monitor "val_loss" mode "min" patience "5", batch size: "2048", epochs: "25"	

Figure 17: hyper-parameters for non resampled EXCLUDED dataset models. Also shows the number of k-folds for cross-validation

Dataset: NON Resampled TOTAL_RATIO

models		hyper-parameters	no. CV-folds
XGBoost Classifier	F1	{'colsample_bytree': 1, 'eval_metric': 'logloss', 'learning_rate': 0.2, 'max_depth': 10, 'min_child_weight': 5, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'scale_pos_weight': 43, 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	Rec	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 3, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'scale_pos_weight': 80, 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	Pre	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 5, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'scale_pos_weight': 1, 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	ROC	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 1, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'scale_pos_weight': 43, 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
XGBoost Random Forest	F1	{'colsample_bynode': 0.75, 'eval_metric': 'logloss', 'max_bin': 256, 'max_depth': 10, 'n_estimators': 400, 'predictor': 'gpu_predictor', 'scale_pos_weight': 43, 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	Rec	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 128, 'max_depth': 7, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'scale_pos_weight': 80, 'subsample': 0.9, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	Pre	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 128, 'max_depth': 10, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'scale_pos_weight': 1, 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	ROC	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 128, 'max_depth': 9, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'scale_pos_weight': 43, 'subsample': 0.9, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
Neural Network		Learning rate : decaying on rate (1e-3 * 10 ** (epoch / 30)), Early stopping: monitor "val_loss" mode "min" patience "5", batch size: "2048", epochs: "25"	

Figure 18: hyper-parameters for non resampled TOTAL_RATIO dataset models. Also shows the number of k-folds for cross-validation

Dataset: NON Resampled RATIO

models		hyper-parameters	no. CV-folds
XGBoost Classifier	F1	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.2, 'max_depth': 10, 'min_child_weight': 5, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'scale_pos_weight': 43, 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	Rec	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 5, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'scale_pos_weight': 80, 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	Pre	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 3, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'scale_pos_weight': 1, 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	ROC	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 5, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'scale_pos_weight': 43, 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
XGBoost Random Forest	F1	{'colsample_bynode': 0.75, 'eval_metric': 'logloss', 'max_bin': 128, 'max_depth': 10, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'scale_pos_weight': 43, 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	Rec	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 256, 'max_depth': 7, 'n_estimators': 400, 'predictor': 'gpu_predictor', 'scale_pos_weight': 80, 'subsample': 0.9, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	Pre	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 256, 'max_depth': 10, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'scale_pos_weight': 1, 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	ROC	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 128, 'max_depth': 9, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'scale_pos_weight': 43, 'subsample': 0.9, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
Neural Network		Learning rate : decaying on rate (1e-3 * 10 ** (epoch / 30)), Early stopping: monitor "val_loss" mode "min" patience "5", batch size: "2048", epochs: "25"	

Figure 19: hyper-parameters for non resampled RATIO dataset models. Also shows the number of k-folds for cross-validation

Dataset: NON Resampled KOMMUNE_DUMMIES

models		hyper-parameters	no. CV-folds
XGBoost Classifier	F1	{'colsample_bytree': 1, 'eval_metric': 'logloss', 'learning_rate': 0.3, 'max_depth': 10, 'min_child_weight': 5, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'scale_pos_weight': 43, 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	Rec	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 5, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'scale_pos_weight': 80, 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	Pre	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 3, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'scale_pos_weight': 1, 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	ROC	{'colsample_bytree': 0.75, 'eval_metric': 'logloss', 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 1, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'scale_pos_weight': 43, 'subsample': 0.75, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
XGBoost Random Forest	F1	{'colsample_bynode': 0.75, 'eval_metric': 'logloss', 'max_bin': 256, 'max_depth': 10, 'n_estimators': 200, 'predictor': 'gpu_predictor', 'scale_pos_weight': 43, 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	Rec	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 128, 'max_depth': 7, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'scale_pos_weight': 80, 'subsample': 0.9, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	Pre	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 256, 'max_depth': 7, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'scale_pos_weight': 1, 'subsample': 0.5, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
	ROC	{'colsample_bynode': 0.5, 'eval_metric': 'logloss', 'max_bin': 128, 'max_depth': 9, 'n_estimators': 300, 'predictor': 'gpu_predictor', 'scale_pos_weight': 43, 'subsample': 0.9, 'tree_method': 'gpu_hist', 'use_label_encoder': False}	3
Neural Network		Learning rate : decaying on rate (1e-3 * 10 ** (epoch / 30)), Early stopping: monitor "val_loss" mode "min" patience "5", batch size: "2048", epochs: "25"	

Figure 20: hyper-parameters for non resampled KOMMUNE_DUMMIES dataset models. Also shows the number of k-folds for cross-validation

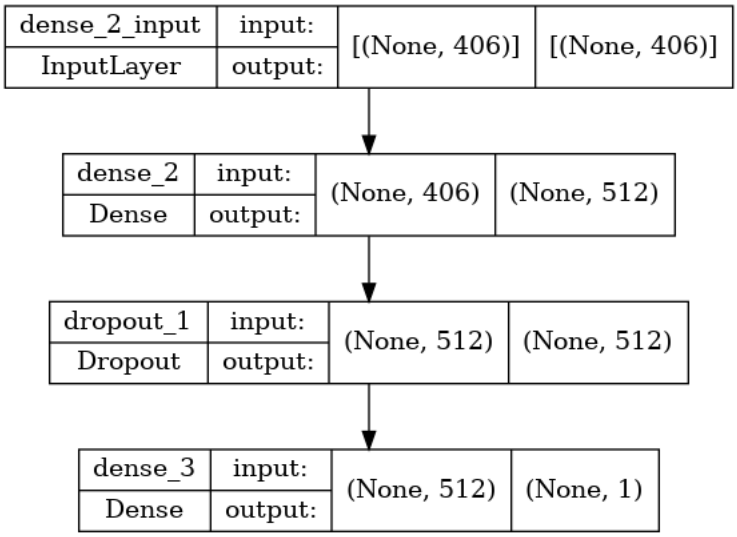


Figure 21: neural network structure for the EXCLUDED dataset.

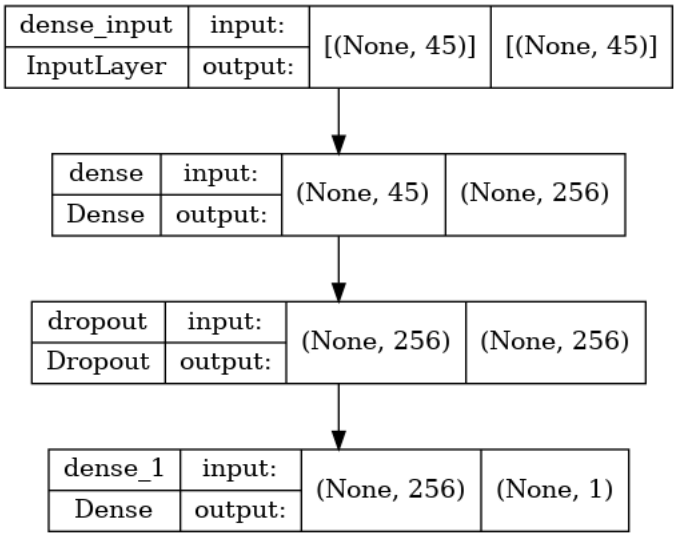


Figure 22: neural network structure for the TOTAL_RATIO dataset.

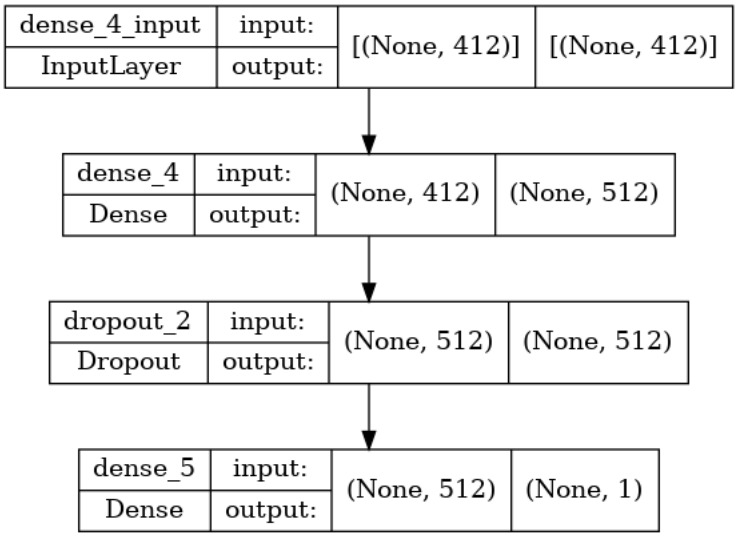


Figure 23: neural network structure for the RATIO dataset.

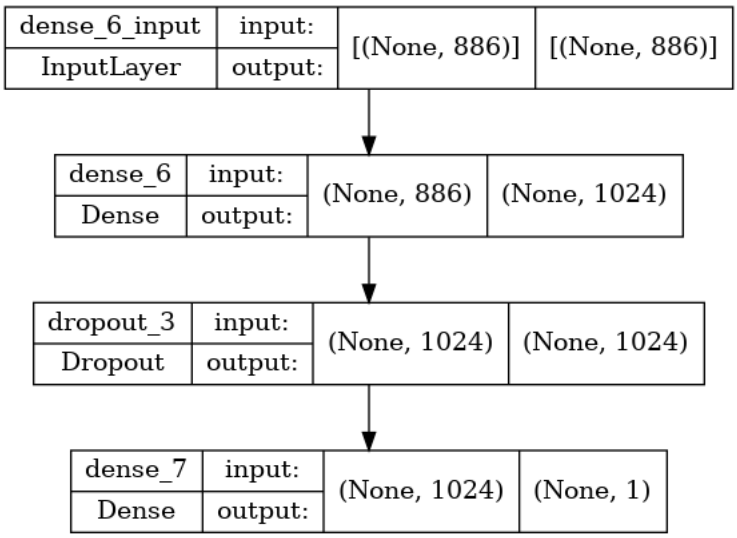


Figure 24: neural network structure for the KOMMUNE_DUMMIES dataset.

4 Results

To evaluate the performance of the models, we had a focus on recall, but were still interested in seeing the results of other performance metrics for comparison. This was done to get a more accurate and nuanced picture of the performance of the models on the different datasets.

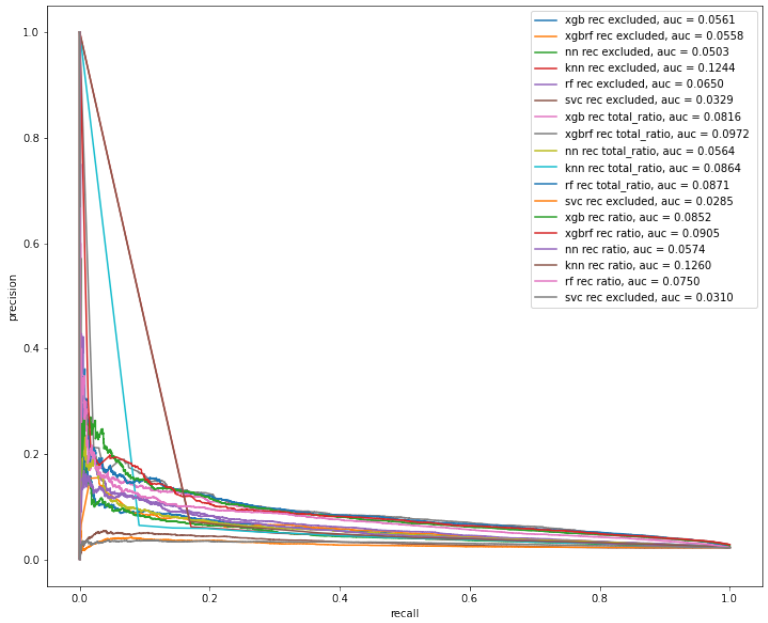


Figure 25: Precision-Recall curve with AUC score for models trained on resampled datasets

4.1 Results from all models

Tables 4 and 4.1 show evaluation metrics scores for the models that were trained on resampled and not resampled datasets respectively. For every ML model type except for neural network, we tested the four hyper-parameter combinations that got the best scores when parameter tuning. In the case of the models run on resampled datasets, when we tried to test the hyper-parameter combinations on *KOMMUNE_DUMMIES* dataset, we ran into issues caused by hardware limitations. Thus, we only have the best score in each evaluation metric but not the rest of the score for the parameter combinations. In the case of models trained on non-resampled datasets. All the models were trained using the Cost-sensitive technique, and Both the XGBoost classifier and XGBoost random forest models' test scores were cross-validated.

Figures 25 and 26 show the plots and score values of the precision-recall AUC. The first figure shows AUC results for the models trained on the reasm-

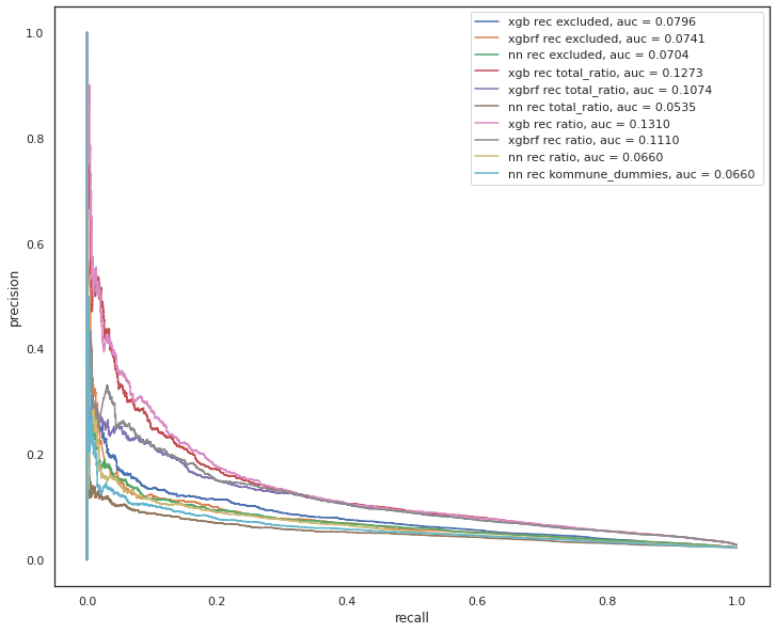


Figure 26: Precision-Recall curve with AUC score for models trained on non-resampled datasets

pled datasets. The second figure have the AUC results of the models trained on the non resampled datasets. The first figure does not have any score for the models trained on the *KOMMUNE_DUMMIES* dataset as we were faced with the same hardware limitations. The second figure only has neural network model trained on *KOMMUNE_DUMMIES*.

5 Discussion

The results in chapter 4, as we can see in Table 4, are missing some values. This is due to issues with hardware restriction and therefore not being able to finish training models because of out-of-memory errors. Due to a strict privacy policy, the team had to keep the dataset in Geomatikk's offices and, therefore, use the hardware provided. Because of that we were unable to get complete results for *kommune_dummies*

At the beginning of the project, we discussed strike damage in both official meetings and unofficial meetings with Geomatikk to understand what a strike damage is and what the causes of strike damage are. Geomatikk informed the team that strike damage has two main causes. About 50% of the cases are caused by human operator error. The other 50 % is caused by measurement error by the equipment used to find underground infrastructure or inaccurate mapping of infrastructure when they were buried in the ground.

models \ scores		EXCULDED					TOTAL_RATIO					RATIO					KOMMUNE_DUMMIES				
		Pre	Rec	F1	ROC	ACC	Pre	Rec	F1	ROC	ACC	Pre	Rec	F1	ROC	ACC	Pre	Rec	F1	ROC	ACC
KNN	F1	3.9	53	7	61	68.7	4	36	7.6	58.5	79.8	3.9	52	7	61	69.6			66.5		
	Rec	3	79.7	6	61.9	44.9	3.6	63	6.9	62	61	3	79	6	62	45.9		67.7			
	Pre	3.9	53	7	61	68.7	4	36	7.6	58.5	79.8	3.9	52	7	61	69.6	35				
	ROC	3	74	6.5	62	51	3.6	63	6.9	62	61	3	70.7	6.6	62	54					
Random Forest	F1	4.6	60.6	8.5	65.5	70	7.5	52.9	13	68.8	84	6	60.5	11.5	69.8	78.6			11		
	Rec	3	87.6	6	61.8	37	5.5	74.8	10	72	70	4	82.9	7.8	68.7	55		93			
	Pre	4.6	60	8.5	65	70	7.5	51.9	13	68.5	84	6	60.5	11.5	69.8	78.6	6				
	ROC	4	66.9	8	66	65.8	6	70.8	11	72	73.9	5	72.4	10	71	70				71.5	
SVC linear	F1	6.6	4.6	5	51.5	96	7.5	37	12.5	63	87.9	13	16	14.5	56.9	95.5			6.8		
	Rec	1.9	60.8	3.8	45	29.9	2.7	52.6	5	53.9	55	3	38.6	5.7	54.9	70.5		26			
	Pre	6.6	4.6	5	51.5	96	7.5	37	12.5	63	87.9	14	14	14	56	96	16.6				
	ROC	2.5	39	4.8	52	64	7.5	37	12.5	63	87.9	13	16	14.5	56.9	95.5				53.8	
XGBoost Classifier	F1	7.7	20.3	11	57	92.5	11	21.6	14.7	58.8	94	15.9	14.5	15	56	96			15.6		
	Rec	6.5	29.6	10.6	59.7	88.5	9	31	14	61.9	91	9	29.6	14	61	91.7		29			
	Pre	7.8	12.4	9.6	54	94.6	19.6	11	14	55	96.8	15.9	14.5	15	56	96	15.5				
	ROC	6.6	29.7	10.8	59.9	88.7	9	31	14	61.9	91	9	29.6	14	61	91.7				60.9	
XGBoost Random Forest	F1	3.8	68.8	7	64	60	6	70	11	72	74	5.6	70	10.5	71	72			10.3		
	Rec	3.6	75	7	64	53.8	5.6	76	10	72.9	70	5	76.8	9.8	72	67		76			
	Pre	3.8	68.8	7	64	60	6	70	11	72	74	5.6	70	10.5	71	72	5.7				
	ROC	3.8	68.8	7.3	64	60	5.8	74	10.8	73	71.6	5.5	74.5	10	72	69.9				72	
Neural Network		6.6	19	9.8	56	91.7	4.8	28	8.8	62.9	77	7	26	11	59	90	9	17.6	12	56.7	94

Table 4: Result Percentage scores table for The resampled datasets. There are five evaluation scores: **Pre**: precission, **Rec**:recall, **F1**, **ROC**:ROC AUC, and **ACC**: accuracy.

models	scores	EXCULDED					TOTAL_RATIO					RATIO					KOMMUNE_DUMMIES				
		Pre	Rec	F1	ROC	ACC	Pre	Rec	F1	ROC	ACC	Pre	Rec	F1	ROC	ACC	Pre	Rec	F1	ROC	ACC
XGBoost Classifier	F1	7	22.8	10	57	68	13	18	15	78	95	10.8	27.7	15.6	78.6	93	9	36.9	14.7	78.7	90
	Rec	3.7	82	7	59.7	74.8	5	82.9	9.7	82	64.5	5	85	9.4	82.5	62.5	4.8	86	9	82	60.9
	Pre	57	0.1	0.2	54	75.7	65	1	2.7	82.9	97.7	59.7	1	2.6	83	97.7	64	1	2.5	82.9	97.7
	ROC	5	64.8	9.6	59.9	75	6.6	71.4	12	82.4	76	6.5	73	11.9	82.7	75	6	73.5	11.6	82.5	74
XGBoost Random Forest	F1	5	60.8	9.5	74	73.5	6.6	70.6	12	82	76	6	69.9	12	82	76.6	6.5	70.5	11.9	82	75.9
	Rec	3	89.9	6	73.5	37	4	91	8	82	52	4	91	8	82	52.6	4	91	8	81.9	52
	Pre	5	61	9.6	74	73	71.8	1	1.9	82.5	97.7	69.5	1	2	82	97.7	82	0.7	1	81.8	97.7
	ROC	5	60.8	9.5	74	73.5	6	76	11	82	72	6	75.4	11	82	72.5	5.9	75.9	11	82	71.8
Neural Network		5	61	9	67	72.8	4.1	62.5	7.7	64	65.4	7.7	29	12	60.5	90	4.8	56.7	8.8	65	73

Table 5: Result Percentage scores table for *not* resampled datasets. The models were trained using the cost-sensitive learning technique, and then tested on each of the datasets. There are five evaluation scores: **Pre**: precission, **Rec**:recall, **F1**, **ROC**:ROC AUC, **ACC**: accuracy.

We observed values in the precision-recall curve25 for the kNN models where it looks like calculations are missing compared to the shape of the other models. We are not 100% sure of the cause, but think it has to do with the probability estimates of kNN returning probabilites with intervals of *1/number of neighbours*. Since the precision-recall curve method in Sklearn returns values based on the number of unique probabilities from the dataset and the number probabilities returned in kNN is set to be within a certain range, the odd shape of the graph made sense. Comparing the other metrics retrieved from the kNN model, the high PR-AUC does not make sense and we chose to disregard the results given by the precision-recall curve for the kNN model.

Looking at the results in Table 4 over the models trained on resampled data the best preforming model on recall overall is random forest, followed by kNN and XGBoost random forest not far behind. From Table 4.1 where the models were trained on the dataset not resampled, but by using cost-sensitive training, XGBoost random forest model preformed the best looking at recall. Reflecting on our result from the angle of how to handle data imbalance. We had two techniques we tried, resampling the data and cost-sensitive learning. If we take the evaluation metric of Recall. we find the best performance is with resampling. The Random forest model trained on *KOMMUNE_DUmmies* got a recall score of 93%, but since we could not run the test on the model with the hyper-paramters after parameter tuning, and the second best recall score is not so far off 91%, we choose to disregard the 93% score. The 91% score came from the XGBoost random forest model trained with cost-sensitive learning on non resampled data. the second best will be from the same model trained on non resampled data as well. Other than Recall we can look on F1 score or precision-recall AUC. The highest value for both of the metrics is less than 16%. Still, the highest F1 score came from XGBoost model trained using cos-sensitive learning on non resampled dataset.

In regards to the issue of non binary categorical features in datasets. We planed to test two encoding technique. DVE and MTE. The plan was originally to make 5 datasets, where one of them will only have DVE used on the non binary categorical features. This Was proven to not possible because of the hardware limitaions. The resulting dataset after encoding was so big that the operating system could not hold it in memory to finish the preprocessing. We then decided to make four datasets instead with varying variation of DVE to TME applied.

Checking the recall metric score, and disregarding the 93% in resampled *KOMMUNE_DUMMIES*. We find the second best score is 91% a tie between three datasets *KOMMUNE_DUMMIES*, *RATIO* and *TOTAL_RATIO*. They have (2 DVE, 2 MTE) , (1 DVE, 3 MTE), and (0 DVE, 4 MTE) respectively. Two have a combination and one only have MTE. This points to that the odds of getting

a better results when using a combination of both technique are higher.

Considering the question of which is the best ML model for our objective. If we check the recall values we find the even if we disregard the 93% score of the random forest model, we still find that all of the best recall results are from tree ML models. Random Forest getting 2, and XGBoost random forest getting 5 out of the 7 best recall scores for the datasets.

On the other hand if we check the precision-recall AUC scores in figures 26 and 25, and we disregard we disregard kNN model scores. We still find that tree ML models are outperforming the rest. We ran the test on 6 of the datasets, as hardware limitation prevented the test from running on both *KOMMUNE_DUMMIES* resampled and non resampled datasets. Out of the 6 datasets' the best AUC values were, random forest got one of the best scores, XGBoost got two and XGBoost Random forest got three. This makes it clear that tree ML models and specially decision boosted trees show promising results for our objective.

We suspected that the neural network model was overfitting. The model showed good recall scores on and off during training, but these result did not translate to good recall scores on the test set. To combat that we implemented measures such as dropout layers, early stopping and we decreased the complexity of the model by decreasing the number of nodes and number of layers. We got somewhat better result after this, but still not as good as some of the other models.

The results show that it is difficult to achieve both high recall and high precision. Looking at Figure 25 and Figure 26 we can see that all values for AUC are relatively low, with 0.131 being the highest for XGBoost classifier with *RATIO* dataset on non-resampled data. Still, it is not preforming drastically better than any of the other models. We can also see from Table 4 and Table 4.1 that where recall is high precision is low and vice versa.

6 Conclusion and Future work

6.1 Conclusion

The major focus of this report was to explore if machine learning could be used to predict if a strike damage will occur. In order to determine this the following research question was formulated:

How can machine learning be used for risk assessment of excavation projects?

From what we found in the results and discussed in Chapter 5, there are no

model and dataset combination from the experiments executed that clearly out-preforms another. Still, there are some promising result, the random forest models preformed well on both the resampled and non-resampled dataset. The boosted trees have a good performance when trained on non resampled datasets. Still, even the best performer random forest xgboost, has 91% recall and only 4% precision. This means that for every one true positive the model is identifying, 24 false positives are misidentified. This will lead to wasting the resources that are meant to mitigate strike damages on the false positives. When it comes to the performance of the different datasets, there is no one preforming significantly better than the other. This can possibly have something to do with the data not fully representing the real world, and that the data is not reflecting all the variables leading to a strike damage happening.

6.2 Future work

As there is no previous research on the topic of this paper, there is still much to explore. First, including different data is an option that should be considered. Other data that could be included are weather conditions or temperature, on or around the day of strike damage occurring or the day of detection. In a practical application of the machine learning model in the future, weather forecasts could be used in making risk assessments, same as weather observations would be used in the training of the model. Topographical data of the terrain or data on the type of ground under the surface of were the project is happening is also an option that could be experimented on. This was discussed with Geomatikk at the beginning of the project, as their data analyses had given some indications that weather might be a factor into causing strike damages. Each inquiry in the dataset also contains information on GPS coordinates of the area, which makes it possible to implement data based on geography. Due to the scope of implementing external data such as this, it was decided not to be implemented.

Second, we have only explored some supervised learning models. Other supervised machine learning models, e.g logistics regressing could be considered. Implementing something like term frequency-inverse document frequency (tf-idf) on organization id is something that was discussed as a possible solution, instead of one-hot-encoding, to deal with the extent of unique values in that column. Semi-supervised or self-supervised learning could also be applied to imbalanced data[26] and might yield better results, as supervised learning has so far not shown great results.

Lastly, we did not apply any feature selection or feature extraction techniques other than those already implemented in any of the models used. This is something to consider implementing.

There are also a few things we were not able to implement or try out due to hardware, software, or time limitations. These are thing that could be tried out in future work. One thing we were unable to do was to run the models on a dataset with *kommune_id* one-hot-encoded as we did not have large enough memory on the computer provided. Also, we used some feature engineering to transform all categorical features, and ran all models on those datasets. Some of the models we used can handle categorical data, and it is an option to try out.

Another interesting topic that can be explored is whether feature selection should be implemented before or after re-sampling. We only had the time to implement re-sampling, but it would be interesting to see if the order of feature selection and re-sampling would have an affect on the result as it seems to vary from case to case[27].

We only implemented a very simple neural network model, and did not do a lot of optimization. This could be to further explore trying out different optimizers and fine-tuning the learning rate and layers of the model.

7 Broader impact

The use of machine learning to prevent strike damage is a new field of research and is still far from being used in the real world. This report provides only a brief overview of some of the possibilities. Nevertheless, we know that preventing strike damages will have a huge impact on society on multiple areas.

From a report[28] published by the Norwegian Ministry of Local Government and Modernization, research has been done into the socio-economic consequences of strike damages. There they estimated the direct repair cost on the electrical grid to be 69.9 millions NOK yearly[28, p. 40]. In addition indirect cost (e.g, loss of income and cost related to postponement of project), and KILE[29] (Cost of Energy Unserved), also impact the total economical costs related to damage on the electrical grid. For damage to the electrical communication grid, the cost is estimated to be even higher. Dalen et al. did an analysis of 12 real-world cases relating to damage on the electrical communication grid and estimated a total cost of 348 million NOK yearly. Only looking at the costs of these two examples, we can see that being able to minimize strike damage by a few percent would save the society for a substantial financial amount yearly.

There are also non-financial consequences related to strike damages. Social costs[28, p. 28], such as an increase in traffic, and increased pollution or noise, have an impact on peoples health and the environment. Another example which could pose a great risk to peoples health is if the mobile net-

work is to be damaged and unavailable. We have examples[30][31] where the EMK has been unavailable to the public for periods of time. A worst case scenario would be if the public was unable to reach emergency services due to the loss of mobile service and a loss of life happened because of the inability to access medical service or the fire department. Therefore, decreasing strike damages and as a result possibly decreasing possible downtime of the EMK could help save lives.

The dataset we worked with contained ids that can be mapped to individual people or companies. When using machine learning with data containing personal information, some ethical questions arise. We have to be aware of potential bias in machine learning. If not careful machine learning models can put false accusation on an individual or company. In the case with risk assessment of strike damages, it could potentially falsely accuse a company of being high risk, while the actual cause of the higher risk might be due to a factor not contributed by the company or detector.

8 References

[1] J. McCarthy. "What is artificial intelligence?" (), [Online]. Available: http://jmc.stanford.edu/artificial-intelligence/what-is-ai/index.html. (accessed: 05.05.2022).

[2] C. Wang, A. V. Savkin, R. Clout, and H. T. Nguyen, "An intelligent robotic hospital bed for safe transportation of critical neurosurgery patients along crowded hospital corridors," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 23, no. 5, pp. 744–754, 2015. DOI: 10.1109/TNSRE.2014.2347377.

[3] G. Lugano, "Virtual assistants and self-driving cars," in *2017 15th International Conference on ITS Telecommunications (ITST)*, 2017, pp. 1–5. DOI: 10.1109/ITST.2017.7972192.

[4] T. Mitchell, "The discipline of machine learning," School of Computer Science, Carnegie Mellon University, Tech. Rep. CMU ML-06 108, Jan. 2006.

[5] I. H. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *SN Computer Science*, vol. 2, no. 3, p. 160, Mar. 2021, ISSN: 2661-8907. DOI: 10.1007/s42979-021-00592-x. [Online]. Available: https://doi.org/10.1007/s42979-021-00592-x.

[6] J. E. van Engelen and H. H. Hoos. "A survey on semi-supervised learning." (), [Online]. Available: https://link.springer.com/article/10.1007/s10994-019-05855-6. (accessed: 15.05.2022).

[7] T. M. Oshiro, P. S. Perez, and J. A. Baranauskas. "How many trees in a random forest." (), [Online]. Available: https://www.researchgate.net/profile/Jose-Baranauskas/publication/230766603_How_Many_Trees_in_a_Random_Forest/links/0912f5040fb35357a1000000/How-Many-Trees-in-a-Random-Forest.pdf. (accessed: 19.05.2022).

[8] M. Kuhn, *Applied Predictive Modeling*, eng, 1st ed. 2013. New York, NY: Springer New York : Imprint: Springer, 2013, ISBN: 1-4614-6849-3.

[9] W. S. Noble. "What is a support vector machine?" (), [Online]. Available: https://www.ifi.uzh.ch/dam/jcr:00000000-7f84-9c3b-ffff-ffffc550ec57/what_is_a_support_vector_machine.pdf. (accessed: 15.05.2022).

[10] A. Pradhan. "Support vector mahine - a survey." (), [Online]. Available: https://d1wqtxts1xzle7.cloudfront.net/54195341/SUPPORT_VECTOR_MACHINE-A_Survey20170821-6881-1lwnw89-libre.pdf?1503314350=&response-content-disposition=inline%3B+filename%3DSUPPORT_VECTOR_MACHINE_A_Survey.pdf&Expires=1652636338&Signature=e-2sFR0WmHmhPXgNozsGTS-omrtfM948O1WH-sKrb6nzgogVz5JFjLMl9VZgiFV1a~RqsiJRiANW5LuS-~1DgGz6lPqTMZhEmGyY87tWfjs5dKSHdK3fwKkXS1TZ9xdzVmxemGhizI7FSxYX50anpZaFyQG5aAfwe9FcJKbqELykSpELHGTX0nzZAedFodbK7Nd6a_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA. (accessed: 15.05.2022).

[11] G. E. Batista and D. F. Silva. "How k-nearest neighbor parameters affect its performance." (), [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.696.2787&rep=rep1&type=pdf. (accessed: 15.05.2022).

[12] R. Mitchell and E. Frank. "Accelerating the xgboost algorithm using gpu computing." (), [Online]. Available: https://peerj.com/articles/cs-127/. (accessed: 15.05.2022).

[13] G. D. Franco and M. Santurro. "Machine learning, arificial neural networks and social research." (), [Online]. Available: https://link.springer.com/article/10.1007/s11135-020-01037-y. (accessed: 15.05.2022).

[14] C. X. Ling and V. S. Sheng. "Cost-sensitive learning and the class imbalance problem." (), [Online]. Available: https://www.researchgate.net/file.PostFileLoader.html?id=56d1ad946307d916ce8b4569&assetKey=AS%3A333743404404737%401456582036953. (accessed: 15.05.2022).

[15] V. Verdhan, *Supervised learning with Python : concepts and practical implementation using Python*, eng, 1st ed. 2020. Place of publication not identified: Apress, 2020, ISBN: 1-4842-6156-9.

[16] C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2017, ISBN: 978-1-4899-7687-1. DOI: 10.1007/978-1-4899-7687-1. [Online]. Available: https://doi.org/10.1007/978-1-4899-7687-1.

[17] Z.-H. Zhou, *Machine Learning*, eng. Singapore: Springer Singapore Pte. Limited, 2021, ISBN: 981-15-1967-6.

[18] H. R. Sofaer, J. A. Hoeting, and C. S. Jarnevich. "The area under the precision-recall curve as a performance metric for rare binary events." (), [Online]. Available: https://www.researchgate.net/profile/Helen-Sofaer/publication/329722807_The_area_under_the_precision-recall_curve_as_a_performance_metric_for_rare_binary_events/links/5cb4bd574585156cd79ad57e/The-area-under-the-precision-recall-curve-as-a-performance-metric-for-rare-binary-events.pdf. (accessed: 19.05.2022).

[19] D. Berrar. "Cross-validation." (), [Online]. Available: https://www.researchgate.net/profile/Daniel-Berrar/publication/324701535_Cross-Validation/links/5cb4209c92851c8d22ec4349/Cross-Validation.pdf. (accessed: 19.05.2022).

[20] R. Ghorbani and R. Ghousi. "Comparing different resampling methods in predicting students' performance using machine learning techniques." (), [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9062549. (accessed: 13.05.2022).

[21] A. D. Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi. "Calibrating probability with undersampling for unbalanced classification." (), [Online]. Available: https://www3.nd.edu/~dial/publications/dalpozzolo2015calibrating.pdf. (accessed: 13.05.2022).

[22] imbalanced-learn developers. "Undersampling." (), [Online]. Available: https://imbalanced-learn.org/stable/under_sampling.html#edited-nearest-neighbors. (accessed: 19.05.2022).

[23] S. Ozdemir, *Feature engineering made easy*, eng, 2018.

[24] D. Micci-Barreca, "A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems," *SIGKDD Explor. Newsl.*, vol. 3, no. 1, pp. 27–32, Jul. 2001, ISSN: 1931-0145. DOI: 10.1145/507533.507538. [Online]. Available: https://doi.org/10.1145/507533.507538.

[25] A. Wrålsen and K. E. Berntsen, "Vitenskapelighet i ba-oppgaven," VT1, Jan. 2021.

[26] Y. Yang and Z. Xu, "Rethinking the value of labels for improving class-imbalanced learning," eng, 2020.

[27] C. Zhang, J. Bi, and P. Soda, "Feature selection and resampling in class imbalance learning: Which comes first? an empirical study in the biological domain," in *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2017, pp. 933–938. DOI: 10.1109/BIBM.2017.8217782.

[28] D. M. Dalen, A. Einarsdottir, J. Furuholmen, H. Ringdal, V. Strøm, and H. Vennemo. Aug. 2020. [Online]. Available: https://www.regjeringen.no/contentassets/6338268c2580455b980a484db2c1f316/va2020_samfunnsokonomiske_kostnader_graveskader_ledninger.pdf.

[29] NVE, no. [Online]. Available: https://www.nve.no/reguleringsmyndigheten/regulering/nettvirksomhet/%c3%b8konomisk-regulering-av-nettselskap/om-den-okonomiske-reguleringen/kile-kvalitetsjusterte-inntektsrammer-ved-ikke-levert-energi/, accessed: 13.05.2022.

[30] T. Reite, *Store telefonproblemer*, nb-NO, May 2011. [Online]. Available: https://www.nrk.no/mr/store-telefonproblemer-1.7644712, accessed: 16.05.2022.

[31] M. Torstveit, *Siljan utan dekning*, nn-NO, Jan. 2020. [Online]. Available: https://www.nrk.no/vestfoldogtelemark/siljan-utan-dekning-1.14848456, accessed: 16.05.2022.