

Markus J. Aase

Predicting persistent weak layers in maritime regions in Norway using meteorological parameters

Master's thesis in Natural Science with Teacher Education

Supervisor: Jo Eidsvik

December 2022



Norwegian University of
Science and Technology

Markus J. Aase

Predicting persistent weak layers in maritime regions in Norway using meteorological parameters

Master's thesis in Natural Science with Teacher Education
Supervisor: Jo Eidsvik
December 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences



Norwegian University of
Science and Technology

Acknowledgments

This thesis concludes my specialization in statistics in the five-year Master's degree programme in Natural Science With Teacher Education at the Norwegian University of Science and Technology (NTNU).

Firstly, I want to express gratitude to my supervisor, Jo Eidsvik, for letting me suggest a thesis in statistical learning about a field of interest of mine, and helping me along the way to reach this goal through support, assistance, and valuable discussions. This is a field that interest me in academic, professional, and recreational ways, and through this semester my interest for both statistical learning and avalanches/snow-science have been enriched. Secondly, I have to thank the Department of Mathematical Sciences, and the Department of Teacher Education for providing a broad and exciting time as a student in Trondheim.

I have received help from a variety of people in this project. Stefanie Muff and Daesoo Lee (Department of Mathematical Sciences, NTNU) gave insight and good discussions during the semester of writing this thesis. Additionally, I would like to thank everyone at Varsom who helped me with gathering the data, and answering my questions that arose throughout this project.

Finally, I want to show appreciation for the most important people around me for support throughout this semester. You all know who you are. Thank you!

Trondheim, 14th December, 2022

Markus J. Aase

Abstract

Avalanche forecasts use different *avalanche problems* to distinguish what might cause an avalanche on a given day. One of the five avalanche problems is called *persistent weak layer* (PWL), a horizontal layer within the snowpack *weaker* than its surrounding snow. The grains of this layer can be *faceted grains*, *depth hoar*, or *surface hoar*. This layer is usually created by a high *temperature gradient* within the snowpack. This thesis aims to predict whether a given day will have a PWL or not by using computer-gathered meteorological parameters for two maritime regions, Romsdal and Sunnmøre. Features such as air temperature and snow depth, at different elevation intervals, are used as an indirect approach to this temperature gradient.

This thesis considers three models. These differ in the number of features, and whether they take time into consideration or not. All models use statistical and machine learning methods such as classification trees, bagging, random forests, and boosting. To extend the analysis of the time-dependent model, we implemented a neural network that uses the same dataset. The results show that including time dependency was a good idea. *Boosted classification trees* trained on the last four seasons and tested on the first season stood out as the best method. It had an accuracy of 0.8927, specificity of 0.9462, and sensitivity of 0.8333. Despite these promising results, the methods were remarkably sensitive to what seasons are used as training data, hence the methods are non-robust. When this is said, one could use the methodology in this thesis when more data is collected, which will happen as the years pass. This thesis only had five winter seasons worth of data for both its training and testing. The neural network performed exceptionally well on the random training set (75% of the data chosen at random), where its test set was the remaining 25%. These high predictive measurements arise from what we have called *seasonal dependencies*, and this leads to overfitting. Although there is a connection between the temperature gradient *within* the snowpack, the snow depth and the air temperature, the findings of this thesis imply that this approach oversimplifies the problem at hand. It made reasonable sense to use these features to predict faceted grains and depth hoar. However, the dataset had its limitations when trying to predict surface hoar, the last grain form of PWLs. From our results, we conclude that this approach is too unreliable to replace the existing workflow of the avalanche forecast. Nevertheless, we have presented methods that could potentially aid the forecast in predicting the avalanche problem PWL in Sunnmøre and Romsdal.

Sammendrag

Skredvarslingen bruker ulike *skredproblem* for å skille mellom hva som kan forårsake et snøskred på en gitt dag. Et av de fem skredproblemene kalles *vedvarende svake lag* (PWL), et horisontalt lag inne i snødekket som er *svakere* enn dets omkringliggende snø. Disse snøkornene kan være *kantkorn*, *begerkrystaller*, eller *overflaterim*. Dette laget dannes vanligvis av en høy *temperaturgradient* inne i snødekket. Denne oppgaven prøver å predikere om en gitt dag vil ha PWL eller ikke ved å bruke datasamlede meteorologiske parametere for to maritime regioner, Romsdal og Sunnmøre. Variabler som lufttemperatur og snødybde, på ulike høydeintervall, brukes som en indirekte tilnærming til denne temperaturgradienten.

Denne oppgaven omfatter tre modeller. Disse skiller seg fra hverandre ved antall variabler, og om de tar tid med i betrakting eller ikke. Alle modellene bruker maskinlæringsmetoder som beslutningstrær, bagging, random forests, og boosting. For å utdype analysen av den tidsavhengige modellen, implementerte vi et nevratlt nettverk som bruker det samme datasettet. Resultatene viser at å inkludere tidsavhengighet var en god idé. *Boosted klassifiseringstrær* trent på de siste fire sesongene og testet på den første sesongen markerte seg som den beste metoden. Den hadde en nøyaktighet på 0.8927, spesifisitet på 0.9462, og sensitivitet på 0.8333. Til tross for disse lovende resultatene, er metodene veldig sensitive for hvilke sesonger som er brukt som treningsdata, derfor er metodene ikke-robuste. Når dette er sagt, så kan man bruke metodene fra denne oppgaven når mer data blir samlet, noe som vil skje når årene passerer. Denne oppgaven hadde bare fem vintersesonger med data til både trening og testing. Det nevrale nettverket gjorde det eksepsjonelt bra med det tilfeldige treningssettet (75% av dataen valgt tilfeldig), hvor testsettet var de resterende 25%. Disse høye prediksjonstallene kommer fra det vi har valgt å kalle *sesongavhengigheter*, og fører til overtilpasning. Selv om det er en sammenheng mellom temperaturgradienten i snødekket, snødybden og lufttemperaturen, så viser resultatene i denne oppgaven at denne tilnærmingen oversimplifiserer problemet. Det ga rimelig mening å bruke disse variablene til å predikere kantkorn og begerkrystaller. Imidlertid hadde datasettet sine begrensninger i å predikere overflaterim, den siste kornformen av PWLs. Fra våre resultater konkluderer vi med at denne tilnærmingen er for upålitelig til å erstatte den eksisterende arbeidsflyten til skredvarselet. Til tross for dette har vi presentert metoder som potensielt kan bistå varslingen i prediksjon av skredproblemet PWL i Sunnmøre og Romsdal.

Contents

1	Introduction	1
1.1	Goals and research questions	2
1.2	Structure of the thesis	2
2	Related work	4
3	Background on snow and avalanche theory	6
3.1	Avalanches	6
3.1.1	Slab avalanches	8
3.2	Norwegian avalanche forecasting	9
3.2.1	How the forecast is made	9
3.2.2	Avalanche problems	12
3.3	Snow science	13
3.3.1	Kinetic transformation - Persistent Weak Layers	14
4	The dataset and preliminary analysis	17
4.1	Data	17
4.1.1	Notation of data	17
4.1.2	Processing of data	19
4.2	Preliminary analysis of data	20
5	Statistical learning	26
5.1	Supervised learning	27
5.2	Splitting of dataset	28
5.3	Decision trees	28
5.3.1	Regression trees	30
5.3.2	Classification trees	31
5.4	Bagging/bootstrap aggregation	32
5.5	Random Forest	33

5.6	Boosting	35
5.7	Deep learning and Neural networks	37
5.8	Measurement of predictive performance	39
6	Results	41
6.1	Stationary temperature model	44
6.2	Stationary temperature model + snow depth as an additional feature	44
6.3	Time-dependent model using temperature and snow depth	45
6.4	Multilayer perceptron	45
6.5	Summary of key machine learning results	54
7	Discussion	55
7.1	Different training sets and overfitting	55
7.2	Comments on the three models	56
7.2.1	Stationary temperature model	56
7.2.2	Stationary temperature model + snow depth as an additional feature	57
7.2.3	Time-dependent model using temperature and snow depth	57
7.2.4	Multilayer perceptron	60
7.3	Avalanche theory discussion and how this can aid the forecast	60
8	Closing remarks and future research	62
8.1	Closing remarks	62
8.2	Future research	63
Bibliography		65
A	Code examples	67
A.1	Retrieving data from Varsom	67
A.2	R code	68
A.3	Python code	69

Chapter 1

Introduction

One-third of all precipitation in Norway is snow (NVE, 2021). All this snow accumulates into a snowpack, a complex mixture of snow grains that go through constant metamorphism, changing the shape and properties of the grains. This can lead to properties that increase or decrease the likelihood of an avalanche. Snow avalanches are a natural hazard in Norway, and can cause the destruction of infrastructure and be dangerous for recreationists. This means that an avalanche forecast is essential for mountainous regions. Norwegian Water Resources and Energy Directorate (NVE) is responsible for the avalanche forecast. The forecast is called *Varsom* and follows an international standard of avalanche forecasts. However, if the forecast is going to help lowering the risk and prevent accidents, the forecast has to be precise, easy to interpret, and consistent (NVE, n.d.-b).

The goal of the forecast is two-sided. On one side, the forecast is made to be a protection of infrastructure and road safety, while the other aspect is for recreationists using the winter-outdoors for skiing, climbing, and other winter activities. The forecast is released daily between the 1st of December to the 31st of May during the winter season for different regions in Norway (NVE, 2016). The forecast is based on weather data and observations from professional snow observers in the field. It is also possible for amateurs to report to Varsom what they see in the field from an application called *Varsom Regobs*. Forecasters gather this data together with the additional weather forecast and then conclude to give an avalanche forecast for a specific region for a given day (NVE, n.d.-b). Unfortunately, it is impossible to find an answer to whether an avalanche will occur or not. Instead, the forecast is a guideline for the public to use, and it gives information about the snowpack, terrain, behaviour-advice, weather, and many more. However, in mountainous regions, these aspects can change dramatically over small geographical areas (NVE, 2016).

When Varsom publishes a daily forecast, it includes an avalanche danger level between 1 and 5, where 5 is extremely rare, but the most dangerous. In addition to this danger level, the forecast publishes an *avalanche problem*, which is supposed to give the reader of the forecast a guideline of what may cause an avalanche in the given region. These avalanche problems are divided into five different categories, and each has specific characteristics. *Persistent weak layer (PWL)* is an avalanche problem usually created by a high temperature gradient within a snowpack. This layer can be very fragile, hence dangerous if a slab forms on top of it. If this happens, it can cause massive and deadly slab avalanches.

Creating an accurate forecast can be challenging when regions lack professional observers to do observations in the field. This thesis will examine if previously gathered data with statistical and machine learning methods can be helpful in this problem, by creating a model that predicts the existence (1) or non-existence (0) of the particular avalanche problem PWL. With difficult, real-world data, this thesis focuses on predicting this binary outcome in two *maritime* regions in Norway, *Sunnmøre* and *Romsdal*.

1.1 Goals and research questions

The goal of this thesis can be summarized as follows:

Goal: *Develop a robust statistical model to predict whether a maritime region will have persistent weak layer(s) or not on a given day.*

The avalanche forecast uses weather data and actual snow observations performed by professionals to publish an avalanche warning. To achieve our goal, we will use this weather data in combination with supervised learning methods to predict this binary outcome (no-PWL (0)/PWL (1)). This motivates the first research question:

Research question 1: *Can familiar statistical models, trained on features only consisting of weather data, yield reliable results in forecasting the existence of PWLs?*

The statistical models in this thesis will primarily be classification trees, bagging, random forests, and boosting. These tree-based methods are well-known and established because of their overall good performance. However, they differ a bit from neural networks in machine learning. This raises another research question:

Research question 2: *Will applying deep machine learning techniques increase the overall predictive performance?*

1.2 Structure of the thesis

Chapters 2, 3 and 4

Chapter 2 focuses on related work that used machine and statistical learning methods in predicting various parts of avalanche hazards. Often there can be a gap between the field of research and the statistician doing the work. Therefore much work was put into Chapter 3 (*Background on snow and avalanche theory*). This laid the foundation for understanding the problem at hand, why a particular model can be of interest, and what meteorological parameters are suitable for features. Chapter 4 includes information about the dataset provided by NVE. In addition, this chapter includes a preliminary analysis of different regions in Norway and why maritime regions became the primary interest of this thesis.

Chapters 5 and 6

Chapter 5 on *statistical learning* lays the framework for the methodology used in the thesis. In Chapter 6, the results from our four models are presented in different tables, and each model's best results and key findings are summarized.

Chapters 7 and 8

The results of Chapter 6 are discussed thoroughly in Chapter 7, where Chapter 3 on the formation of PWL and Chapter 5 on statistical and machine learning are used to explain the trends in the results. Chapter 8 focuses on the main findings of this thesis and closing remarks. We will also present thoughts on future research on this topic.

Appendix

Appendix A showcase the data gathering and main code used to find the predictive measurements in Chapter 6. This section is split into three parts. Firstly, we present the retrieval of the data (A.1), then the R code (A.2), and lastly, the Python code (A.3).

Chapter 2

Related work

The books *Skredfare* (translated: *Avalanche danger*) by Landrø (2002) and *Staying alive in avalanche terrain* by Temper (2018) emphasize the complexity of snow avalanches. They discussed snow transformation (that cause, *e.g.*, PWL), terrain, decision-making, the forecast, avalanche rescue, physics, stability tests, weather, and many more. Jamieson (1995) investigated two different snow stability tests and performed fieldwork in British Columbia and Alberta in 1992-93 and 1994-95. They focused their fieldwork on PWL(s), the layers that cause the most fatal slab avalanches in Canada. The mentioned books and thesis introduced some of the broader topics related to this thesis. We next focus on papers that use machine and statistical learning approaches to automate different aspects of the avalanche forecast.

Marienthal et al. (2015) used meteorological parameters to forecast deep slab avalanches on PWL, meaning that the weak layer is buried deep within the snowpack. This means that it can be challenging to trigger these avalanches, such that the *stress* surpasses the *strength* of the snowpack. When deep slab avalanches occur, they often propagate to large, destructive avalanches. They defined *deep slab avalanches* as slab avalanches that slid on a PWL deeper than 0.9 meters and were triggered after the 1st of February. They used classification trees and random forests to predict whether or not meteorological data can predict seasons and days with deep slab avalanches. For this research, they had 44 years of avalanche control and meteorological data from southwest Montana. Studies in this field had previously used features from the days before the event of a deep slab avalanche, but Marienthal et al. used features far back before the deep slab avalanche. Their work showed that seasons with deep slab avalanches on PWL(s) generally had less precipitation from November to January, meaning the early ski season. They found that both daily meteorological parameters and seasonal parameters can be used to aid the avalanche forecast when it comes to deep slab avalanches on PWL, in combination with avalanche experts doing observations in the field.

Widforss (2021) applied machine learning techniques to three seasons of data from 23 different forecasting regions in Norway. His goal was to automate the avalanche forecasting process, looking at danger levels, warning for spontaneous release, avalanche problems, and many more. He split regions into training, validation, and test set. Widforss tried different machine learning techniques, such as decision trees and multilayer perceptrons. These models were compared to what he called *naive models*, which were simple models using previous forecasts and most occurring values as data. These were called *NaiveYesterday* and *NaiveMode*. When only weather data was used, the machine learning models would beat the NaiveModels. However, when using more than weather data, meaning snow observations and previous forecasts, the amount of data was lacking and presented problems.

Dkengne Sielenou et al. (2021) used random forests with class balancing to predict avalanche activity in the French Alps. They combined their method with a variety of existing statistical learning methods: linear discriminant analysis, K-nearest neighbors, classification trees, random forest, and support vector machines. Their dataset consisted of over 50 years of daily avalanche observations in 23 areas in the French Alps. They chose to split avalanche activity into three parts:

-
- Days with no avalanche activity.
 - Days with moderate avalanche activity.
 - Days with high avalanche activity.

When they looked into variable importance they found that the chosen features in their random forest coincided with the variables that were important in the existing avalanche literature. For the different areas in the French alps, their random forest classifier performed the best out of the learning methods, in terms of the minimum overall classification error. They concluded that their methods are a potential aiding tool for the avalanche forecast in the French Alps.

The discussed papers either investigate if an avalanche will occur based on avalanche observations or meteorological parameters, or trying to automate the forecasting process. The scope of this thesis will solely focus on predicting the specific avalanche problem persistent weak layer (PWL) in two maritime regions, *i.e.*, if PWL exists on a given day or not.

Chapter 3

Background on snow and avalanche theory

3.1 Avalanches

When a large mass of snow slides rapidly down a mountainside, it is called an avalanche. Avalanches are usually split into two types, *slab avalanches*, and *loose-snow avalanches*. The latter is when snow starts moving at one point, and snow gathers and rolls downwards in a fan-shape from the starting point, see Figure 3.1. Slab avalanches occur when the avalanche breaks into several pieces of cohesive snow blocks, see Figure 3.2. The snow gets cohesive by wind packing the snow crystals tightly together, hence forming a slab. When a skier/snowboarder triggers slab avalanches, the victim is usually in the middle of the avalanche when it occurs. This is an important feature of human-released slab avalanches (Temper, 2018). Hence this type of avalanche is the most dangerous for freeskiers, and 99 percent of all avalanches that occur because of skiers are of this type (Landrø, 2002).



Figure 3.1: Illustration of a typical fan-shaped loose-snow avalanche (Landrø, 2002).

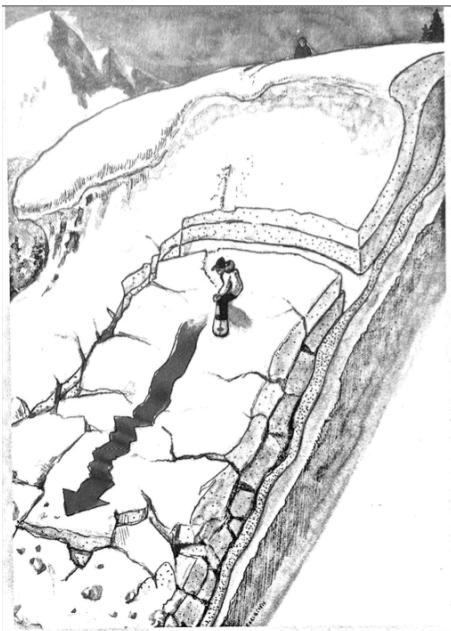


Figure 3.2: Illustration of a typical slab avalanche with the trigger (the snowboarder) already being on top of the sliding snow (Landrø, 2002).

Both avalanche types mentioned above can be dry or wet avalanches, meaning that the snow that ends up in the avalanches is either dry or wet (Temper, 2018). Wet loose-snow avalanches occur when there is a loss of cohesion between snow crystals from rain or melting snow. In contrast, wet slab avalanches occur because rain or melted snow increase the additional load on the existing snowpack, or water can make weak layers in the snow even weaker. This can cause a fracture such that a slab is released (EAWS, 2017).

There are other types of avalanches, called *Cornice-Fall avalanches*, *Icefall avalanches*, *Slush-flow avalanches* and *Roof avalanches* (Temper, 2018). These are difficult to forecast, and not in the main scope of the Norwegian avalanche forecast service, Varsom. While *Glide avalanches* are forecasted by Varsom, and is also a type of wet avalanche, see Figure 3.3.



Figure 3.3: Picture of glide avalanche that killed two people on 28th of April, 2001 (UAC, n.d.).

Glide avalanches are the phenomenon when the entire snowpack, all the way down to the ground, slides down a mountainside. This type of movement is more similar to when a glacier "creeps" downward, which differs from slab and loose-snow avalanches. Glide avalanches are hard to predict, and are usually not released by the additional load of a human. Glide cracks are not necessarily an immediate sign of danger of gliding snow, but it is a sign that the snowpack is creeping and can potentially release naturally. These avalanches can be very destructive, because of the magnitude

of snow released. In Norway, these avalanches usually happen during springtime, when the sun and rain cause water to percolate through the snowpack, weakening the strength of the snowpack and resulting in enormous avalanches. Despite this, glide avalanches are not the main cause of deaths, most of which are released naturally (Temper, 2018). As mentioned, slab avalanches are the most dangerous avalanche type for recreationists. Hence it is the main interest of the Norwegian avalanche forecast and therefore the primary avalanche type of concern in this thesis.

3.1.1 Slab avalanches

The snowpack is a complex mixture of different snow grains with different hardness, bonds, and forms. This difference in the snow is the key component for slab avalanches. A sliding slab is when a cohesive layer of snow slides upon another layer, caused by a fracture in an in-between, weaker layer. In avalanche terminology, we call these for *slab*, *weak layer*, and *bed surface*, see Figure 3.4.

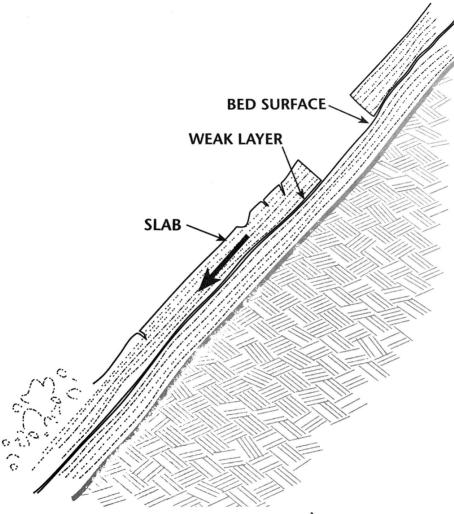


Figure 3.4: Illustration of the cross-section of a slab avalanche (Temper, 2018).

Dry slab avalanches occur in terrain when the angle of the slope is 30 degrees or steeper (Landrø, 2002). They occur when a fracture happens within the snowpack, which means that the *stress* surpasses the *strength* of the snow, see Figure 3.5. The most unstable and unpredictable situation is when $\text{stress} \approx \text{strength}$, due to a small additional load on top of the snowpack (*i.e.*, increased stress) will cause a fracture within the snowpack. This fracture can lead to a fracture, propagation, and a slab avalanche if a cohesive slab is on top of the weak layer (Temper, 2018).

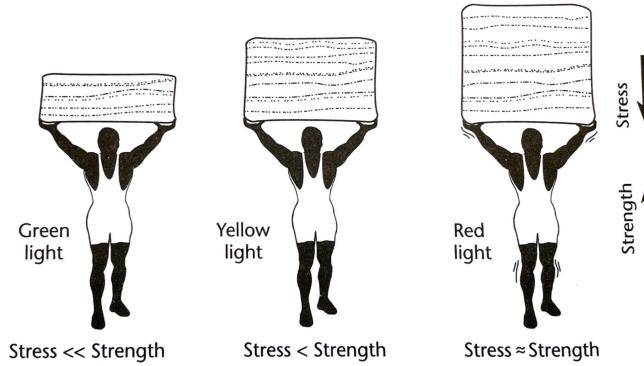


Figure 3.5: Explanatory illustration of stress vs. strength of a slab (Temper, 2018).

When discussing avalanches, it's important to note that avalanches are only dangerous when people or infrastructure are in the *starting zone*, *track*, or *runout zone*. Figure 3.6 introduces the terminology used for avalanche paths.

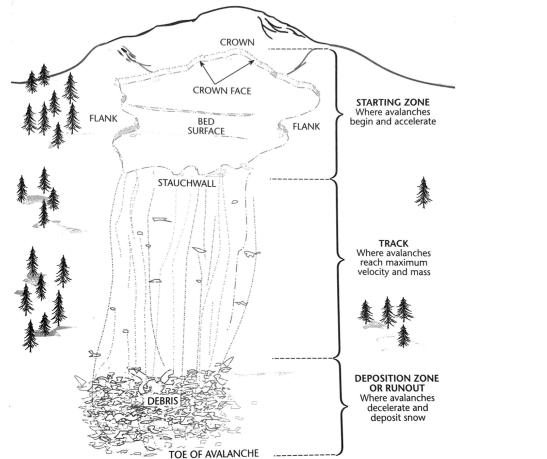


Figure 3.6: Illustration of slab avalanche path (Temper, 2018).

3.2 Norwegian avalanche forecasting

NVE is responsible for the social security that the avalanche forecast provides and is called Varsom. They cooperate with the Norwegian Public Roads Administration and The Norwegian Meteorological Institute (NVE, n.d.-b). Although the forecast cannot give any precise conclusions on whether there will be an avalanche or not, it provides its users with information about the snowpack, weather, avalanche danger, avalanche problems, and behavior advice. It is important to note that the forecast is less accurate further into the future due to insecurities in the weather forecast and the development of the snowpack (NVE, n.d.-d).

3.2.1 How the forecast is made

The general workflow Varsom uses to create the avalanche forecast is as follows:

1. Gather data on the snowpack and weather.
2. Evaluate the current situation.
3. Look at the weather forecast.
4. Evaluate probable development of the conditions.
5. End up with an avalanche forecast.

Avalanches are classified by different attributes, their size, what causes them, the avalanche danger, their distribution in the terrain, and their probability of releasing. The Canadian Avalanche Association (CAA) defined a chart for avalanche sizes in 2016, where avalanches are characterized on a scale between 1 and 5 on potential destruction, volume/mass, and the typical length of the avalanche path. The latter is characterized in slightly different ways when comparing CAA (CAA, 2016) to Varsom (NVE, n.d.-e), although the essence is the same. See Table 3.1 for a description of the different avalanche sizes.

Table 3.1: Avalanche size chart translated from Varsom.

Size	Description of potential destruction.	Volume	Typical length of avalanche
1 (Small)	Relatively harmless to people.	< 100 m ³	Avalanche stops in mountainside.
2 (Medium)	Could bury, injure, or kill a person.	< 1000 m ³	Avalanche stops in the end of mountainside.
3 (Large)	Could destroy and bury a car, damage trucks, destroy small houses and trees.	< 10,000 m ³	Avalanche continues up to 50 meters when it reaches end of mountainside in significantly less steep areas (<< 30°)
4 (Very large)	Could destroy and bury trains and big trucks, several buildings, and forest-areas.	< 100,000 m ³	Avalanche continues over 50 meters when it reaches end of mountainside in significantly less steep areas (<< 30°) and reaches the bottom of the valley.
5 (Extremely large)	Largest snow avalanche known. Disastrous damage is possible.	> 100,000 m ³	Reaches the bottom of the valley. Largest avalanches known to mankind.

The avalanche sizes of Table 3.1 are logarithmic, similar to Richter scale, meaning that a size 3 avalanche is ten times bigger than an avalanche of size 2. Consequently, avalanches of sizes 4 and 5 are enormous. These are not the main avalanche size involved in accidents of humans because most people do not travel in avalanche terrain when these avalanches happen. Thus the avalanche forecast most often deals with avalanches of sizes 1 – 3.

A major component in a daily avalanche forecast is the *avalanche danger*, which is a number between 1 and 5, where 5 is very high avalanche danger, and 1 is low (Figure 3.7). This scale is not linear, and an increase in avalanche danger is said to double the probability of an avalanche (NVE, n.d.-c).

European Avalanche Danger Scale (2018/19)				
Danger level	Icon	Snowpack stability	Likelihood of triggering	
5 very high		The snowpack is poorly bonded and largely unstable in general.	Numerous very large and often extremely large natural avalanches can be expected, even in moderately steep terrain*.	
4 high		The snowpack is poorly bonded on most steep slopes*.	Triggering is likely, even from low additional loads**, on many steep slopes*. In some cases, numerous large and often very large natural avalanches can be expected.	
3 considerable		The snowpack is moderately to poorly bonded on many steep slopes*.	Triggering is possible, even from low additional loads**, particularly on the indicated steep slopes*. In certain situations some large, and in isolated cases very large natural avalanches are possible.	
2 moderate		The snowpack is only moderately well bonded on some steep slopes*; otherwise well bonded in general.	Triggering is possible, primarily from high additional loads**, particularly on the indicated steep slopes*. Very large natural avalanches are unlikely.	
1 low		The snowpack is well bonded and stable in general.	Triggering is generally possible only from high additional loads** in isolated areas of very steep, extreme terrain*. Only small and medium natural avalanches are possible.	

* The avalanche-prone locations are described in greater detail in the avalanche bulletin (altitude, slope aspect, type of terrain):

- moderately steep terrain: slopes shallower than about 30 degrees
- steep slopes: slopes steeper than about 30 degrees
- very steep, extreme terrain: particularly adverse terrain related to slope angle (more than about 40 degrees), terrain profile, proximity to ridge, smoothness of underlying ground surface

** Additional loads:

- low: individual skier / snowboarder, riding softly, not falling; snowshoer; group with good spacing (minimum 10m) keeping distances
- high: two or more skiers / snowboarders etc. without good spacing (or without intervals); snowmachine; explosives

natural: without human influence

Figure 3.7: Standard European avalanche danger scale (EAWS, 2022).

Varsom uses the ADAM matrix to find this avalanche danger level (NVE, n.d.-a). The inputs of the matrix are the *stability of the snowpack*, *distribution (of the releasability)*, and the *biggest expected avalanche size*, see Figure 3.8. Here the stability and distribution lead to the letters A, B, C, D, or E, which summarizes a *probability* ("Sannsynlighet" in Figure 3.8). This combined with the *biggest expected avalanche size* will lead to the final avalanche *danger level*. From the 2022/23 season Varsom will use an updated version of the ADAM matrix.

The figure consists of two tables. The left table is a 4x3 grid where rows represent snow stability (Svært dårlig, Dårlig, Moderat, God) and columns represent distribution (Utløsharhet) (Få, Noen, Mange). The right table is a 5x5 grid where rows represent probability (Sannsynlighet) (E, D, C, B, A) and columns represent the biggest expected avalanche size (Størst forventet skredstørrelse) (Str. 1 (små), Str. 2 (middels store), Str. 3 (store), Str. 4 (svært store), Str. 5 (ekstrem e)). The color scale for both tables ranges from dark blue (low risk) to red (high risk).

Snowstability	Utløsharhet	Utbredelse (av utløsharheten)			Snøskredfare	Størst forventet skredstørrelse				
		Få	Noen	Mange		Str. 1 (små)	Str. 2 (middels store)	Str. 3 (store)	Str. 4 (svært store)	Str. 5 (ekstrem e)
Svært dårlig	Naturlig / Svært lett å løse ut	C	D	E	E	2	3	4	4	5
Dårlig	Lett å løse ut	B	C	D	D	2	3	3	4	4
Moderat	Vanskelig å løse ut	A	B	B	C	2	2	3	3	4
God	Svært vanskelig å løse ut			1	B	1	2	2	3	3

Sannsynlighet	Snøskredfare	Størst forventet skredstørrelse				
		Str. 1 (små)	Str. 2 (middels store)	Str. 3 (store)	Str. 4 (svært store)	Str. 5 (ekstrem e)
E	2	3	4	4	5	
D	2	3	3	4	4	
C	2	2	3	3	4	
B	1	2	2	3	3	
A	1	1	2	2	3	

Figure 3.8: Illustration of ADAM matrix, which uses snow stability, distribution, and expected avalanche size to determine the danger level (NVE, n.d.-a).

Observers in the field

Varsom has to gather data on snowpack and the weather. Weather information is gathered from meteorological systems and is updated automatically through their internal forecast tool, the Avalanche Problem Solver (APS). This registers snow depth, new snow, wind, and more, but does not give a detailed view of the snowpack. For this job, Varsom educates professional observers to perform snow analysis in the field. They inspect the snowpack for grain forms, recent avalanche activity, weather observations, stability tests, and give their thoughts on the avalanche danger in their specific area (NVE, 2019). Stability tests are standardized tests, often by clapping with a snow shovel on top of an isolated column of snow. Extended Column Test (ECT) is an example of a stability test and was first proposed by an Israeli physicist, Ron Simenhois. The ECT is performed by isolating a 90 cm wide and 30 cm deep column, to about the depth of 1 meter, see Figure 3.9. 1 meter is because humans impact the snow approximately 1 meter down into the snowpack. If the performer suspects a weak layer buried at 120 cm, the isolated column is dug down below this. Then the executor puts their shovel on top of the column, and taps with their hand on the shovel blade (with the help of gravity) from their wrist 10 times, elbow 10 times, and shoulder 10 times. The executor then registers the number of hits needed for a fracture to occur, the surface of where the weak layer is (*e.g.*, shear quality and bed surface), and the structure of the snowpack (Temper, 2018).

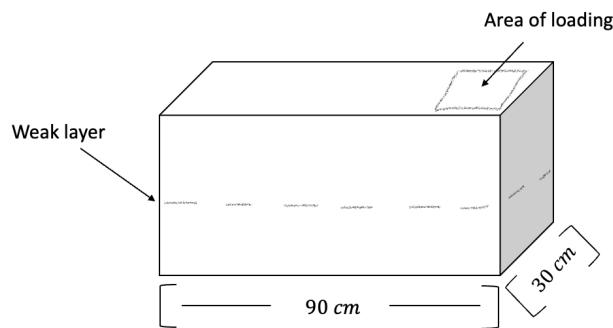


Figure 3.9: Illustration of Extended Column Test (ECT).

As mentioned, Varsom publishes more than just the avalanche danger. The avalanche danger level, in itself, is just a number with a color. Varsom also publishes an avalanche problem, which often

yields more information about the current status of the snowpack. This information is shown to be more important for mountaineering experts than the avalanche danger level because it tells users what to be aware of on that given day (Landrø et al., 2020).

3.2.2 Avalanche problems

The avalanche danger level, the numerical value between 1 – 5 (Figure 3.7), is a short description of the daily avalanche conditions. The forecast, and its users, need more information than this number. It's been standardized that the avalanche forecasts describe one or more avalanche problems. This was done by the European Avalanche Warning Services (EAWS). These avalanche problems give a more detailed description of what the conditions are actually like. The avalanche problem includes the following descriptive properties (NVE/EAWS, 2018):

- The *type* of avalanche that is potentially expected.
- Characterizations of the problem.
- The *location* of this potential avalanche, and its distribution in the terrain.
- The *likelihood* of the avalanche being triggered or released.
- The expected *size* of a potential avalanche.
- Where in the snowpack the weak layer is.
- What could trigger an avalanche.
- How to identify the avalanche problem.
- How long the problem will endure.

Varsom, which uses EAWS's definitions of *Typical avalanche problems*, focuses more on how to avoid and handle the given avalanche problem (NVE/EAWS, 2018). The five typical avalanche problems given in the Norwegian forecast are:

1. New snow
2. Wind-drifted snow
3. Persistent weak layers (PWL)
4. Wet snow
5. Gliding snow

To summarize, it is essential to note that if humans are outside of *avalanche terrain* (starting zone and runout zone) there is no hazard of being buried, and therefore no danger, see Figure 3.6. Varsom easily states it as follows:

$$\text{Avalanche terrain} + \text{Avalanche problem} = \text{Dangerous} .$$

So far this chapter has given a short description of different aspects of the avalanche terminology and the forecast. This thesis focus on forecasting the specific avalanche problem PWL. To get a theoretical understanding of how these weak layers form, the next section focuses on snow science and metamorphism of snow grains.

3.3 Snow science

So far we have discussed general avalanche types, introduced some terminology and the avalanche forecast. An important factor behind all this, and also the main part of this thesis, is how these avalanche problems emerge. To understand this, we discuss how snow crystals transform at a molecular level.

When avalanches happen, *i.e.*, slab avalanches, they occur because of differences between the layers in the snowpack. A *snow layer* can be defined as a horizontal layer within the snowpack with equal *grain form*, see Figure 3.10.

KORNFORMER (GRAIN FORMS)

	+	PP	Nysnø (Precipitation Particles)
	/	DF	Delvis nedbrutt (Decomposing and Fragmented Particles)
	•	RG	Avrundede korn (Rounded Grains)
	□	FC	Kantkorn (Faceted Crystals)
	^	DH	Begerkrystaller (Depth Hoar)
	▽	SH	Overflaterim (Surface Hoar)
	○	MF	Smelteformer (Melt Form)
	∞	MFcr	Smelte-fryse skare (Melt Freeze Crust)
	-	IF	Islag (Ice Formations)

Figure 3.10: Illustration of grain forms by Georg Sjoer (Müller, 2020).

The thermodynamic relationship between the air and the snowpack determines how quickly different snow layers form. Wind, rain, new snow, sun, and temperature are the main reasons to why snow transforms from *precipitation particles* into different grains. NVE published a paper called *Snow-transformation*, which focuses on three different ways snow can transform:

1. *Weak-gradient transformation/isothalmic transformation.*
2. *Kinetic transformation.*
3. *Wet snow transformation.*

The *kinetic transformation* will be of main interest in this thesis, because it is responsible for the formation of *depth hoar*, *faceted crystals (facets)* and *surface hoar* (Müller, 2020). These grain forms lead to the avalanche problem persistent weak layer (PWL).

3.3.1 Kinetic transformation - Persistent Weak Layers

Snow is formed in the clouds by the deposition of water vapor on small dust particles. These precipitation particles can come in various shapes, one example is seen in Figure 3.10. Throughout the winter season, snow will accumulate into a complex, stratified snowpack with different snow grains. As meteorological parameters change, *i.e.*, temperature, wind, and precipitation, snow crystals change and transform. Snow is thermodynamically unstable because its often close to its melting temperature (Müller, 2020). Snow crystals can transform into different kinds of grains by *wet snow metamorphism*, *weak gradient*, or *kinetic transformation*, see Figure 3.11.

Kinetic transformation, or *strong gradient metamorphism*, is the reason behind the formation of PWLs. Facets and depth hoar form within the snowpack and is discussed first.

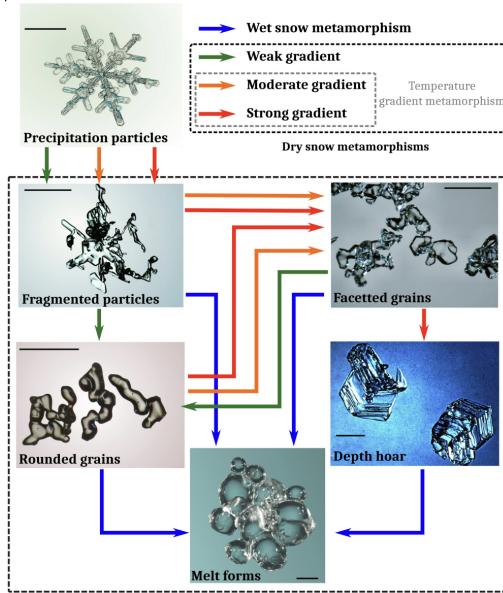


Figure 3.11: Illustration of snow-metamorphism, and how precipitation particles can transform into other grain types (Granger, 2019).

The orange and red arrows in Figure 3.11 illustrate kinetic transformation or *faceting*. Water molecules can move from areas in the snowpack with high water vapor pressure to be deposited in regions where it is lower water vapor pressure. Temperature differences within the snowpack are the biggest reason for these differences in pressure (Müller, 2020). Then snow crystals can become faceted crystals, and if the process endures over time, it can develop from faceted crystals to depth hoar, as seen in Figure 3.11 (Granger, 2019). These temperature differences within the snowpack can be written as

$$\text{Temperature gradient} = \frac{\text{Temperature difference}}{\text{Distance in the snowpack}}. \quad (3.1)$$

A property that causes high temperature gradients is that the ground's temperature is always approximately 0°C . Hence if the snowpack is thin, *e.g.*, $0,5\text{ m}$ and the surface temperature is -10°C (Figure 3.12), then the temperature gradient becomes

$$\text{Temperature gradient} = \frac{10^\circ\text{C}}{0,5\text{ m}} = 20^\circ\text{C m}^{-1}.$$

If the temperature gradient is bigger than 10°C m^{-1} over time, then kinetic transformation will be the predominant transformation happening, and facets can form (Müller, 2020). These weak layers are *persistent* because they endure over time and will need wet snow metamorphism or weak gradient metamorphism for the layer to be destroyed, see Figure 3.11.

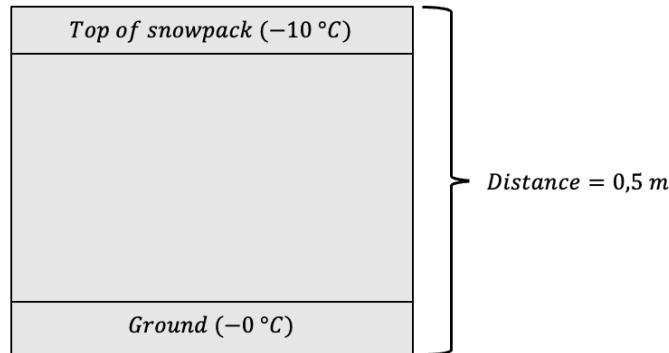


Figure 3.12: Example of conditions when the kinetic transformation will be the dominant transformation within the snowpack.

As mentioned in the previous paragraph, the *temperature gradient* is an important factor in the formation of weak layers created by kinetic transformation. It is important to note that this is the temperature within the snowpack, not the air temperature, although they are related.

In addition to facets forming when the temperature gradient is bigger than $10^{\circ}C\ m^{-1}$ over time, facets can form rapidly close to the surface of the snowpack in three different ways:

1. *Dry snow on top of wet snow.*
2. *Fluctuating temperatures throughout the day.*
3. *Balanced radiation and radiance.*

Dry snow on top of wet snow can cause an extreme temperature gradient, hence speeding up the process of forming facets. This can happen when there has been a warm period, meaning that the snowpack is wet from either high temperatures or rain. If the temperature suddenly drops and cold snow falls on top of this wet snowpack, then the temperature gradient in the transition from wet, old snow to new, cold snow, suddenly gets very high. *Fluctuating temperatures throughout the day* causes the same sped-up formation of facets. This happens because the snowpack gets warm during the daytime and cools when the night approaches. This is a typical phenomenon in the Alps and in springtime in Norway. This causes a very large temperature gradient in the top section of the snowpack, and PWLs can form. *Balanced radiation and radiance* cause a high temperature gradient in the top section of the snowpack. This happens because most of the radiation (shortwaved radiation) is reflected due to the high albedo of the white snow, but some radiation is captured and warms up a section of the snow underneath the surface. While this happens, longwaved radiance causes the snow's surface to cool down. This creates a large temperature gradient in the upper section of the snowpack and the kinetic transformation will dominate (Müller, 2020).

Faceted crystals and depth hoar have been the center of attention in this section. However, surface hoar (Figure 3.10) is also created by kinetic transformation, but not by a temperature gradient within the snowpack. *Surface hoar* is created when water vapor in the air hits the cold snow surface. This happens when there is enough available water vapor, a breeze (approximately 1-3 m/s), and clear skies during the night (Müller, 2020). A clear sky during the night is a key factor because then the radiance is at its largest. If there is no wind at all, the water vapor will not be transported towards the cold snow surface. If there is too much wind, these fragile crystals will be destroyed. It is important to note that surface hoar is not dangerous unless it is followed by a snowfall. Especially if this snowfall happens during a windstorm. Then the wind creates a slab on top of this very weak layer. If it is steep enough, and *e.g.*, a skier triggers an avalanche, then it can create large, slab avalanches.

Avalanche climates

The snow condition within a region is dependent on the local climate, and different climates cause different trends in snow conditions. In avalanche theory, one uses the terminology *avalanche climates*, to split mountainous regions into three categories:

- *Maritime*
- *Intermountain*
- *Continental*

Weak layers occur in all climates, but generally, the persistence of these weak layers differs in the listed avalanche climates. In continental regions, weak layers form and persist over a long period. This is due to stable cold temperatures and less precipitation. Hence we get a thin snowpack, resulting in a large temperature gradient. Maritime regions usually have a thicker snowpack, and more variable temperatures. This causes midwinter rain to occur every now and then. As a consequence, wet snow metamorphism will dominate the snowpack, transforming facets into melt forms, see Figure 3.11. Hence maritime weak layers do not *persist* as long as in continental regions with stable, cold temperatures. *Intermountain* is in-between maritime and continental regions when it comes to its properties with PWLs. Meaning these layers usually persists longer than in maritime regions, but shorter than in continental regions (Temper, 2018).

Chapter 4

The dataset and preliminary analysis

This chapter will present the dataset and how it was gathered, notated, and processed. Additionally, we will present a preliminary analysis that showcase trends in the dataset concerning PWLs, and potential feature values to be used in Chapter 6.

4.1 Data

The avalanche forecast on a given day is published on <https://varsom.no/snorskred>. This daily forecast is available to the open public before 16:00 the preceding day and, if necessary, updated before 10:00 on the given day. This is done if the forecasters think it is necessary based on, *e.g.*, overnight weather or observations in the field.

The data used in this thesis is for the open public, but for the purpose of this work, NVE helped provide weather data, avalanche danger, and avalanche problems for different regions. The data was fetched from the Python-library *regobslib*, gathering data from the winter season 2017/18 to the end of the season 2021/22, see Appendix A.1. This is because preceding 2017, the forecast was different from after 2017. The avalanche problem PWL was fetched from Varsom, and the weather data was gathered from a frequently updated, automatic weather forecast called Avalanche Problem Solver (APS). The Python code (Appendix A.1) shows how this data was retrieved using CSV files containing avalanche danger, avalanche problem, and weather data.

4.1.1 Notation of data

In the data, the *avalanche danger* was notated with d_{region} , where the regions were TROMSO, JOTUNHEIMEN, ROMSDAL and SUNNMORE. The weather data was structured with five rows of data for each day, at different elevations, per feature. For Romsdal and Sunnmøre, the regions were split into different elevation intervals

- 0 – 400 m
- 400 – 800 m
- 800 – 1200 m
- 1200 – 1600 m
- 1600 – 2000 m

For Tromsø, the intervals were 200m and 500m for Jotunheimen. This is due to the different elevations above sea level.

At each elevation interval, the variables were:

- precip (mm): Mean precipitation in the region.
- precip_{\max} (mm): Maximum precipitation within an area with the most precipitation within the region.
- temp ($^{\circ}\text{C}$): Mean temperature in the region.
- $\text{snow}_{\text{depth}}$ (mm): Measured snow depth in the region.
- $\text{new} - \text{snow}$ (mm): Mean new snow in the region.
- $\text{new} - \text{snow}_{\max}$ (mm): Maximum new snow within an area with the most new snow within the region.
- wind (m/s): Mean wind in the region.
- wind_{dir} : Wind direction in the region.

All these variables (except wind_{dir}) were given in percentiles; 0, 5, 25, 50, 75, 95, and 100. 0 is the minimum, 100 is the maximum, and 50 is the mean. 25 and 75 are the first and third quartiles. The mean and maximum were the most used percentiles. This was done to limit the number of features used.

The variable wind_{dir} was split into 0, 1, 2, ..., 7. Where $N = 0$, $NE = 1$, $E = 2$, ..., $NW = 7$. Each number between 0 and 7 had a number between 0 and 1, indicating the dominant wind-direction on a given day at a given elevation.

The data on avalanche problems for the different days throughout the season between the 1st of December and 31st of May yield information about what avalanche problem(s) was forecasted on a given day, expected size, exposition, and elevation the problem exists, to mention a few. This is data determined by experts in Varsom and will be used as 'reference' or our *response*, i.e., 1 if PWL is forecasted and 0 otherwise. The weather data and snow observations from avalanche professionals in the field are the data used to determine these avalanche problems.

From Chapter 3.3.1, we know that the *temperature gradient* within the snowpack is one of the critical factors in the formation of PWLs. Our weather data are not observations within the snowpack, but meteorological parameters and snow depths. Regardless, these parameters have an indirect influence on the temperature gradient. Both the temperature and snow depth are essential features in calculating the temperature gradient, see Equation 3.1. Therefore are $\text{snow}_{\text{depth}}$ (mm) and temp ($^{\circ}\text{C}$) reasonable features to be used in a model.

4.1.2 Processing of data

Python was used to fetch the data for this thesis, but most of the work was performed in RStudio. The data needed processing before it could be implemented in the methods discussed in Chapter 5, and applied in Chapter 6.

Much work was put into processing the available data for this thesis. Many available features (with percentiles) could have been used, but this thesis focuses on PWL(s). Hence air temperatures and snow depths were considered the most important features, see Chapter 3.3.1. The dataset was lacking snow depth registrations for the first 30 days in the 2017/18 seasons. Other than that, the dataset was consistent. The three CSV files, gathered from Python code in Appendix A.1, needed to be translated to a single data frame in R consisting of rows with a response Y_i (no PWL (0)/PWL (1)) and its features ($\mathbf{X}_i = X_{i1}, X_{i2}, \dots, X_{ip}$). The avalanche problem PWL was translated into a column with 0 or 1 for each day between the 1st of December and to 31st of May, from the 2017/18 season to 2021/22. The weather data was structured similarly, but each day included five rows with different elevation intervals. This problem was solved by iterating through the data such that each elevation interval got its own column.

In this thesis we will use three different datasets:

- One dataset consisting of air temperatures at the different elevation intervals (5 features).
- One dataset consisting of air temperatures and snow depths at the different elevation intervals (10 features).
- One dataset consisting of air temperatures and snow depths at the different elevation intervals including the five preceding days (60 features).

There are many potential combinations of features, but we chose these three based on the avalanche literature, see Chapter 3.

4.2 Preliminary analysis of data

After the data was processed, it needed a preliminary analysis to determine what regions to investigate. The regions Jotunheimen, Sunnmøre, Romsdal, and Tromsø were chosen in the starting phase of this thesis work to investigate what avalanche climate could be of interest.

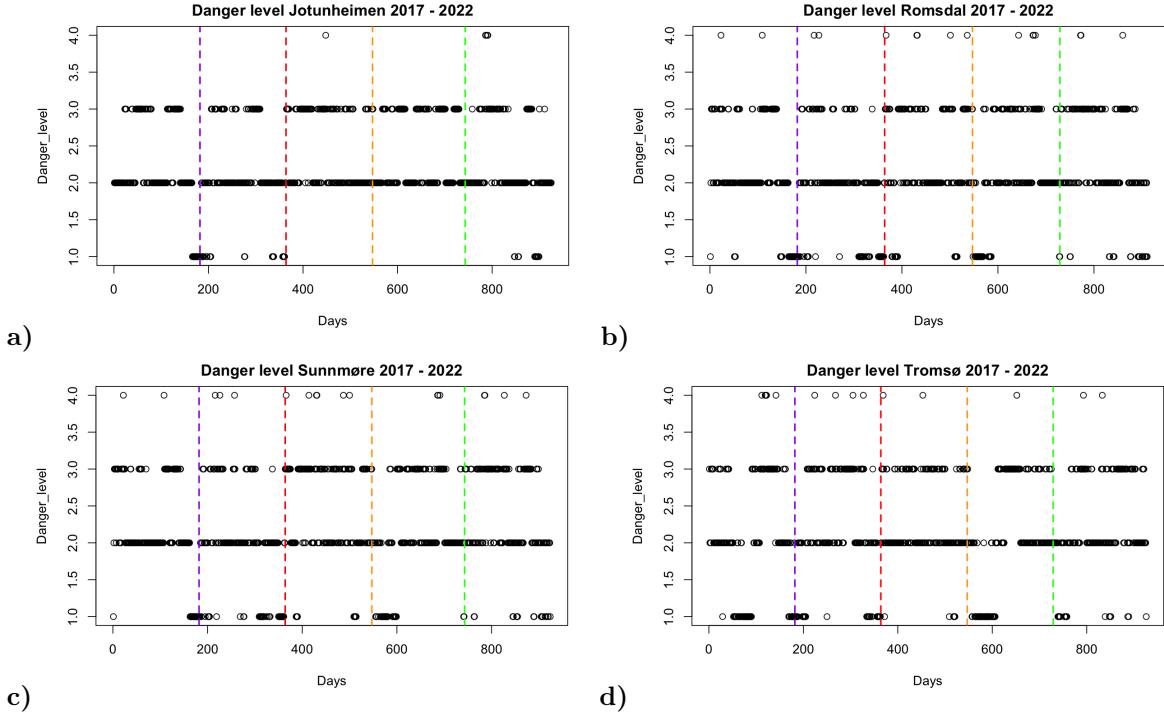


Figure 4.1: Plot of the different danger levels throughout the last five seasons for Jotunheimen (a), Romsdal (b), Sunnmøre (c), and Tromsø (d). Dotted, colored lines split the seasons.

Figure 4.1 shows different plots illustrating the danger level throughout the last five seasons in four different regions in Norway. The avalanche danger level is (for the majority) 2 or 3. Hence a statistical model to predict the avalanche danger is not of great interest, but what type of avalanche problem(s) that exists in a region could be more interesting. Predicting the avalanche problem, *e.g.*, *wind-drifted snow (slab)* would be relatively simple. For example, if its been snowing and windy from, *e.g.*, SW we would expect the avalanche problem of wind-drifted snow in the opposite cardinal direction (*i.e.*, NE). However, predicting PWL is a more difficult task because how it comes about, see Chapter 3.3.1. Figure 4.2 shows if PWL is forecasted on a given day (1) or not (0) for the four regions, Sunnmøre, Jotunheimen, Romsdal, and Tromsø.

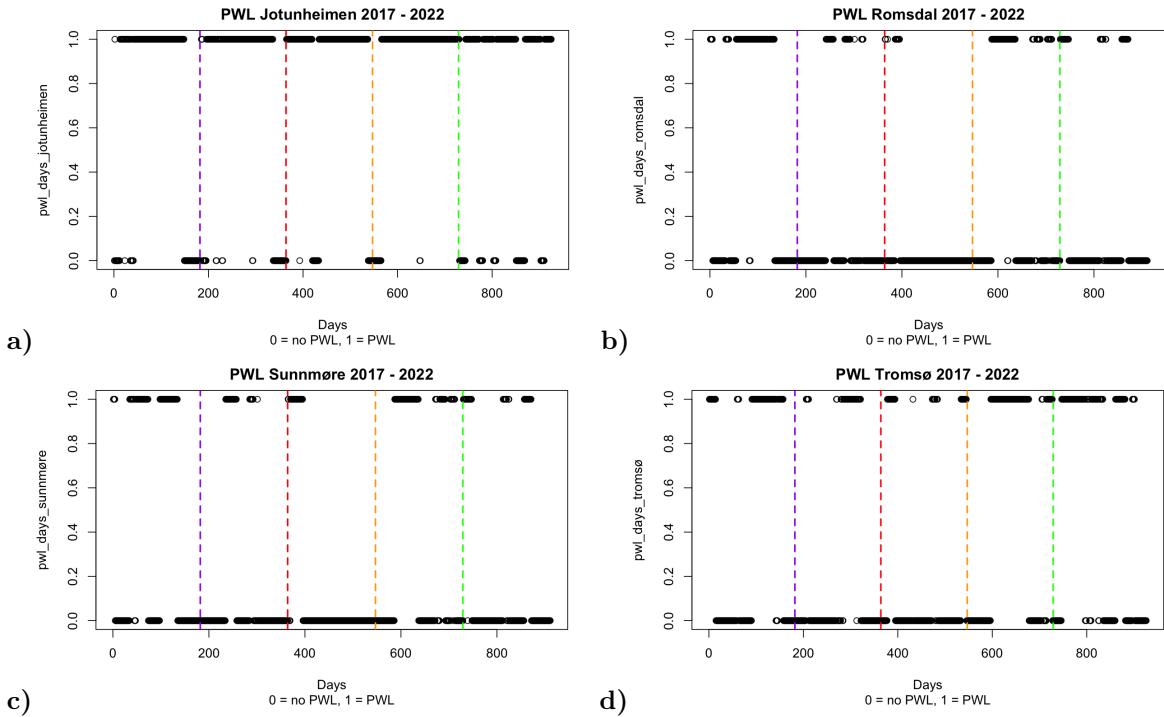


Figure 4.2: Plot illustrating the occurrence of the avalanche problem PWL in Jotunheimen (a), Romsdal (b), Sunnmøre (c), and Tromsø (d). 1 represents that PWL was forecasted on that given day, 0 otherwise. Dotted, colored lines split the seasons.



Figure 4.3: Map section showing the geographical location of the four regions mentioned in this chapter.

Figure 4.3 shows where the four regions are located in Norway. Tromsø is far up in the arctic region, which is an *intermountain* area, meaning that the region is intermediately influenced by being close to the ocean (Temper, 2018). Further, Jotunheimen is a *continental* region characterized by PWL being a general avalanche problem for the majority of the winter season, see Figure 4.2a.

Romsdal and Sunnmøre are *maritime* regions and are often characterized by PWL not being the predominant avalanche problem. This is because coastal temperatures and midwinter rain can destroy the persistent weak layer (facets, depth hoar, or surface hoar). Figures 4.2b and 4.2c confirm that PWLs do occur in these maritime regions, but the plots show that the weak layers do not persist as long as in the continental region Jotunheimen, see Figure 4.2a. To confine the task of this thesis to a reasonable and achievable task, we chose to focus on the aforementioned maritime regions.

Before performing a statistical analysis of the data, we looked further into the response and features of the datasets. First, we investigated the number of days the avalanche problem PWL was forecasted (*i.e.*, the 0s and 1s). This is visualized in the following pie plots, see Figure 4.4.

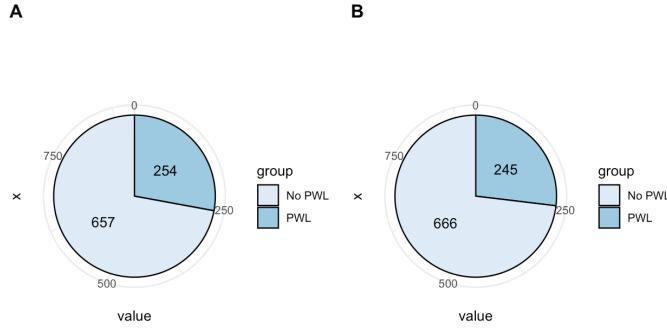


Figure 4.4: Number of days with and without PWL in Sunnmøre (A) and Romsdal (B).

When investigating the maritime regions, Sunnmøre and Romsdal, we observe that we do not have a balanced dataset, see Figure 4.4. Days without PWL occur more often than days with PWL, this is due to mild weather midwinter in these maritime regions. From this, we see that the number of days with PWL is very similar for the two regions, and this is confirmed by the plots, see Figure 4.2b and 4.2c.

As discussed in Chapter 3.3.1, the temperature gradient within the snowpack is the key to understanding when these PWLs come about. This temperature gradient depends on temperature and snow depth, and for this preliminary analysis, we looked into how these developed over time. Figure 4.5 shows how the temperature develops at the lowest and highest elevations throughout the 2017/18 - 2021/22 season. Here we can see that these features are dependent, which is expected since they are temperature measurements from the same regions.

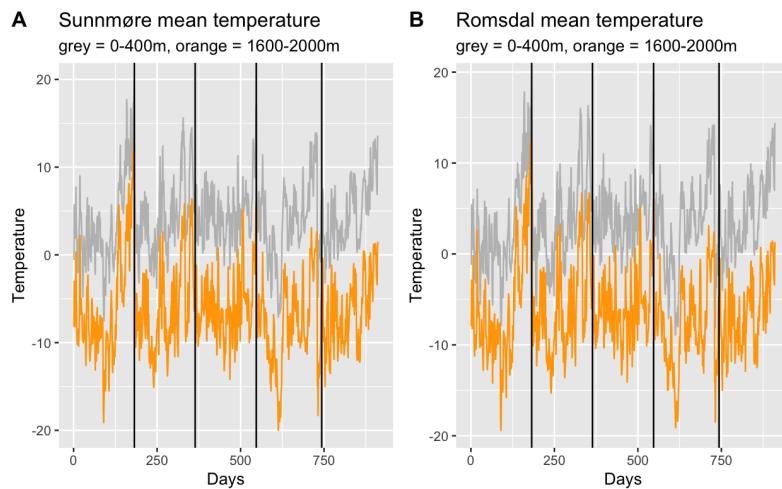


Figure 4.5: Temperature (*temp* (°C)) throughout the seasons in Sunnmøre (A) and Romsdal (B). The black vertical line splits the seasons. The other temperatures lie in between these intervals, but were kept out for better visualization.

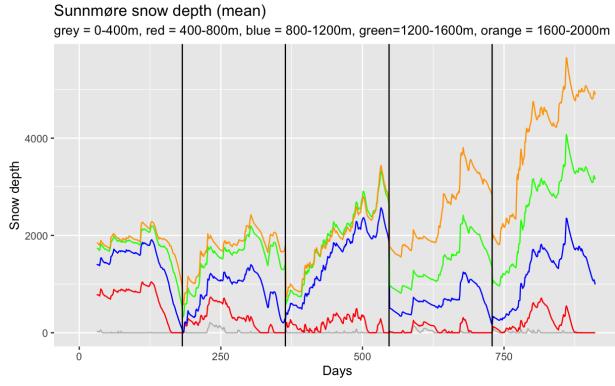


Figure 4.6: Snow levels ($snow_{depth}$ (mm)) at different elevation intervals in Sunnmøre. The black line splits the seasons.

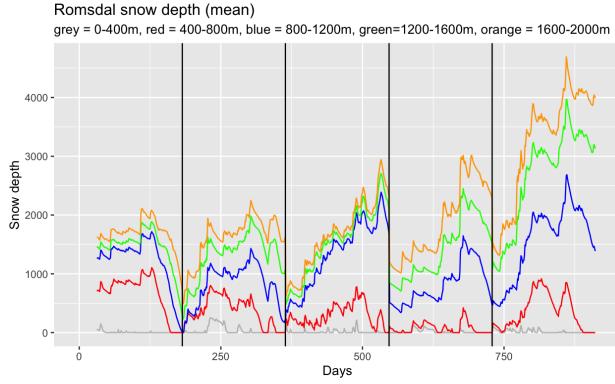


Figure 4.7: Snow levels ($snow_{depth}$ (mm)) at different elevation intervals in Romsdal. The black line splits the seasons.

The same goes for snow depths, see Figures 4.6 and 4.7. But how are these related to when PWL is forecasted? Figure 4.8 shows the first season of Sunnmøre and how the temperature fluctuates throughout the season, together with the occurrences of the avalanche problem PWL. The red dashed lines outline the time period 8th of March until the 13th of April. By inspection of this plot we observe that there was a cold period right before this PWL got forecasted. This makes sense because a temperature gradient has to be consistently high over several days, which happens during a cold period.

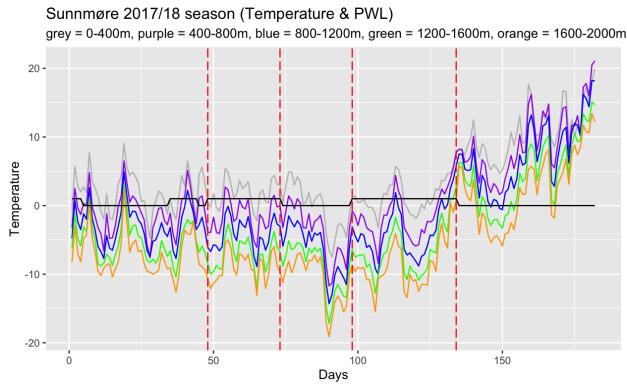


Figure 4.8: Plot of the different temperature intervals ($temp$ (°C)) and PWL for the first 2017/18 season in Sunnmøre. The black line is 0 (no PWL) or 1 (PWL). The red dashed lines show the time period 17.01.2018 - 11.02.2018 and 08.03.2018 - 13.04.2018.

In the second season of Sunnmøre, we see a similar trend to the first season, see Figure 4.9. Here we

have outlined a period between 22nd of January and 13th of February, where PWL was forecasted. Preceding to the 22nd of January it was a cold period. When this cold period endured, so did the forecast of PWL, until it suddenly became milder the 14th of February. This led to snow melting and probably caused water in liquid form to percolate the snowpack. Hence destroying the PWL within the snowpack by wet snow metamorphism.

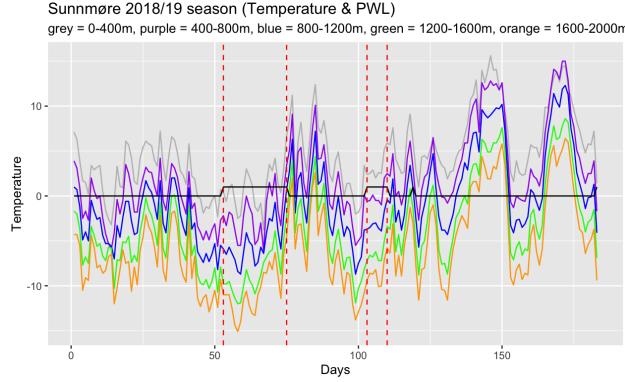


Figure 4.9: Plot of the different temperature intervals (temp ($^{\circ}\text{C}$)) and PWL for the second 2018/19 season in Sunnmøre. The black line is 0 (no PWL) or 1 (PWL). Red dashed lines show the time period 22.01.2019 - 13.02.2019 and 13.03.2019 - 20.03.2019.

A fairly similar trend is shown in Figures 4.10 and 4.11 for the 2017/18 and 2018/19 seasons in Romsdal. A period of forecasted PWL had a preceding cold period. The time of the event of the PWL forecasts was quite similar for Sunnmøre and Romsdal. This is probably due to the regions being close in geographical manners.

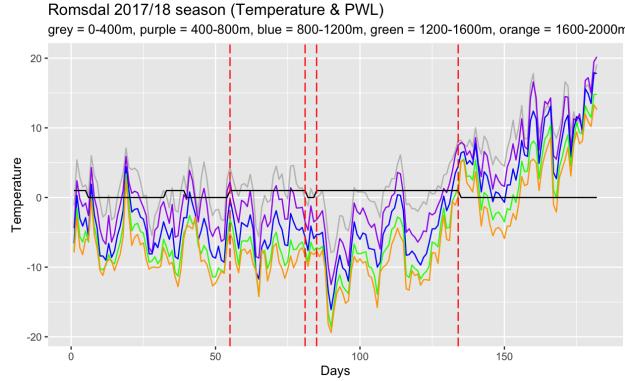


Figure 4.10: Plot of the different temperature intervals (temp ($^{\circ}\text{C}$)) and PWL for the first 2017/18 season in Romsdal. The black line is 0 (no PWL) or 1 (PWL). Red dashed lines show time period 24.01.2019 - 19.02.2019 and 23.02.2019 - 13.04.2019.

We have investigated different regions, and confined the task to maritime regions. Additionally, we looked into the occurrences of PWL in these two regions, Sunnmøre and Romsdal (Figure 4.4). This thesis focuses on creating different statistical and machine learning models using features mentioned in this preliminary analysis and measuring their predictive performance when trained on different training sets. The reason of why temp and snowdepth were the two features of attention is due to their indirect influence on the temperature gradient, the trends observed in Figures 4.8 to 4.11, and adding more features did not enhance the predictive performance. This is discussed more thoroughly in Chapter 6 and 7.

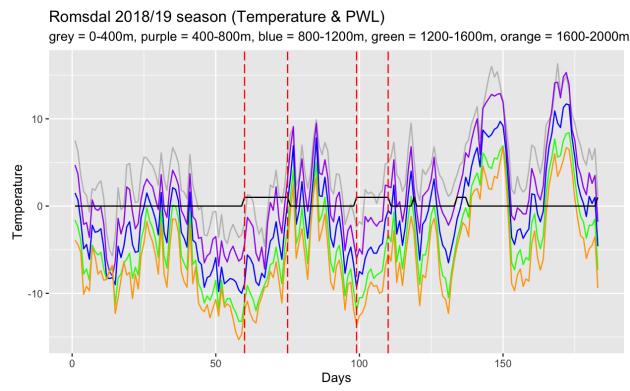


Figure 4.11: Plot of the different temperature intervals (temp ($^{\circ}\text{C}$)) and PWL for the second 2018/19 season in Romsdal. The black line is 0 (no PWL) or 1 (PWL). Red dashed lines show time period 29.01.2019 - 13.02.2019 and 09.03.2019 - 20.03.2019.

Chapter 5

Statistical learning

The goal of statistical learning is to understand something about available data. Statistical learning can be split into *supervised methods* or *unsupervised methods*. For this thesis, we used supervised methods, which means that the features and response have labels, and the goal is to link these features to a response. The response is binary, either existence of PWL (1) or the absence of PWL (0). This chapter will discuss some general concepts of supervised methods, and then go into more detail on specific methods. Table 5.1 shows a summary of the mathematical notation used in this chapter and throughout the thesis.

Table 5.1: Notation used in Chapter 5.

\mathbf{X}_i	Vector with p features for day i
p	Number of predictors
n	Number of responses
ϵ_i	Random error term that is i.i.d with mean 0 and constant variance σ_i^2
Y_i	Random variable for the response
\hat{Y}_i	Predicted variable of Y_i
$E[X]$	Expected value of X
$Var(X)$	Variance of X
$Cov(X_b, X_z)$	Covariance between X_b and X_z
R_j	Partition j of the feature space
m^*	Number of partitions of feature space
RSS	Residual sum of squares
y_i	Observed response for day i
\hat{y}_i	Predicted response for day i
\hat{y}_{R_j}	Predicted response for region R_j
$CC(Tree)$	Cost-complexity function
$I(y_i = k)$	Indicator function for class k
η	Number of terminal nodes
d_i	Dummy variable
$CV_{\text{average}}^{(k)}$	k -fold cross validation estimate
D_j	Purity measures
$\hat{f}_b(x)$	The b^{th} classification tree
B	Number of trees in bagging/random forest/boosting
m	Number of predictors considered in each split (bagging/random forest)
$T_{\text{bagg}/RF}^B$	B classification trees aggregated together (bagging/random forest)
d	Interaction depth for boosting
λ	Shrinkage parameter
z_i	The gradient in boosting
$g(\cdot)$	Activation function
w_{kj}	Weights of neural network
A_q	Activation of the hidden units

5.1 Supervised learning

In *supervised learning*, we want to use some inputs to get an output. If we have the input variables \mathbf{X}_i , called *predictors* or *features*, we then want to get an output variable called *response* or *dependent variable*, often denoted with Y_i .

We denote p features by $\mathbf{X}_i = (X_{i1}, X_{i2}, \dots, X_{ip})$. Then the relationship between the response Y_i and predictors \mathbf{X}_i can be written as

$$Y_i = f(\mathbf{X}_i) + \epsilon_i, \quad i = 1, \dots, n, \quad (5.1)$$

where f is an unknown function linking the predictor \mathbf{X}_i to the response Y_i , and the random error term ϵ_i is independent and identically distributed with mean zero and constant variance σ_i^2 . We often wish to estimate this unknown function with \hat{f} , which can be done with two purposes in supervised learning: *prediction* or *inference*. Prediction is used when we want to summarise something about available data to an output Y . The error-term ϵ_i will average to zero, due to $E(\epsilon_i) = 0$ for all i . Hence we can predict Y_i by

$$\hat{Y}_i = \hat{f}(\mathbf{X}_i) \quad i = 1, \dots, n, \quad (5.2)$$

where the shape or function of \hat{f} is not of main interest. We want as good predictions \hat{Y}_i of Y_i as possible. How good of an estimate \hat{Y}_i is of Y_i can be explained by looking at the *expected value* of the squared difference between the predicted response \hat{Y}_i and Y_i . This expected value depends on the *reducible* and *irreducible error* and can be derived in the following manner

$$\begin{aligned} E(Y_i - \hat{Y}_i)^2 &= E[f(\mathbf{X}_i) + \epsilon_i - \hat{f}(\mathbf{X}_i)]^2 \\ &= \underbrace{[f(\mathbf{X}_i) - \hat{f}(\mathbf{X}_i)]^2}_{\text{reducible}} + \underbrace{\text{Var}(\epsilon_i)}_{\text{irreducible}}. \end{aligned} \quad (5.3)$$

Equation 5.3 shows that we can *reduce* the reducible term $[f(\mathbf{X}_i) - \hat{f}(\mathbf{X}_i)]^2$ with a good approximation \hat{f} . In contrast, $\text{Var}(\epsilon_i)$ is not possible to reduce, hence *irreducible*. When performing *predictions*, the art lies in finding a good \hat{f} such that we reduce this squared difference as much as possible. *Inference*, in contrast to prediction, is used when the goal is to understand which predictors are important for the response. In inference, the shape or form of f is essential to inform users of the relation between the features \mathbf{X}_i and the response Y_i (James et al., 2013).

Binary classification - the focus of this thesis

The paragraphs above are for regression problems, but the ideas are similar for classification problems. This thesis will focus on binary classification, where the output is either 0 or 1, *e.g.*, no PWL forecasted on a given day or PWL forecasted on a given day. Hence in the case of our application, each vector $\mathbf{X}_i = (X_{i1}, X_{i2}, \dots, X_{ip})$ with the weather data has a corresponding response Y_i

$$Y_i = f(\mathbf{X}_i) = \begin{cases} 0 & \text{if no PWL on day } i, \\ 1 & \text{if PWL on day } i, \end{cases} \quad i = 1, \dots, n. \quad (5.4)$$

The predictions \hat{Y}_i can then be described as

$$\hat{Y}_i = \hat{f}(\mathbf{X}_i) = \begin{cases} 0 & \text{if no PWL predicted on day } i, \\ 1 & \text{if PWL predicted on day } i, \end{cases} \quad i = 1, \dots, n. \quad (5.5)$$

5.2 Splitting of dataset

In statistical and machine learning we need data to make an *algorithm* or *predictive model*. The algorithm then has to be trained on some data, meaning we fit a given model using a specific subset of the data. This introduces the concept of splitting our collected dataset into a *training set*, a *validation set*, and a *test set* (James et al., 2013).

As the name indicates, the training set is the data used to train our desired algorithm, and hopefully find patterns in the data linking the data to its response. The validation set, sometimes called the *hold-out set*, is used to validate our algorithm created on the training set. This means we fit our model to our validation set to predict the response (James et al., 2013). Then we can improve our algorithm based on the results of the validation set, and finally test it on our test set. Although there exists no unique way to split the dataset correctly, the training set needs to be the biggest because this task is the most demanding. However, if the predictive model is fitted *too well* to our training data, it can make the model *overfit* the data. Overfitting implies that the algorithm performs very well on the data it has been trained on, but performs poorly on new test data. This proves the necessity of verifying how well the algorithm performs when it receives unseen test data (Ying, 2019).

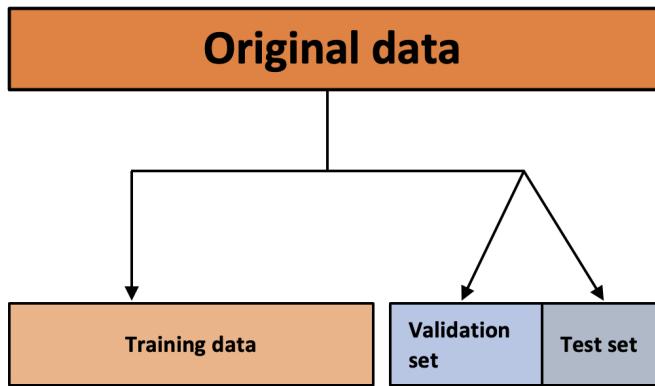


Figure 5.1: Illustration of how a dataset can be split into training, validation, and test set.

When performing splitting of the dataset, as in Figure 5.1, we might unintentionally perform the splitting in a manner that gives an unreasonable good or bad test error. Cross-validation (CV) is often used in supervised learning to answer how well a learning method will do on new, independent data. It can be implemented in several different ways, and *k-fold cross-validation* is one example. The entire dataset (\mathbf{X}_i, Y_i) , $i \in [1, n]$ are then split into k non-overlapping subsets. Firstly, one of the k subsets is treated as the test set, and the rest are used for training. Then we can compute the test error for that particular test set. This is repeated k times such that each non-overlapping subset is used as a test set. Then we can compute the average test error over all k subsets to indicate how we might expect our learning algorithm to perform on new data (James et al., 2021). This average test error can be written as

$$\text{CV}_{\text{average}}^{(k)} = \frac{1}{k} \sum_{i^*=1}^k \text{Test error}_{i^*} . \quad (5.6)$$

5.3 Decision trees

In Chapter 2, we discussed some previous related work on automating the avalanche forecast, predicting deep slab avalanches, and avalanche activity. All used classification trees as one of their methods in their papers, which motivated us to try this learning method on the dataset at hand.

Decision trees are a supervised learning method known for its intuitive display, and being user-friendly for a variety of problems. It is a powerful way to perform predictions and is evolved in both

statistical and machine learning problems (Ville, 2013). Trees need few assumptions compared to other statistical methods and can often produce good results. Especially when combined with other concepts, such as bootstrapping (James et al., 2013).

Decision trees rely on splitting the *feature space* into non-overlapping regions. If we look at a one-dimensional vector of predictors X_1, X_2, \dots, X_p , then we divide the feature space into R_1, R_2, \dots, R_m distinct regions, see Figure 5.2. For every observation X_i that falls in a specific region R_j we make the same prediction for that X_i .

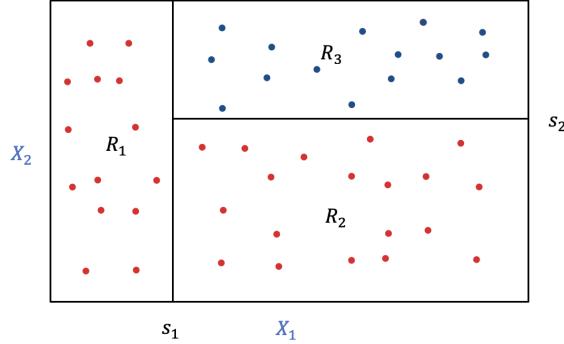


Figure 5.2: Example of splitting of the feature space in three partitions. For this binary example, blue represents 1s and red represents 0s.

There are too many possible partitions of the feature space to consider all of them. It would simply be too computer-intensive. A solution to this is *recursive binary splitting*. This means that the first split is a point s . This creates a *root node*, where the response is either larger or equal to the value of this splitting point, or lower. Then the feature space is divided into $R_1 = \{X|X_j < s\}$ and $R_2 = \{X|X_j \geq s\}$, where s is chosen to fit a splitting criterion which is dependent on whether we want regression or classification trees. This is done recursively by splitting the partitions R_j into finer and finer regions, where each R_j represents a terminal node. For example, in Figure 5.2, if X_1 represents *Wind* ($s_1 = 5 \text{ m/s}$) and X_2 represents *Snowing* ($s_2 = 5 \text{ mm}$), then three regions can be denoted as $R_1 = \{X|X_1 < s_1\}$, $R_2 = \{X|X_1 \geq s_1 \text{ and } X_2 < s_2\}$ and $R_3 = \{X|X_1 \geq s_1 \text{ and } X_2 \geq s_2\}$. These are represented in the sense of a classification tree in Figure 5.3. Note that these splits are only done within existing partitions and cannot intersect each other (James et al., 2021). The different splitting criteria are discussed later in this chapter.

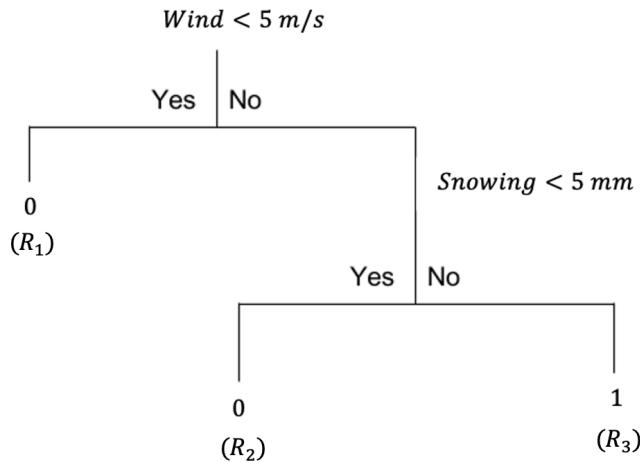


Figure 5.3: Illustration of a simple classification tree to check whether wind-drifted snow (slab) is forecasted as an avalanche problem (1) or not (0).

Tree methods can also perform well if we have missing data problems. This can be solved in different ways depending on the reason the data is missing (Faraway, 2006, pp. 253 - 255). In our

case, there are only a few missing feature values for a response, hence we can omit the entire row \mathbf{X}_i since the problem at hand only makes sense with all feature values present for each response Y_i .

There are several advantages to using decision trees as a predictive tool. Trees can be applied for both *classification* and *regression* problems. It is simplistic to explain to the outside world, and when the response is a factor we do not need any dummy variables. That is a binary variable d_i that specifies whether something is present or absent,

$$d_i = \begin{cases} 0 & \text{if the } i^{\text{th}} \text{ effect is absent} \\ 1 & \text{if the } i^{\text{th}} \text{ effect is present} \end{cases}, \quad (5.7)$$

and is often used in regression problems (Fahrmeir et al., 2013). Although there are a lot of positives to using trees, they are not necessarily a very *robust* method. This can be shown if there is a slight change in the training data, especially if the dataset is small, then the predictive model it yields can change dramatically (James et al., 2013).

5.3.1 Regression trees

Regression trees are used in both statistics and computer science. These trees are implemented when the response is continuous, typically on the real line. First we have to split the feature space into the non-overlapping R_1, R_2, \dots, R_{m^*} regions. Then for each of the observations \mathbf{X}_i that falls into a partition R_j , we take the mean of the response in that region and this mean value is the prediction for \mathbf{X}_i . This is done for all \mathbf{X}_i observations. Then we calculate the residual sum of squares (RSS) of R_j , and we wish to find the partitions R_j , $j = 1, \dots, m^*$ such that we minimize

$$RSS = \sum_{j=1}^{m^*} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2, \quad (5.8)$$

where y_i is the true response, and \hat{y}_{R_j} is the estimated response for a region R_j (James et al., 2021).

These trees are grown by starting with a root node, and then splitting the tree into finer and finer branches. However, they can quickly become very large and hence *fitting the data hard*. This means that we fit a tree *too well* to its training data, and a smaller tree with fewer branches might perform better on unseen test data. One might think cross-validation (CV) is a good way to consider different subsets of our tree to find an optimal size for our tree. But performing CV on every possible combination of splits will quickly become very computationally expensive. A solution to this is the *cost-complexity function*

$$CC(Tree) = \sum_{\text{terminal nodes: i}} RSS_i + \eta \cdot m^*, \quad (5.9)$$

where η is a tuning parameter that punishes a tree for having a large number of terminal nodes m^* .

Performing this *cost-complexity pruning* makes it possible to determine the best tree for a given size, and then we can perform CV on these trees (Faraway, 2006). Then the number of trees that needs to be considered in this CV is dramatically reduced by first applying the cost-complexity function, see Equation 5.9. Figure 5.4 illustrates a CV plot to find the optimal tree size for a classification tree, which is the tree type used in this thesis, and is explained next.

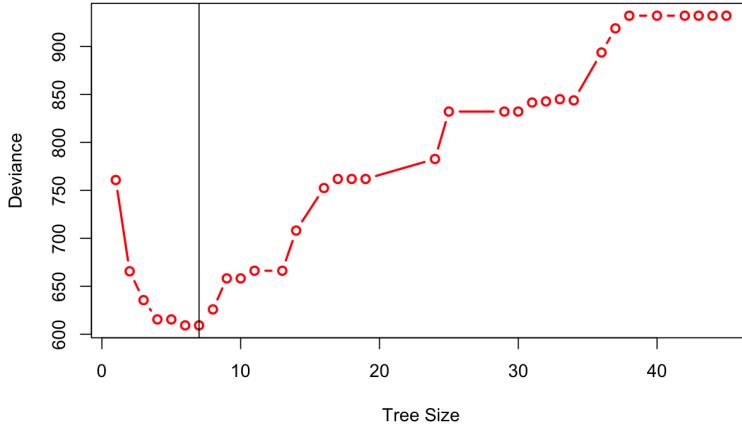


Figure 5.4: Example of $K = 5$ fold cross-validation to find the optimal tree size by inspecting which tree size minimizes the fitting criterion, which in this case is deviance.

5.3.2 Classification trees

Classification trees work in a similar manner to regression trees. In classification problems the response is no longer a numerical value, but a factor (*i.e.*, categorical or binary). In regression we make splits that minimize the RSS (Equation 5.8), but for classification we make the splits based on gathering the observations in a cluster such that the observations within a given split is of the same type, see Figure 5.2.

The general idea is using recursive binary splitting by starting with a *root node* and then splitting the feature space into finer and finer regions, hence creating the branches of the tree (Rokach and Maimon, 2010). Then the final decision nodes will split the observations into one *class* (*i.e.*, 1/Yes/True), or to another class (*i.e.*, 0/No/False) for binary problems. These are called *terminal nodes* or *leaf nodes*. We want our tree to perform this splitting in a good manner, and this can be checked by measuring the *purity* of a node (Faraway, 2006).

As mentioned we have R_j , $j \in [1, m^*]$ different partitions of the feature space, where each R_j represents a terminal node with N_j observations. For this given m^* one can measure the quantity of a given class observed at the j^{th} node. Then we can define

$$\hat{p}_{jk} = \frac{1}{N_j} \sum_{x_i \in R_j} I(y_i = k), \quad (5.10)$$

where the sum is over the different observations in the partition R_j , k is the class and $I(y_i = k)$ is the indicator function. This purity can be quantified using different measures; the *misclassification error*, *cross-entropy/deviance*, or *Gini index* (Hastie et al., 2009).

1. *Misclassification error*: $D_j = 1 - \hat{p}_{jk}$
2. *Cross-entropy/Deviance*: $D_j = -\sum_k \hat{p}_{jk} \log(\hat{p}_{jk})$
3. *Gini index*: $D_j = -\sum_k \hat{p}_{jk}(1 - \hat{p}_{jk})$

D_j is the measure of purity at a given node j , hence for the whole tree it is $\sum_j D_j$. Although the measures above work differently, they all obtain the minimum value when a node only contains observations of a single *class*. Then the node classifies the observations correctly, hence it can be looked upon as pure (James et al., 2021). The cross-entropy/deviance is the purity measure used throughout this thesis.

5.4 Bagging/bootstrap aggregation

It is known that decision trees have a disadvantage in that they often produce high *variance*. An example is if we create two different training sets of the same data, the two resulting trees can be very different. *Bagging* or *bootstrap aggregation* is a solution to this problem, and is a way of reducing this variance.

Bootstrapping is a method generally used on data to calculate the standard deviation of a quantity, e.g., the empirical mean. However, this approach can be implemented in tree-making. Then we randomly draw values from our training set *with replacement* to simulate the data-gathering process without actually gathering more data. If done repeatedly on our training data we get several *bootstrapped training sets*. Then we can grow several classification trees and *aggregate* them to make a more accurate prediction, see Figure 5.5. Despite this advantage, this comes at the cost of the interpretability of ordinary classification trees (Breiman, 1996).

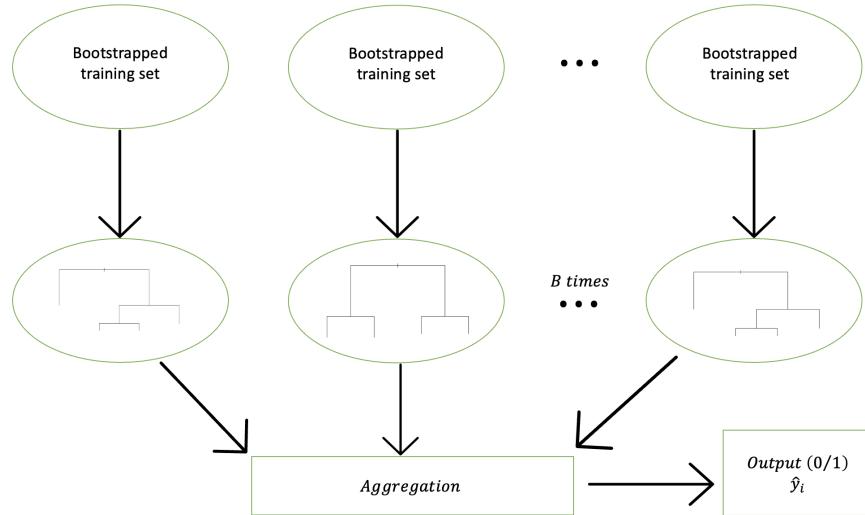


Figure 5.5: Illustration of created B trees on separate bootstrapped trees, and then taking aggregating them together to make a prediction.

To get an intuition on how bagging reduces variance, let us look at the case when B observations are independent and identically distributed (i.i.d) from a random variable X , and all these B 's have equal variance σ^2 . Then

$$\bar{X} = \frac{1}{B} \sum_{b=1}^B X_b , \quad (5.11)$$

and the variance of \bar{X} is

$$Var(\bar{X}) = Var\left(\frac{1}{B} \sum_{b=1}^B X_b\right) = \left(\frac{1}{B}\right)^2 \cdot \sum_{b=1}^B Var(X_b) = \frac{\sigma^2}{B} . \quad (5.12)$$

This is when we have i.i.d observations, then the variance is reduced by a factor of $\frac{1}{B}$. Then for regression problems, we get

$$\hat{f}_{\text{bagging}}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}_b(x) , \quad (5.13)$$

but for classification problems we use the *majority vote* instead of the average, see Algorithm 1. The majority vote is when the classifier is handed its test data, and the final prediction is the most common occurring class among all the trees $\hat{f}_b(x)$, $b = 1, \dots, B$ (James, 1998).

Algorithm 1 Bagging algorithm for classification trees

- 1: Split dataset \mathbf{X} into training $\mathbf{X}^{\text{train}}$ and test set \mathbf{X}^{test} .
 - 2: Bootstrap $\mathbf{X}^{\text{train}}$ B times $\implies \mathbf{X}_1^{\text{train}}, \dots, \mathbf{X}_B^{\text{train}}$.
 - 3: Grow classification trees for $\mathbf{X}_1^{\text{train}}, \dots, \mathbf{X}_B^{\text{train}} \implies \hat{f}_1(x), \dots, \hat{f}_B(x)$. $\triangleright m = p$ (Table 5.1)
 - 4: Aggregate the trees to T_{bagg}^B .
 - 5: Output the *majority vote* as the new prediction.
-

When the trees are made, all features $m = p$ are considered in each split, see Algorithm 1 (James, 1998). Hence the trees can look quite similar. This can cause the algorithm to get trapped in a local maximum, and is a weakness in this algorithm. This results in not exploring the model space optimally (James et al., 2021).

Bagging uses the concept of bootstrapping, and it creates different trees for different bootstrapped training sets from the available data. Each tree $\hat{f}_b(x)$, $b = 1, \dots, B$ uses approximately $\frac{2}{3}$ of the available data. This is due to the nature of the bootstrap (James et al., 2013). The result of this is that the remaining $\frac{1}{3}$ of the data is not used. This subset of the original observations is called the *out-of-bag* observations. Hence bagging yields its own test set, and the corresponding response to the out-of-bag observations can be used to calculate, *e.g.*, test error of the bagged tree. This is discussed more in Chapter 5.5. It is worth noting that bagging a good or decent classifier, like a classification tree, will make it better. However, bagging a very bad classifier can actually worsen the predictive performance (Hastie et al., 2009).

5.5 Random Forest

Random forest is another method to increase the accuracy of tree-based classifiers (Ho, 1995). Following the notation from Chapter 5.4, if we have X_b , $b = 1, \dots, B$ i.i.d random variables with variance σ^2 . Then the average of these X_b variables will have a variance equal to $\frac{\sigma^2}{B}$ (Hastie et al., 2009). However, often the variables are not independent, and then it gets more complicated. Then the variables are identically distributed (i.d. and not i.i.d.) with a pairwise positive correlation ρ , hence the variance of \bar{X} is

$$\begin{aligned} \text{Var}(\bar{X}) &= \text{Var}\left(\frac{1}{B} \sum_{b=1}^B X_b\right) \\ &= \frac{1}{B^2} \left(\sum_{b=1}^B \text{Var}(X_b) + \sum_{b \neq z} \text{Cov}(X_b, X_z) \right) \\ &= \frac{\sigma^2}{B} + \frac{B-1}{B} \sigma^2 \rho \\ &= \rho \cdot \sigma^2 + \frac{1-\rho}{B} \cdot \sigma^2. \end{aligned} \tag{5.14}$$

Equation (5.14) and Equation (5.12) show the core difference between *bagging* and *random forests*. Here, bagging can perform well when the training data produces high variance and low bias. Although looking at Equation 5.14, we see that the second term

$$\lim_{B \rightarrow \infty} \frac{1-\rho}{B} \cdot \sigma^2 = 0, \tag{5.15}$$

but bagging does not do anything with the first term $\rho \cdot \sigma^2$, and the trees in bagging are therefore often highly correlated. Hence the goal of random forests is to produce several trees that are less correlated, and thereby decreasing the first term without significantly increasing the variance. Random forests have shown to have better predictive performance than bagging, especially in positive correlation-problems (Hastie et al., 2009, Chapter 15).

Bagging uses all features ($m = p$) before making a split in the bootstrapped tree, which leads to many fairly similar trees, but random forest picks $m \leq p$ features before each split, see Algorithm 2. These m features are chosen at random, and we make the best split among these m features, according to the splitting criterion (e.g., deviance for classification problems). This decreases the positive correlation ρ , hence decreasing the expression in Equation 5.14 (Hastie et al., 2009). The consequence of this is that random forests often explore the model space more thoroughly than bagging (James et al., 2021).

Algorithm 2 Random forest algorithm for classification trees

- 1: Split dataset \mathbf{X} into training $\mathbf{X}^{\text{train}}$ and test set \mathbf{X}^{test} .
 - 2: Bootstrap $\mathbf{X}^{\text{train}}$ B times $\Rightarrow \mathbf{X}_1^{\text{train}}, \dots, \mathbf{X}_B^{\text{train}}$.
 - 3: Grow classification trees for $\mathbf{X}_1^{\text{train}}, \dots, \mathbf{X}_B^{\text{train}} \Rightarrow \hat{f}_1(x), \dots, \hat{f}_B(x)$. $\triangleright m = \sqrt{p}$ (Table 5.1)
 - 4: Aggregate the trees to T_{bag}^B .
 - 5: Output the *majority vote* as the new prediction.
-

For classification problems the most used value for m is $m = \sqrt{p}$, and $m = p/3$ for regression problems. In general, these benchmark values perform well in random forest algorithms. But m can be thought of as a tuning parameter, implying different values of m can produce better results. This can be the case when trees are grown using a lot of features, then using benchmark values might be too low. Especially if the number of relevant features in making a good prediction is low. An example is if a tree is grown using 100 features, where only 8 features are shown to be of importance. Then choosing $m = \sqrt{100} = 10$, will mean that there is a high probability that the random forest algorithm will choose features that are not necessarily very important, which leads to poorer predictions (Hastie et al., 2009). Therefore, if a random forest is fitted with many features one should use m as a tuning parameter to obtain the best predictive model possible.

Random forests and bagging differ in the number of parameters m considered at each split when the trees are constructed, but they work in a similar manner. They both produce a *out-of-bag sample*, and this can be used to obtain an *OOB error estimate* (James et al., 2013). Each tree created in the bagging/RF algorithm gets tested on the data not contained in the bootstrapped training set, and then the OOB error estimate is the average error for each prediction calculated. This can be looked upon as a validation set, since the out-of-bag sample was not used in the creation of the tree and is therefore *unseen* test data. Then bagging or random forest actually tests itself through its training.

Both methods are claimed to have the advantage that the number of bootstrapped training sets B will not cause overfitting. B has to be chosen in a manner in which the test error is stable, sufficiently low and a reasonable number in terms of computation time (James et al., 2021). Breiman (1996), the author of *Bagging predictors*, wrote the following on the manner: “In our experiments, 50 bootstrap replicates was used for classification and 25 for regression. This does not mean that 50 or 25 were necessary or sufficient, but simply that they seemed reasonable. My sense of it is that fewer are required when y_i is numerical and more are required with an increasing number of classes.” (p. 135)

When the number of trees B goes toward infinity, we get

$$T_{RF}^B = \lim_{B \rightarrow \infty} \hat{f}_B(x) , \quad (5.16)$$

and this limit *can*, but it does not necessarily, overfit to the training data. The average (regression) or the majority vote (classification) of *too many* fully grown trees can result in a model that is too well fitted on its training data and causes additional variance (Hastie et al., 2009). Bearing this in mind, most often it is sufficient to choose B such that the test error has stabilized.

5.6 Boosting

Boosting is a statistical learning algorithm that can be applied to various methods. When applied to trees, it uses many decision trees (classification or regression) and combines them into a more robust and better predictive tool, a *committee* of trees. In this sense, boosting has a similar purpose to our previously discussed methods (bagging and random forests). Despite this similarity, boosting differs substantially from bagging and random forests (Hastie et al., 2009). The latter two grow their trees independently of each other on separate bootstrapped training data (Figure 5.5), but boosting follows a different manner where the trees are grown *sequentially*, see Equation 5.17. By this, we mean that each tree created uses information from the previously created tree (James et al., 2021). This is shown in Figure 5.6, where $\hat{f}_0(x)$ is an initial tree, d -index, and b -index are explained in the following paragraphs.

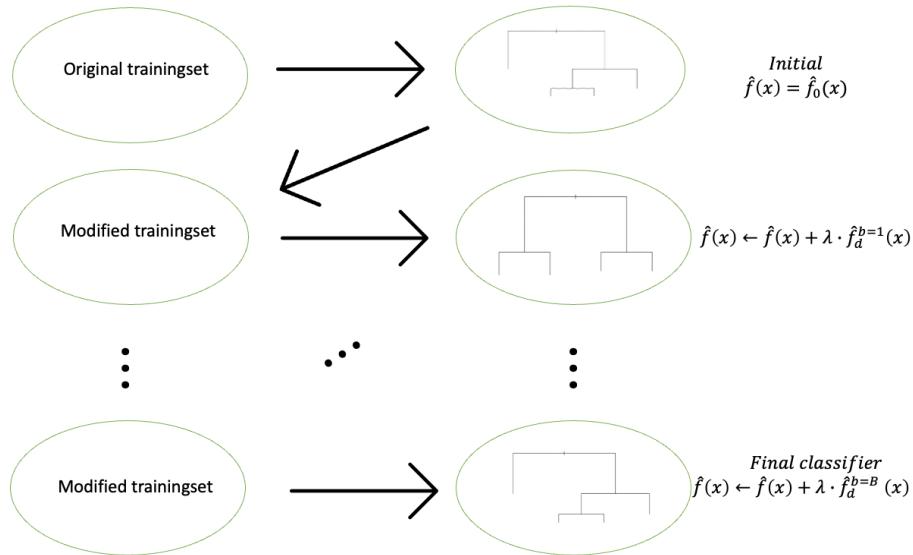


Figure 5.6: Illustration of boosted decision trees, where $\hat{f}(x)$ is updated sequentially.

In Chapter 5.3, we discussed the algorithm of decision trees, which can be said to be fitting the data hard, in the sense that we might overfit when including *too many* splits (James et al., 2021). Boosting is trying to avoid overfitting by learning *slowly*, this is done by a shrinkage parameter λ . Then the updating scheme can be written as

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \cdot \hat{f}_d^b(x), \quad b = 1, \dots, B. \quad (5.17)$$

Here $\hat{f}(x)$ has to be initialized by a $\hat{f}_0(x)$, and this is chosen according to the loss function used. For classification problems **bernoulli** is recommended, and then the initial value will be $\hat{f}_0(x) = \log\left(\frac{\sum w_i y_i}{\sum w_i (1-y_i)}\right)$, where w_i are calculated weights (Ridgeway, 2020). The other tuning parameters are B , λ , and d :

- B , Number of trees.
- λ , Shrinkage parameter.
- d , Interaction depth.

The main elements of the boosting approach for classification problems are summarized in Algorithm 3.

Algorithm 3 Boosting algorithm for (binary) classification trees

- 1: Split dataset \mathbf{X} into training $\mathbf{X}^{\text{train}}$ and test set \mathbf{X}^{test} .
 - 2: Initialize $\hat{f}(x) = \hat{f}_0(x)$ according to which **distribution** is used. \triangleright `bernoulli` or `adaboost`.
 - 3: **for** $b = 1, \dots, B$ **do** :
 - 4: Compute $\hat{f}_d^b(x)$ using a subset of $\mathbf{X}^{\text{train}}$.
 - 5: Update sequentially: $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \cdot \hat{f}_d^b(x)$. \triangleright For notation, see Table 5.1.
 - 6: **end for**
 - 7: Final boosted model is $\hat{f}(x) = \sum_{b=1}^B \lambda \hat{f}_d^b(x)$.
 - 8: Make predictions using a selected threshold. \triangleright Turning probabilities into 0 or 1.
-

Algorithm 3 can be implemented in R using the package `gbm` as follows:

```
model <- gbm(response ~ features, data = ..., distribution = 'bernoulli',
n.trees = B, interaction.depth = d, shrinkage = λ, bag.fraction = ...).
```

In more detail, this `gbm` function computes the negative gradient, which for `distribution = 'bernoulli'` is

$$z_i = y_i - \frac{1}{1 + \exp(-f(\mathbf{x}_i))}. \quad (5.18)$$

This z_i is used as the working response to fit the tree $\hat{f}_d^b(\mathbf{x}_i)$ by computing an expectation $E(z_i | y_i, \hat{f}(\mathbf{x}_i) | \mathbf{x}_i)$, where \mathbf{x}_i is the randomly chosen subset of $\mathbf{X}^{\text{train}}$ determined by the subsampling rate (`bag.fraction`). This was inspired by *Stochastic gradient boosting*, first proposed by Friedman (2002), which indicates how much of the initial training data is sampled randomly (*without replacement*) at each iteration step. This is implemented to increase the overall robustness of the method. Then the `gbm`-function computes the optimal terminal nodes and the tree $\hat{f}(x)$ gets sequentially updated for $b = 1, \dots, B$ as in Algorithm 3 (Ridgeway, 2020). Additionally, it is worth noting that when `distribution = 'bernoulli'`, the deviance is twice negative (weighted) log-likelihood

$$-2 \frac{1}{\sum w_i} \sum w_i (y_i f(\mathbf{x}_i) - \log(1 + \exp(f(\mathbf{x}_i)))) . \quad (5.19)$$

Tuning of boosting

The number of trees is an important tuning parameter of boosting methods because too large a B value can cause overfitting, and a too small value will lead to poor predictions. This is due to the *slow learning* analogy of boosting. λ is a scalar that controls how much information is added from the b^{th} tree to the predictive tree \hat{f} . In this sense, it tunes the rate at which the boosting algorithm learns. For tuning, $\lambda = 0.01$ or $\lambda = 0.1$ are frequently used values. The parameter d controls the number of splits made in each tree created. This means it gives a maximum of how many variable interactions the model considers. These tuning parameters have to be chosen wisely (tuned) by the user to acquire a good performance on the test set, while additionally avoiding overfitting (Hastie et al., 2009).

5.7 Deep learning and Neural networks

Neural networks have received a lot of attention in the field of deep learning. In fact, neural networks can be thought of as non-linear statistical models, which usually do a good job in classification problems due to their ability to *learn* patterns in the data (Faris et al., 2016). A *Multilayer Perceptron (MLP)* is a *feedforward neural network* that consists of three parts.

- An input layer.
- One or several hidden layer(s).
- An output layer.

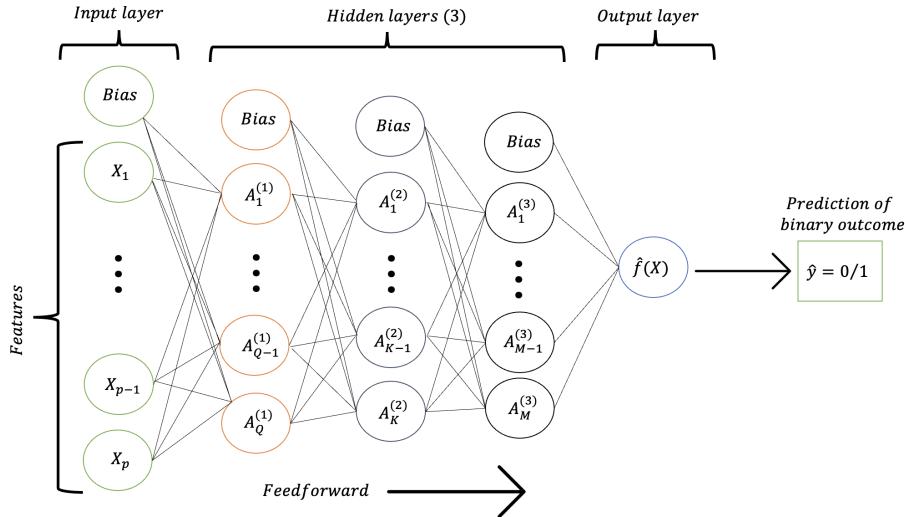


Figure 5.7: Illustration of Multilayered Perceptron (MLP) with 3 hidden layers.

These three parts are illustrated in Figure 5.7. The input layer consists of features \mathbf{X}_i , $i = 1, \dots, n$ to be used in the MLP. But for the sake of explanation, we look at the one-dimensional vector of features X_1, X_2, \dots, X_p , which can be called *neurons* in the setting of neural networks. The features are *fed* into the hidden layer(s). This first hidden layer consists of Q *hidden units*, and each of these units computes the activation for the q unit. Let us call these $A_q^{(1)}$ (James et al., 2021). This activation for the first hidden layer is calculated as

$$A_q^{(1)} = h_q(X) = g\left(w_{q0} + \sum_{r=1}^p w_{qr} X_r\right), \quad (5.20)$$

where $g(\cdot)$ is an activation function, see Algorithm 4. This is a non-linear function chosen by the user and has the purpose of capturing non-linear connections within the neural network. These are also called *nonlinearities*. The following equations are examples of activation functions

$$\text{Sigmoid: } \phi(z) = \frac{1}{1 + \exp\{-z\}}, \quad (5.21)$$

$$\text{ReLU: } \phi(z) = \max(0, z), \quad (5.22)$$

$$\text{Tanh: } \phi(z) = \frac{\exp\{z\} - \exp\{-z\}}{\exp\{z\} + \exp\{-z\}}. \quad (5.23)$$

The w_{qp} values are called *weights* and are parameters specified for each input X_p to the hidden units within the hidden layer, and these are estimated from the training set. Then the activation of the hidden unit $A_q^{(1)}$ is computed by applying an activation function to this weighted sum as in Equation 5.20 (James et al., 2021). Hence, the weights transform the inputs $(X_1, \dots, X_{p-1}, X_p)$

<i>Training set</i> $\in \mathbb{R}^{n \times p}$				$\mathbf{W} \in \mathbb{R}^{q \times p}$			
$x_{1,1}$	$x_{1,2}$...	$x_{1,p}$	$w_{1,1}$	$w_{1,2}$...	$w_{1,p}$
$x_{2,1}$	$x_{2,2}$...	$x_{2,p}$	$w_{2,1}$	$w_{2,2}$...	$w_{2,p}$
...
$x_{n,1}$	$x_{n,2}$...	$x_{n,p}$	$w_{q,1}$	$w_{q,2}$...	$w_{q,p}$

Figure 5.8: Example of how training data and weight matrix can be denoted. Where the inputs (training data) and the weights in \mathbf{W} are used to find a linear combination to be used in a neural network.

by linear combinations of the two matrices in Figure 5.8. The $A_q^{(1)}$ are then *fed* forward to the output layer, when leaving the hidden layer(s). Then the output can be written as

$$f(X) = \beta_0 + \sum_{q=1}^Q \beta_q A_q^{(1)}, \quad (5.24)$$

when there is one hidden layer, where the β s are parameters which are calculated through training of the network. For MLP we can add several hidden layers, with different amounts of hidden units. The number of layers depends on the problem at hand, and is for the user to tune.

For the convenience of this thesis, lets look at a MLP that consists of three layers. Lets call the first hidden layer $A_q^{(1)}$, see Equation 5.20. Then a second hidden layer $A_k^{(2)}$ would be

$$A_k^{(2)} = h_k^{(2)}(X) = g \left(w_{k0} + \sum_{q=1}^Q w_{kq}^{(2)} A_q^{(1)} \right), \quad (5.25)$$

where this second hidden layer $A_k^{(2)}$ uses the first hidden layer activations $A_q^{(1)}$, $q = 1, \dots, Q$ to calculate K new activations. This is repeated in the third hidden layer, and is of length M

$$A_m^{(3)} = h_m^{(3)}(X) = g \left(w_{m0} + \sum_{k=1}^K w_{mk}^{(3)} A_k^{(2)} \right). \quad (5.26)$$

Typical activation functions for the different hidden layers are Sigmoid (Equation 5.21), ReLU (Equation 5.22) and TanH (Equation 5.23). For binary classification, as in Algorithm 4, the Sigmoid-function is often used as the activation function in the output layer. This is because this function squishes the outcome (*e.g.*, Equation 5.24) to a number between 0 and 1. Then we can implement a threshold, *e.g.*, 0.5 to get the binary outcome. If the predicted number is above 0.5 we predict the outcome as 1, and vice versa for 0. The main workflow is summarized in Algorithm 4.

Algorithm 4 Multilayer Perceptron for binary classification (main steps)

- 1: Compute a weight-matrix \mathbf{W} from the training set.
 - 2: Calculate the weighted sum: $w_{q0} + \sum w_{qp} X_p$ for the input layer. $\triangleright w_{q0}$ is the bias term.
 - 3: Apply an activation function $g(\cdot)$ to this weighted sum.
 - 4: Repeat 1 to 3 for desired number of hidden layers. \triangleright Figure 5.7 has 3 layers.
 - 5: Apply Sigmoid as activation function in the output layer.
 - 6: Implement a threshold to transform the \mathbb{R} outputs into 0s and 1s.
-

As with the previously discussed methods, the MLP follows the same procedure regarding training and test data. Most of the data is used for training the neural network, and the remaining is held

out to be used for testing. Then the neural network is tested on unseen data, where the user has to tune different parts of the network, to get the best set of *weights* that minimizes the classification error (Faris et al., 2016).

5.8 Measurement of predictive performance

So far in this chapter, we have focused on trees, bagging, random forests, boosting, and neural networks. When these are trained on a training set, they need to be tested on an unseen test set. How well they perform in predicting unseen test data can be measured in different ways. In binary classification problems, as the focus of this thesis, the *confusion matrix* is a good tool. The confusion matrix is a 2×2 matrix with the predictions on the vertical part, and the correct/true classification on the horizontal part. We have chosen to call the latter 'reference', see Figure 5.9.

		Reference	
		0	1
Prediction	0	True Negative (TN)	False Negative (FN)
	1	False Positive (FP)	True Positive (TP)

Figure 5.9: Illustration of a confusion matrix for binary classification problems.

In this thesis *negative* is the 0 value, and is obtained when there is no PWL on a given day. While 1 is *positive*, and corresponds to PWL being forecasted on a given day, see Equation 5.5. *True negative (TN)* is when our algorithm or method predicts 0 and the correct response is 0, and vice versa for *true positive (TP)*. *False positive (FP)* is when our statistical algorithm predicts something as true when it is actually negative, and vice versa for *false negative (FN)*.

Accuracy, *specificity*, and *sensitivity* are measurements based on this matrix and are defined as follows:

$$\begin{aligned} \text{Accuracy} &= \frac{TN + TP}{TN + TP + FP + FN}, \\ \text{Specificity} &= \frac{TN}{TN + FP}, \\ \text{Sensitivity} &= \frac{TP}{TP + FN}. \end{aligned} \tag{5.27}$$

A good predictive model will have high accuracy, specificity and sensitivity at once. If the dataset is imbalanced, in the sense that there are significantly more 0s than 1s, or vice versa, the *balanced accuracy* can be used. Balanced accuracy is the mean of the sensitivity and specificity,

$$\text{Balanced accuracy} = \frac{\text{Specificity} + \text{Sensitivity}}{2}. \tag{5.28}$$

Overfitting - in the eyes of the methods described

Overfitting is said to happen when a given model is too well fitted to a particular dataset, hence performing extremely well on a given test set. The trade-off is that it performs poorly on another test set, hence it does not generalize well. Modern machine learning techniques have the advantage of finding difficult, complex relationships between features and a response. So far we have discussed several different machine and statistical learning methods. It is important to note when using bagging and random forest, which use interpretable classification trees and aggregate them together. This comes at the cost of interpretability because once we ensemble several trees together, they are really no longer trees. This goes for boosting (Chapter 5.6) as well. For these ensemble methods, we make our predictions using a "black box". By this, we mean that we are solely interested in the final prediction (*e.g.*, 0/1), hence the internal parameters within the algorithm are more or less uninterpretable. This can, often unintentionally, cause overfitting to the training data since these methods could potentially capture noise in the data (Hosseini et al., 2020).

Chapter 6

Results

As discussed in Chapter 3.3.1, the temperature gradient within the snowpack is the main factor in determining whether faceted grains and depth hoar come about or not. The weather data used in this thesis is the air mean temperature ($temp$ ($^{\circ}C$)) and the mean snow depth ($snow_{depth}$ (mm)) at different elevation intervals in the given region of interest, *i.e.*, Sunnmøre or Romsdal. Other models with additional features such as $precip$ (mm), $precip_{max}$ (mm), $new - snow$ (mm), and $wind$ (m/s) did not increase predictive performance, and some features led to worse results. Hence the primary interest of this thesis is whether air temperature and snow depth are good features to predict PWL(s) due to their indirect influence on the temperature gradient within the snowpack.

Structure of this chapter

This chapter will include four subsections, where we introduce three different models. The last subsection includes an extension of the time-dependent model. These are listed and explained below.

1. $M_{stationary}$ is a stationary temperature model. It uses air temperature (at different elevations) for the given day of interest as its features.
2. $M_{stationary*}$ is also a stationary model, but it uses air temperature and snow depth (at different elevations) for the given day of interest as its features.
3. $M_{time-dependent}$ considers the time aspect. Hence it uses air temperature and snow depth (at different elevations) for the given day of interest, and the five preceding days, as its features.
4. M_{MLP} uses the same features as $M_{time-dependent}$, and is an extension of the time-dependent model.

All machine learning methods were *region-specific*, meaning they were trained and tested on data from the same region, *i.e.*, Sunnmøre or Romsdal. The dataset consisted of five winter seasons from 2017/18 to 2021/22. To test the predictive performance and robustness of each approach, four different training sets were tried:

- Training set where 75% of the data was chosen randomly, and the test set was its complement. This was denoted as R .
- The first four seasons were used as training set, and the test set was the last season 2021/22. This was denoted as $4F$.
- The last four seasons were used as training set, and the test set was the first season 2017/18. This was denoted as $4L$.

-
- The three first seasons and the last season made up the training set, and the test set was 2019/20. This was denoted as 3/1.

The three first subsections ($M_{stationary}$, $M_{stationary^*}$ and $M_{time-dependent}$) focus on classification trees, pruning, bagging, random forests, and boosting, see Chapter 5. In the last subsection, we use a multilayer perceptron (M_{MLP}) to classify the binary response \hat{Y}_i on the time-dependent model. How these methods were implemented in R and Python are exemplified in Appendix A.2 and A.3. In each subsection, we will give a brief explanation of the different tables with respect to their different training sets and best results to clarify the findings of this thesis. For each machine learning approach the predictive measurements were accuracy, sensitivity and specificity, see Equation 5.27.

R and Python packages used

Classification trees: For creating classification trees we implemented the R package `trees` (<https://www.rdocumentation.org/packages/tree/versions/1.0-42/topics/tree>). This was used to grow the trees using binary recursive splitting with deviance as the splitting criterion as described in Chapter 5.3.

Bagging/Random forest: When applying bagging and random forest to classification trees the R library `randomForest` was used (<https://www.rdocumentation.org/packages/randomForest/versions/4.7-1.1/topics/randomForest>). Each tree was fully grown and we set $B = 500$ (number of trees), because of computation time, and the test error stabilized at approximately 500. The parameter `mtry` was equal to p (number of predictors) for bagging, but for random forest the default $mtry = \sqrt{p}$ was used.

Boosting: In this thesis, the boosting package `gbm` (Gradient Boosting Machine) in R was implemented (<https://www.rdocumentation.org/packages/gbm/versions/2.1.8.1>). For classification problems, `bernulli` or `adaboost` were recommended loss functions by Ridgeway (2020) (author of the R package), with particular recommendations for using `bernulli`. The parameter `bag.fraction` had default-value of 0.5. This means 50% of the training data is chosen (at random) to create the next tree in the sequential updating of the classifier. When the data size is small, a higher number could be considered (Hastie et al., 2009, p. 412). `bag.fraction` was therefore set to 0.8. This showed, by trial and error, to outperform the default value. The other tuning parameters, and the choice of their values, are discussed more thoroughly later in this chapter.

Multilayer perceptron: To create and train the feedforward neural network we used Python instead of R. For this task, we chose PyTorch due to its GPU acceleration for tensor computations and its applicability for deep learning problems (<https://pytorch.org/>). Additionally, libraries such as `pandas` and `scikit-learn` were used.

Additional information about tuning and the methods

There were a lot of different choices made when applying trees, bagging, random forests, boosting and neural networks. The following list gives additional information about the tuning parameters so that a fair comparison between the methods can be made.

- For reproducible results the `set.seed(563)` was used.
- The pruned tree is chosen by the tree size n that minimizes the 5-fold cross-validation error, see Equation 5.6.
- When Bagging and random forests (RF) are denoted with "Bagging*" or "RF*", the "*" indicate that it is an OOB estimate.
- Bagging includes a B which indicates how many trees have been created in the algorithm. B was chosen such that the test error stabilized.

-
- Boosting was tuned by trying all combinations of five different interaction depths ($d = \{1, 2, 3, 4, 5\}$), five different shrinkage parameters ($\lambda = \{0.1, 0.05, 0.01, 0.005, 0.001\}$) and five different number of trees ($B = \{1000, 2000, 3000, 4000, 5000\}$). The parameters were selected in a manner such that the balanced accuracy was maximized.
 - The first 30 days of 2017/18 season had missing snow depth data. Hence the models using snow depth omitted these 30 rows.
 - For the MLP, three activation functions were tried in the hidden layers (**ReLU**, **Sigmoid** and **TanH**), while **Sigmoid** was used for the output layer.
 - The loss function used in M_{MLP} was **cross-entropy**, and batch size was set to 32 after trial and error. Epoch and learning-rate were tuned using methods from PyTorch.
 - After trial and error, the threshold for determining if the M_{MLP} predicted 1 (PWL) was set to 0.5.

6.1 Stationary temperature model

Formation of PWL requires that the *temperature gradient* is larger than $10^{\circ}C\ m^{-1}$ over time, see Chapter 3.3.1. The stationary temperature model, call it $M_{stationary}$, is *stationary* in time. This means that the algorithms only use *features* for the given day i of interest. The features in $M_{stationary}$ is temperatures at the five different elevation intervals (of length 400 m).

The accuracy, sensitivity, and specificity of $M_{stationary}$ are summarized in Tables 6.1 and 6.2. Table 6.1 shows predictive measurements for R and $4F$ training sets. For R training set, boosted classification trees with $d = 2$, $B = 1000$, and $\lambda = 0.05$ gave the best results when leaving out the OOB estimates of bagging and random forests. Boosted classification trees for Sunnmøre gave an accuracy of 0.6754, specificity of 0.8313, and sensitivity of 0.3008, this is enhanced in bold in Table 6.1. For $4F$, when the methods were trained on the first four seasons, the random forest results are enhanced in bold in Table 6.1. This shows the difference in the predictive performance based on the OOB samples ("RF**"), or estimates from when the model is tested on the last 2021/22 season.

Table 6.2 shows measurements for $4L$ and $3/1$ training set, where neither of them produced any extraordinary results. For $4L$, the OOB estimates for bagging are enhanced in bold to show the high predictive performance versus the estimates calculated when tested upon the first season. For $3/1$, the RF ($m = 2$) measurements enhanced a model that yields very high specificity, but at a cost of low sensitivity (true positive rate).

6.2 Stationary temperature model + snow depth as an additional feature

$M_{stationary}$ is a huge simplification of the problem at hand, and adding $snowdepth$ as an additional feature seems reasonable. This was done due to the concept of the *temperature gradient* (Equation 3.1), the key factor in *kinetic transformation*. We call this model $M_{stationary*}$, and it consists of

$$2 \text{ (feature types)} \cdot 5 \text{ (elevation intervals)} = 10 \text{ (total features).}$$

In general, all methods in $M_{stationary*}$ performed better than the trees from $M_{stationary}$. For each training set, the methods with the highest balanced accuracy (Equation 5.28) is enhanced in bold. Here we chose to not enhance the OOB estimates because all training sets show that the OOB estimates are overall better than the test error estimates, as for $M_{stationary}$. In Table 6.3, the random training set resulted in an accuracy of 0.9224, specificity of 0.9740, and sensitivity of 0.8000. While the $4F$ training set gave accuracy of 0.7582, specificity of 0.6857, and sensitivity of 0.8571. However, as for $M_{stationary}$, the $4L$ training set produced better results than $4F$, see Table 6.4.

The predictive measurements for the four different training sets are summarized in Tables 6.3 and 6.4.

6.3 Time-dependent model using temperature and snow depth

To increase predictive performance (when compared with $M_{stationary}$ and $M_{stationary^*}$) an idea is to include features over time. In a more complex model, call it $M_{time-dependent}$, we use features that consider the aspect of time. This is due to the fact that PWLs form when the temperature gradient, within the snowpack, endures over time (often days). Then *kinetic transformation* is the dominating transformation in the snowpack, and could potentially capture existence/non-existence of PWLs in a better way than the preceding, simpler models.

In $M_{time-dependent}$ the features, $temp$ ($^{\circ}C$) and $snow_depth$ (mm), are given for five different elevation intervals and for day $i, i - 1, \dots, i - 5$. This means that $M_{time-dependent}$ consists of

$$2 \text{ (feature types)} \cdot 5 \text{ (elevation intervals)} \cdot 6 \text{ (different days)} = 60 \text{ (total features)}.$$

The random R training set resulted in the best predictive measurements, as with the two preceding models. Another similarity of $M_{time-dependent}$ with $M_{stationary}$ and $M_{stationary^*}$, is that the training set $4L$ performed better than $4F$ and $3/1$. For $M_{time-dependent}$ (Sunnmøre) using $4L$ as training set resulted in an accuracy of 0.7853, specificity of 0.7596 and sensitivity of 0.8219 when using boosted trees with $d = 4$, $B = 2000$, and $\lambda = 0.05$, see Table 6.6. And for Romsdal, the measurements were even higher. This is discussed more thoroughly in Chapter 7.

The accuracy, sensitivity, and specificity for all the different training sets are summarized in the following tables, see Tables 6.5 and 6.6.

6.4 Multilayer perceptron

The three previous models use more classic, well-known statistical approaches than this last approach. It is reasonable to suspect that a more complex approach might perform better with the same dataset as $M_{time-dependent}$ with 60 features. M_{MLP} is a classifier that extends the time-dependent model by implementing a multilayer perceptron (MLP) with three hidden layers.

The random R training set resulted in extremely good measurements, as $M_{time-dependent}$ and $M_{stationary^*}$. With accuracy, sensitivity, and specificity all being approximately 0.90. When compared to $M_{time-dependent}$, M_{MLP} performed much better with the $3/1$ training set, but a lot worse with the $4L$ training set. In general, the activation function in the hidden layers of M_{MLP} did not cause major differences in performance.

The predictive measurements for all four training sets are summarized in Tables 6.7 and 6.8.

Table 6.1: $M_{stationary}$ results for training sets R and $4F$, where the features are the temperature at five different elevation intervals. Here, * indicates that the estimates are calculated using OOB samples. The predictive measurements (in bold) showcase the best results or the difference between OOB-estimates versus the test set estimates.

Training:		R		Pruned tree (size = 3)		Bagging* ($B = 500$)		Bagging ($B = 500$)		RF* ($m = 2$)		RF ($m = 2$)		Boosting ($d = 2, B = 1000, \lambda = 0.05$)	
Romsdal	Accuracy	0.7018	0.7018	0.7013	0.6579	0.6999	0.6579	0.6754							
	Specificity	0.9062	0.9062	0.7593	0.8187	0.7580	0.8187	0.8313							
	Sensitivity	0.2206	0.2206	0.4237	0.2794	0.4188	0.2794	0.3088							
$4F$	Accuracy	0.7149	0.7149	0.7277	0.6579	0.7335	0.7193	0.7105							
	Specificity	0.9512	0.9152	0.7821	0.8110	0.7827	0.8841	0.8232							
	Sensitivity	0.1094	0.1094	0.4797	0.2656	0.4957	0.2969	0.3219							
Training:		Class. tree (size = 3)		Pruned tree (size = 3)		Bagging* ($B = 500$)		Bagging ($B = 500$)		RF* ($m = 2$)		RF ($m = 2$)		Boosting ($d = 4, B = 3000, \lambda = 0.05$)	
Romsdal	Accuracy	0.7348	0.7348	0.6904	0.7017	0.6863	0.7182	0.7017							
	Specificity	0.8857	0.8857	0.7530	0.8214	0.7483	0.8429	0.8143							
	Sensitivity	0.2195	0.2195	0.4581	0.2927	0.4467	0.2927	0.3171							
$Summ\o re$	Accuracy	0.7680	0.6022	0.7137	0.6961	0.7233	0.7182	0.6685							
	Specificity	0.9574	0.6809	0.7669	0.8440	0.7687	0.8652	0.7650							
	Sensitivity	0.1000	0.3250	0.4855	0.1750	0.5116	0.2000	0.3250							

Table 6.2: $M_{stationary}$ results for $4L$ and $3/1$, where the features are the temperature at five different elevation intervals. Here, * indicates that the estimates are calculated using OOB samples. The bold predictive measurements showcase the difference between OOB-estimates and test set estimates ($4L$), and a model with high specificity, and very low sensitivity ($3/1$).

Training: $4L$		Training: $3/1$			
Summøre	Class. tree (size = 3)	Pruned tree (size = 3)	Bagging* ($B = 500$)	Bagging ($B = 500$)	RF* ($m = 2$)
Accuracy	0.5470	0.5470	0.7562	0.5359	0.7671
Specificity	0.9327	0.9327	0.7971	0.8654	0.7988
Sensitivity	0.0260	0.0260	0.4949	0.0909	0.4607
Romsdal	Class. tree	Pruned tree (size = 3)	Bagging* ($B = 500$)	Bagging ($B = 500$)	RF* ($m = 2$)
			0.7890	0.5635	0.7959
Accuracy	0.5801	0.5801	0.7890	0.5635	0.7959
Specificity	0.9783	0.9783	0.8231	0.9348	0.8234
Sensitivity	0.1685	0.1685	0.5125	0.1798	0.5479
Training: $3/1$	Class. tree	Pruned tree (size = 3)	Bagging* ($B = 500$)	Bagging ($B = 500$)	RF* ($m = 2$)
			0.6989	0.5936	0.7141
Accuracy	0.5989	0.5989	0.6989	0.5936	0.6096
Specificity	0.9643	1.0000	0.7633	0.9107	0.7674
Sensitivity	0.0533	0.0000	0.3107	0.1200	0.3478
Romsdal	Class. tree	Pruned tree (size = 3)	Bagging* ($B = 500$)	Bagging ($B = 500$)	RF* ($m = 2$)
			0.7445	0.6471	0.7334
Accuracy	0.6898	0.6310	0.7445	0.6471	0.7334
Specificity	0.9831	1.0000	0.7886	0.8390	0.7822
Sensitivity	0.1884	0.0000	0.4526	0.3188	0.4105
					RF ($d = 5, B = 4000, \lambda = 0.01$)
					Boosting ($d = 5, B = 4000, \lambda = 0.1$)
					RF ($m = 2$)
					Boosting ($d = 5, B = 4000, \lambda = 0.005$)
					RF ($d = 5, B = 2000, \lambda = 0.005$)
					0.5824
					0.5635
					0.9565
					1.0000
					0.1461

Table 6.3: $M_{stationary^*}$ results for R and $4F$, where the features are the temperature and snow depth at five different elevation intervals. Here, * indicates that the estimates are calculated using OOB samples. The predictive measurements (in bold) showcase the best results for each training set. Additionally, note the differences between OOB-estimates and test set estimates for bagging and random forests.

Training:		<i>R</i>		Bagging*		RF*		RF		Boosting	
Summøre	Class. tree	Pruned tree (size = 10)	Bagging ($B = 500$)	Bagging ($B = 500$)	Bagging ($B = 500$)	RF*	($m = 3$)	($m = 3$)	($d = 2$)	($B = 3000$, $\lambda = 0.05$)	
Romsdal	Accuracy	0.8904	0.8816	0.8957	0.9224	0.9018	0.9178	0.9210			
	Specificity	0.9437	0.9750	0.9163	0.9740	0.9137	0.9805	0.9813			
	Sensitivity	0.7647	0.6618	0.8372	0.8000	0.8659	0.7692	0.7794			
4F	Accuracy	0.8684	0.8684	0.8997	0.9269	0.8943	0.9087	0.9254			
	Specificity	0.9207	0.9695	0.904	0.9810	0.9025	0.9873	0.9939			
	Sensitivity	0.7344	0.6094	0.8533	0.7869	0.8651	0.7049	0.7500			
Training:		Class. tree	Pruned tree (size = 9)	Bagging ($B = 500$)	Bagging ($B = 500$)	RF*	($m = 3$)	($m = 3$)	Boosting		
Romsdal	Accuracy	0.7253	0.8022	0.9198	0.7637	0.9199	0.7253	0.6243			
	Specificity	0.8857	0.9929	0.9376	0.8500	0.9394	0.8286	0.5929			
	Sensitivity	0.1905	0.1407	0.8762	0.4762	0.8725	0.3810	0.7317			
Summøre	Accuracy	0.6923	0.8287	0.9099	0.7637	0.9130	($m = 3$)				
	Specificity	0.7234	0.9007	0.9210	0.8156	0.9118	0.9078	0.8794			
	Sensitivity	0.5854	0.5854	0.8778	0.5854	0.9157	0.5810	0.6000			

Table 6.4: $M_{stationary*}$ results for $4L$ and $3/1$, where the features are the temperature and snow depth at five different elevation intervals. Here, * indicates that the estimates are calculated using OOB samples. The predictive measurements (in bold) showcase the best results for each training set.

Training: $4L$		Training: $3/1$					
Summøre	Class. tree (size = 10)	Pruned tree (size = 10)	Bagging* ($B = 500$)	Bagging ($B = 500$)	RF* ($m = 3$)	RF ($m = 3$)	Boosting ($d = 5, B = 4000, \lambda = 0.1$)
Romsdal	Accuracy	0.7582	0.7582	0.9314	0.7434	0.9287	0.7500
	Specificity	0.6857	0.6857	0.9466	0.5316	0.9386	0.7342
	Sensitivity	0.8571	0.8571	0.8802	0.9726	0.8931	0.7671
Romsdal	Accuracy	0.6648	0.6703	Bagging* ($B = 500$)	Bagging ($B = 500$)	RF* ($m = 3$)	RF ($d = 3, B = 3000, \lambda = 0.01$)
	Specificity	0.8710	0.9032	0.9150	0.8947	0.9108	0.6184
	Sensitivity	0.4494	0.4270	0.9237	0.9412	0.9071	0.9706
Summøre	Accuracy	0.7637	0.7692	0.9299	0.6374	0.9342	0.6264
	Specificity	0.9907	1.000	0.9480	0.8224	0.9426	0.9813
	Sensitivity	0.4400	0.4400	0.8706	0.3733	0.9057	0.1200
Romsdal	Accuracy	0.5604	0.5604	Bagging* ($B = 500$)	Bagging ($B = 500$)	RF* ($m = 3$)	RF ($d = 4, B = 2000, \lambda = 0.001$)
	Specificity	0.7788	0.7788	0.9417	0.7876	0.9314	0.9027
	Sensitivity	0.2029	0.2029	0.8910	0.2029	0.9172	0.4058

Table 6.5: $M_{time-dependent}$ results for R and $4F$, and uses 60 features, i.e., temperature and snow depth at five different elevation intervals for day i , $i = 1, \dots, i - 5$. Here, * indicates that the estimates are calculated using OOB samples. The predictive measurements (in bold) showcase the best results for each training set, when excluding the OOB-estimates.

Training: Sunmore	<i>R</i>	Class. tree (size = 15)	Pruned tree (size = 15)	Bagging* ($B = 500$)	Bagging ($B = 500$)	RF* ($m = 8$)	RF ($m = 8$)	Boosting ($d = 5, B = 2000, \lambda = 0.1$)
Accuracy	0.8767	0.8855	0.9087	0.9132	0.9148	0.9224	0.9559	
Specificity	0.9529	0.9259	0.9155	0.9675	0.9195	0.9870	0.9630	
Sensitivity	0.7538	0.7846	0.8875	0.7846	0.9000	0.7692	0.9385	
Romsdal	Class. tree (size = 5)	Pruned tree (size = 5)	Bagging* ($B = 500$)	Bagging ($B = 500$)	RF*	RF	Boosting	
Accuracy	0.8590	0.8678	0.8950	0.9224	($m = 8$)	($m = 8$)		
Specificity	0.9112	0.9586	0.9002	0.9752	0.9072	0.9132	($d = 3, B = 4000, \lambda = 0.05$)	
Sensitivity	0.7069	0.6034	0.8767	0.7759	0.9064	0.9629	0.9645	
Training: 4F	Class. tree (size = 11)	Pruned tree (size = 11)	Bagging* ($B = 500$)	Bagging ($B = 500$)	RF*	RF	Boosting	
Accuracy	0.6667	0.7740	0.9485	0.4124	0.9496	0.4696	0.7182	($d = 1, B = 2000, \lambda = 0.005$)
Specificity	0.7786	0.9000	0.9593	0.4071	0.9556	0.4857	0.7929	
Sensitivity	0.2432	0.2973	0.9231	0.4324	0.9347	0.4146	0.4634	
Romsdal	Class. tree (size = 5)	Pruned tree (size = 5)	Bagging* ($B = 500$)	Bagging ($B = 500$)	RF*	RF	Boosting	
Accuracy	0.7006	0.8644	0.9227	0.7853	($m = 8$)	($m = 8$)		
Specificity	0.8227	0.9291	0.9237	0.8298	0.9098	0.8511	0.8865	($d = 5, B = 4000, \lambda = 0.1$)
Sensitivity	0.2222	0.6111	0.9200	0.6216	0.9141	0.6000	0.5500	

Table 6.6: $M_{time-dependent}$ results for $4L$ and $3/1$, and uses 60 features, *i.e.*, temperature and snow depth at five different elevation intervals for day i , $i = 1, \dots, i - 5$. Here, $*$ indicates that the estimates are calculated using OOB samples. The predictive measurements (in bold) showcase the best results for each training set.

Training: $4L$		Training: $3/1$							
Summøre	Class. tree (size = 9)	Pruned tree (size = 9)	Bagging* ($B = 500$)	Bagging ($B = 500$)	RF* ($m = 8$)	RF ($m = 8$)	Boosting ($d = 4, B = 2000, \lambda = 0.05$)		
Accuracy	0.6780	0.6780	0.9396	0.7483	0.9465	0.6054	0.7853		
Specificity	0.9355	0.9462	0.9425	0.6173	0.9430	0.7467	0.7596		
Sensitivity	0.3929	0.3810	0.9290	0.9091	0.9600	0.4583	0.8219		
Romsdal	Class. tree (size = 6)	Pruned tree (size = 6)	Bagging* ($B = 500$)	Bagging ($B = 500$)	RF* ($m = 8$)	RF ($m = 8$)	Boosting ($d = 2, B = 4000, \lambda = 0.1$)		
Accuracy	0.6780	0.4972	0.9053	0.7483	0.9150	0.4898	0.8927		
Specificity	0.9355	0.9462	0.9000	0.9091	0.9088	0.9697	0.9462		
Sensitivity	0.3929	0.0000	0.9394	0.6173	0.9519	0.0988	0.8333		
Summøre	Class. tree (size = 9)	Pruned tree (size = 9)	Bagging* ($B = 500$)	Bagging ($B = 500$)	RF* ($m = 8$)	RF ($m = 8$)	Boosting ($d = 4, B = 4000, \lambda = 0.01$)		
Accuracy	0.6538	0.6484	0.9452	0.6868	0.9539	0.6648	0.7327		
Specificity	0.7850	0.7580	0.9599	0.9813	0.9621	1.0000	1.0000		
Sensitivity	0.4667	0.4533	0.9000	0.2667	0.9277	0.1867	0.4000		
Romsdal	Class. tree (size = 13)	Pruned tree (size = 13)	Bagging* ($B = 500$)	Bagging ($B = 500$)	RF* ($m = 8$)	RF ($m = 8$)	Boosting ($d = 5, B = 4000, \lambda = 0.001$)		
Accuracy	0.6758	0.8242	0.9280	0.7912	0.9279	0.7857	0.7418		
Specificity	0.8053	0.9823	0.9343	0.9558	0.9311	0.9569	1.0000		
Sensitivity	0.4638	0.5652	0.9041	0.5217	0.9155	0.5247	0.3288		

Table 6.7: M_{MLP} results for training sets R and $4F$, using 60 features as the time-dependent model. Here listed with 3 different activation functions in the hidden layers.
The predictive measurements (in bold) showcase the best results for each training set.

Training:		R			
Summøre	Act. function in hidden layers:	ReLU	Sigmoid	TanH	
Accuracy	ReLU	0.9041	0.9041	0.8721	
	Sigmoid	0.8968	0.8968	0.8671	
	TanH	0.9219	0.9219	0.8852	
Specificity	ReLU	0.9219	0.9219	0.8852	
	Sigmoid	0.8767	0.8858	0.8676	
	TanH	0.8580	0.8642	0.8519	
Sensitivity	ReLU	0.9298	0.9298	0.9123	
	Sigmoid	0.9298	0.9298	0.9123	
	TanH	0.9298	0.9298	0.9123	
Training:		$4F$			
Summøre	Act. function in hidden layers:	ReLU	Sigmoid	TanH	
Accuracy	ReLU	0.6477	0.6761	0.6477	
	Sigmoid	0.7954	0.8029	0.7910	
	TanH	0.2045	0.2308	0.1905	
Specificity	ReLU	0.3465	0.3864	0.4375	
	Sigmoid	0.7407	0.7089	0.8387	
	TanH	0.1721	0.1237	0.2193	
Sensitivity	ReLU	0.6477	0.6761	0.6477	
	Sigmoid	0.7954	0.8029	0.7910	
	TanH	0.2045	0.2308	0.1905	
Romsdal	Accuracy	0.3465	0.3864	0.4375	
	Specificity	0.7407	0.7089	0.8387	
	Sensitivity	0.1721	0.1237	0.2193	

Table 6.8: M_{MLP} results for training sets $4L$ and $3/1$, using 60 features as the time-dependent model. Here listed with 3 different activation functions in the hidden layers.
The predictive measurements (in bold) showcase the best results for each training set.

Training:		<i>4L</i>			<i>3/1</i>		
Summøre	Act. function in hidden layers:	ReLU	Sigmoid	TanH	ReLU	Sigmoid	TanH
Accuracy		0.4514	0.4743	0.5486			
Specificity		0.5327	0.5462	0.5968			
Sensitivity		0.3235	0.3214	0.4314			
Romsdal	Act. function in hidden layers:	ReLU	Sigmoid	TanH			
Accuracy		0.5029	0.5829	0.5486			
Specificity		0.5256	0.5745	0.5573			
Sensitivity		0.4211	0.6176	0.5227			
Training:							
Summøre	Act. function in hidden layers:	ReLU	Sigmoid	TanH			
Accuracy		0.6831	0.6831	0.7198			
Specificity		0.6299	0.6320	0.6560			
Sensitivity		0.8036	0.7931	0.8448			
Romsdal	Act. function in hidden layers:	ReLU	Sigmoid	TanH			
Accuracy		0.6667	0.6885	0.6940			
Specificity		0.6400	0.6667	0.6613			
Sensitivity		0.7241	0.7272	0.7627			

6.5 Summary of key machine learning results

The preceding tables (Tables 6.1 to 6.8) show all predictive measurements for the statistical and machine learning methods used. Since all methods were applied to different training sets, and we used three different versions of the dataset (5, 10, and 60 features), it resulted in a lot of numbers. Some predictive measurements were presented in bold to either showcase the best results for a specific training set, or to enhance the findings to be used in the discussion (Chapter 7).

The following list tries to summarize the key findings in the results, and will be discussed thoroughly in the following chapter.

- $M_{stationary}$ is the simplest model with only five features, namely the temperature at different elevation intervals. This leads to overall poor predictive measurements, where the boosted classification trees generally produced the best results.
- In general, $M_{time-dependent}$ (60 features) did better than $M_{stationary*}$ (10 features), and $M_{stationary*}$ did better than $M_{stationary}$ (5 features). Therefore it seems like a good decision using a model that takes time into consideration.
- The extension of the time-dependent model, M_{MLP} , performed (in general) worse than bagging, RF and boosting from $M_{time-dependent}$. But the neural network performed a lot better for the training set 3/1 compared to the other approaches, see Tables 6.5 to 6.8.
- The three models all included bagging and random forests. Tables 6.1 to 6.6 show that the OOB estimates (Bagging* or RF*) were consistently higher than the test set estimate (Bagging or RF).
- For all statistical and machine learning methods, the random training set (R) performed the best.
- When excluding the random training set, the $4L$ training set resulted in the best predictive measurements for both $M_{stationary*}$ and $M_{time-dependent}$.
- In general, there is a dispersion in the predictive measurements between different training sets. This indicates that the methods presented here suffer from non-robustness.

Chapter 7

Discussion

In the results (Chapter 6), we used four different training sets (R , $4F$, $4L$, and $3/1$), and these will be discussed with respect to overfitting and what we chose to call *seasonal dependencies*. Then we discuss the four models concerning their predictive performance. At last, we will discuss what these statistical and machine learning methods might bring to aid the avalanche forecast in maritime regions by combining the statistical results with the snow science theory discussed in Chapter 3.

7.1 Different training sets and overfitting

Figure 7.1 and 7.2 show how the training sets R (75% of the dataset chosen at random) and $4F$ (first four seasons) were sampled for Sunnmøre. This is analogous for Romsdal and the other training sets, $4L$ and $3/1$.

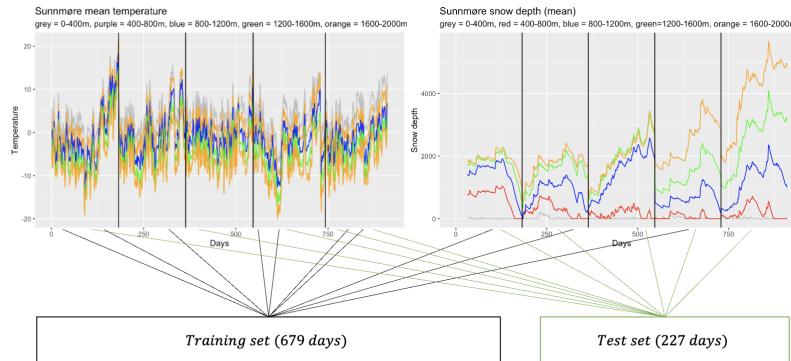


Figure 7.1: Illustration of how the R (random) training set was sampled for Sunnmøre, using temperature (temp ($^{\circ}\text{C}$)) and snow depth (snowdepth (mm))) as features. The training and test set included observations from all five seasons.

Every test set created is unseen in the sense that these exact data points were not used in training. Nevertheless, the test set used for the random training set (R) is chosen among all five seasons. Hence it captures what we have chosen to call seasonal dependencies. Meaning that each season has its own "characteristics". As we can see in Figures 7.1 and 7.2, the snow depth is different for each season, and temperatures fluctuate throughout the season and vary between seasons. But as seen in Figure 4.2 in Chapter 4.2, PWL usually persists over several days. Hence sampling training data as in Figure 7.1 (at random), one might choose a given day with PWL as training data, and the next day as test data. This causes the features to have very similar values (*i.e.*, temperature and snow depth at different elevations). Hence using random training set (R) might lead to overfitting by capturing these seasonal dependencies, as seen in Table 6.5. Boosting produced extremely high predictive measurements when it used the random training set, but its predictive performance was

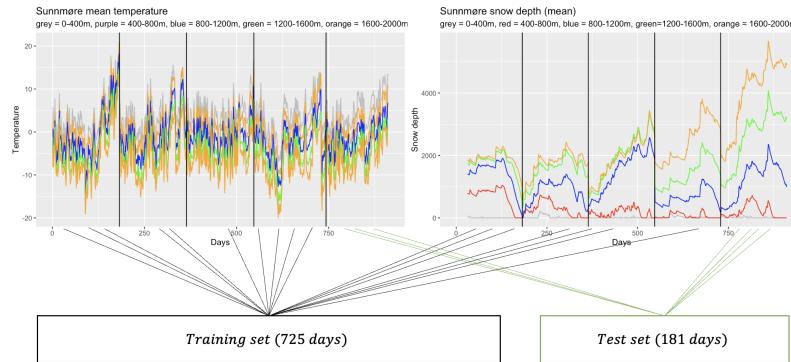


Figure 7.2: Illustration of how 4F (four first seasons) training set was sampled for Sunnmøre, using $temp$ ($^{\circ}C$) and $snowdepth$ (mm) as features. The training set included all observations from the first four seasons, and the test set was the observations from the last season (2021/22).

significantly lower for the other three training sets, see Tables 6.5 and 6.6. The training sets 4F, 4L, and 3/1 can be considered *totally unseen*, since the training data is four entire seasons and the test set is the last unseen season.

So using 4F, 4L and 3/1 produces more realistic predictions, and for an upcoming season, one would use all preceding seasons to predict the existence of PWL in the following season. However, using 4F as a training set did not produce as good predictions as 4L for $M_{time-dependent}$. But the test set for 4F is the last 2021/22 season, the rightmost part of Figure 7.1, was quite different in terms of snow depth, and was an outlier in terms of the other seasons. Hence the models trained on 4F were "surprised" by the feature values of this test set, and performed poorly. However, using training set 4L includes this outlier season in its training, then the training data captures a broader specter of feature values and performs better on its test set.

Another way we sense that seasonal dependencies lead to overfitting is when we use the training sets 4F, 4L, and 3/1. Here, the out-of-bag estimates for bagging and random forest are extremely high. This makes sense because the bootstrapping part of these algorithms causes not all training observations to be used, as discussed in Chapter 5.4. And those not used, the out-of-bag observations, are then used as the test set. Hence this out-of-bag test set is data within the same seasons as the training data, therefore capturing these seasonal dependencies.

7.2 Comments on the three models

Here we will discuss each of the three models ($M_{stationary}$, $M_{stationary*}$, and $M_{time-dependent}$), including the extension M_{MLP} , from Chapter 6. Tables 6.1 to 6.8 show the results that will be discussed next. The bold numbers in these tables are used throughout the following discussion to amplify some of the key findings.

7.2.1 Stationary temperature model

Table 6.1 shows that the unpruned tree and pruned tree, for both regions, do a good job of predicting if PWL does *not exist*, which leads to high *specificity*. However, it does a poor job in predicting if PWL *exists* on a given day, which results in low *sensitivity*. We are interested in a good predictive model, meaning we want high specificity and especially sensitivity. If the sensitivity is low, it means there are a lot of false negatives. In the eyes of the avalanche forecast, predicting no PWL on a day with PWL is a major problem. This is because a false negative can give the impression of safer conditions than reality and should be avoided. Hence the sensitivity of $M_{stationary}$ of approximately 0.10-0.30 is neither impressive nor valuable.

Tables 6.1 and 6.2 show that the OOB estimates for random forest and bagging have the highest

sensitivity for $M_{stationary}$. This is probably because the OOB estimates are tested on days within the training data. In that sense, not being totally unseen test data, hence capturing the seasonal dependencies as discussed in Chapter 7.1. For training set 3/1, the *RF* ($m = 2$) measurements show that for this imbalanced dataset it is easier for a model to get high specificity than high sensitivity. This is due to the few numbers of actual positives (1s) in the dataset, because a model that always predicts 0 (no PWL) will be correct approximately 70% of the time, see Figure 4.4.

In general, $M_{stationary}$ does not perform very well, due to its simplistic nature with only 5 features. For the majority, boosting gave the best results for the different training and test sets, probably because boosting *learns slowly* and is an ensemble method that takes a lot of *weak learners* to produce a *committee* that performs fairly well. When this is said, none of the methods led to any reliable results, see Tables 6.1 and 6.2. $M_{stationary}$ could be looked upon as an *initial model*, or a very simplistic model, which was a starting point for this thesis work. Increasing the number of features will likely capture more information, and thus do a better job of predicting this binary outcome.

7.2.2 Stationary temperature model + snow depth as an additional feature

When implementing snow depth as an additional feature we obtain the $M_{stationary*}$ model. This model is stationary because it only uses features for the given day i the response is forecasted. Tables 6.3 and 6.4 show a trend that the specificity (true negative rate) is very high. Meaning that the models do a good job of classifying 0 (no-PWL). This is the same trend as $M_{stationary}$ and is reasoned with the imbalanced dataset (Figure 4.4). To aid the forecast, the goal should be to create a model that also predicts many true positives (sensitivity).

The predictive measurements had the same trend as for $M_{stationary}$, in the sense that the random training set R captures the seasonal dependencies, and thus yields very high measurements. When looking at the 4F training set, $M_{stationary*}$ outperforms $M_{stationary}$ by a lot, see Tables 6.1 and 6.3. Here, bagging, RF, and boosting performed the best. Comparing boosting in these two models shows a considerable improvement in predicting true positives (sensitivity) when adding snow depth as an additional feature ($M_{stationary*}$). This comes at the cost of slightly lower specificity. In general, the results for $M_{stationary*}$ yield decent results for particular methods and particular training sets. The problem is that the results are inconsistent, the methods are extremely sensitive to what their training data is, and are therefore non-robust methods. When this is said, including $snow_{depth}$ (mm) as an additional feature led to (in general) better predictions than the simpler model $M_{stationary}$. This was expected, due to the formation of PWL and its dependence on the temperature gradient.

7.2.3 Time-dependent model using temperature and snow depth

The snowpack is a complex mixture of different snow grains, and the water molecules have numerous different shapes and different abilities within the snowpack. The snow will always go through some sort of metamorphism, either *weak-gradient*, *wet snow* or *kinetic*. Kinetic metamorphism is dominating when the temperature gradient is larger than $10^{\circ}C\ m^{-1}$. If this happens over several days, PWL will form within the snowpack, as discussed earlier. From this we would expect the model that considers the aspect of time to perform better than a model which does not. This was generally the case when looking at the different methods for different training sets, and including time was a good decision in adding complexity to the models.

The results in Tables 6.5 and 6.6 show that the seasonal dependencies (R training set) cause very high predictive measurements, as for the preceding models. When inspecting boosted classification trees on the 4L training set, we see that they yield good and more realistic results. Here, the very snowy season of 2021/22 is included in the training, and tested on the 2017/18 season. If an upcoming season had feature values similar to 2017/18, we would expect a boosted model to produce good results. Nevertheless, one cannot neglect the big differences in predictive performance

based on which training set was used.

Random forest and choice of m

For random forests we used the benchmark value of $m = \sqrt{p}$, and this produced variable results. Figure 7.3 shows that for $M_{time-dependent}$ (with training set $4L$) in Sunnmøre, $m = \sqrt{p}$ (purple dashed line) is not necessarily the best in terms of predictive performance. When $m = \sqrt{60} \approx 8$, the predictive measurements would not be as good as they could. This is because essential features might be left out in the tree-making, as discussed in Chapter 5.5. Hence m should be treated as a tuning parameter when the number of features is high, which is the case for the dataset used here (60 features).

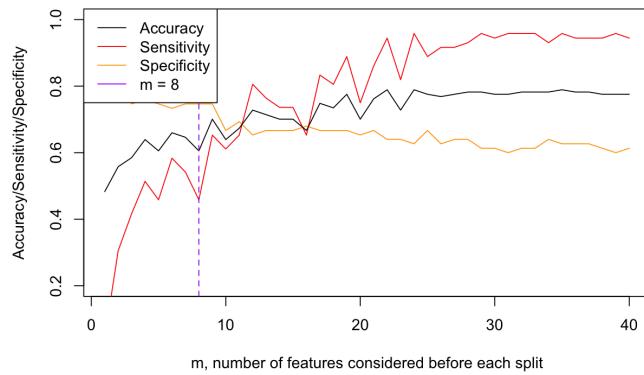


Figure 7.3: Predictive measurements, using different m values, for random forest on Sunnmøre with $M_{time-dependent}$ using training set 4L.

However, this discussion depends on what is considered a "good" prediction. In this thesis, the goal is to aid the avalanche forecast in predicting the existence of PWL. Hence one might prefer a more conservative model, which has more false positives than false negatives. If a model predicts no PWL when the layer actually exists (low sensitivity), it can be dangerous because of the harm PWL can cause. Hence informing the users of the forecast that PWLs do not exist, when it actually exists, should be avoided. Predicting PWL when they do not exist (low specificity) is not good either, because the model will "cry wolf" and will not be trustworthy for its user. Our main intention is therefore to reach high specificity and especially sensitivity.

Continuing the discussion of using random forest for Sunnmøre ($4L$), Figure 7.3 shows that when m approaches 30 the sensitivity gets very high, approximately 0.94. Figure 7.4a shows the confusion matrix for $m = 8$, while Figure 7.4b shows the confusion matrix for $m = 30$.

Reference			Reference		
Prediction	0	1	Prediction	0	1
0	56	39	0	46	4
1	19	33	1	29	68

Figure 7.4: Confusion matrix for random forest on Sunnmøre with $M_{time-dependent}$ on training set 4L.
 (a) $m = 8$, (b) $m = 30$.

Figure 7.4 shows that changing the tuning parameter m for the random forest dramatically changes the sensitivity. Decreasing the number of false negatives by 35, and at the same time (only) increasing the false positives by 10. Further, 8 features chosen at random at each split will not capture enough information to make a solid prediction in determining if PWLs exist, as shown in

Figure 7.4. Table 7.1 compares the predictive measurements between $m = 8$ and $m = 30$. Here we see that all predictive measurements increases, except specificity. Increasing m to 30 will lead to significantly more true positives, with the cost of a few more false positives. For Sunnmøre and 4L training set, the results of the random forest classifier with $m = 30$ is fairly similar to the boosted model with $d = 4$, $B = 2000$, and $\lambda = 0.05$ (Table 6.2).

Table 7.1: $M_{time-dependent}$ random forest results for $m = 8$ and $m = 30$ in Sunnmøre.

Training: 4L	$m = 8$	$m = 30$
Accuracy	0.6054	0.7755
Specificity	0.7467	0.6133
Sensitivity	0.4583	0.9444
Balanced accuracy	0.6025	0.7789

Random forests and bagging lose the interpretation advantage that classification trees offer, since it is really no longer a tree when we aggregate all $B = 500$ trees together. But by using the R function `reptrtree:::plot.getTree(ranf.sunnmøre, k=2)` we can plot the second tree created in the random forest algorithm where each split considers $m = 30$ random features out of the total $p = 60$ features. Here we see that this particular tree makes its root node, and first internal nodes, by splitting on temperature and snow depth at day $i - 4$ and $i - 5$, see Figure 7.5. This could have its reason due to PWLs usually requiring several days of a consistently high temperature gradient.

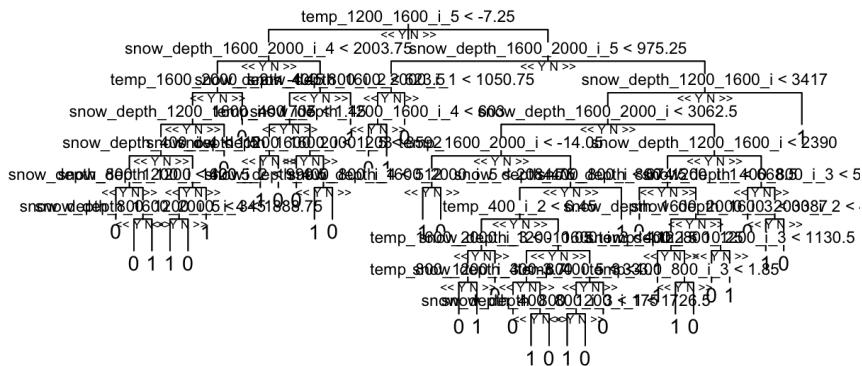


Figure 7.5: Second classification tree created in the random forest algorithm when $m = 30$ for Sunnmøre with training set 4L.

A similar trend is observed for Romsdal, where the benchmark value $m = \sqrt{p}$ does not yield the best predictive performance, see Figure 7.6.

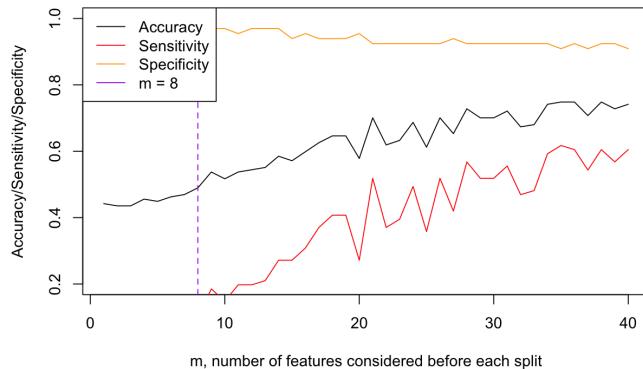


Figure 7.6: Different m values for the random forest on Romsdal with $M_{time-dependent}$ (training set 4L).

7.2.4 Multilayer perceptron

Tables 6.7 and 6.8 show the results of our feedforward neural network (M_{MLP}) with three hidden layers, using different activation functions in the hidden layers. As expected, the network performed well for the random R training set, due to seasonal dependencies as discussed in Chapter 7.1. As with the previous methods, the results vary on the training data used, and is a general trend in the results presented in this thesis. To our surprise, the network performed a lot worse for the other training sets when compared with the methods from $M_{time-dependent}$. Although one should not necessarily be surprised. Because simpler methods often outperform more complex approaches. This was the case for $4F$ and $4L$ training sets, but when the training set $3/1$ was used, M_{MLP} performed better than the methods from $M_{stationary}$, $M_{stationary*}$, and $M_{time-dependent}$.

The choice of activation functions in the hidden layers (ReLU, Sigmoid, or TanH) showed not to be of importance. Since each function produced fairly similar results, see Tables 6.7 and 6.8. Throughout the training of the neural network, the cross entropy (loss function) steadily decreased, but it is possible to suspect that the network found noise patterns in the data and therefore performing poorly when trained on the training sets that do not capture seasonal dependencies, and thus generalize poorly.

These different models perform quite differently based on the training and test set used, and it seems like some models with a particular training and test set are capturing more generic features than others.

7.3 Avalanche theory discussion and how this can aid the forecast

We have used different statistical and machine learning techniques to predict a complex binary outcome. However, there are some problems with the models used here concerning the avalanche theory and theory about kinetic transformation, as discussed in Chapter 3.3.1.

The boosted model from $M_{time-dependent}$ performed the best on totally unseen test data (training sets $4F$, $4L$, $3/1$), especially using the training set $4L$. Where the features used here were $snow_{depth}$ and $temp$ (air temperature) at different elevations for day i , $i - 1, \dots, i - 5$. This makes sense in trying to predict *facets* and *depth hoar*, but is problematic if we want to predict *surface hoar*, see Figure 3.10 for grain forms. Surface hoar forms when its water vapor in the air, a clear sky (at night), and a slight breeze of wind (typically 1 to 3 m/s). But implementing this criterion as features showed to be problematic. The dataset at hand made it challenging to find suitable features for this problem. Hence the models presented in this thesis could potentially only capture true positives in the sense of facets and depth hoar, not surface hoar.

Facets (and depth hoar) usually form when there is a high temperature gradient over time. Meaning that kinetic transformation can dominate the snowpack such that facets form. But facets can form rapidly close to the surface, as discussed in Chapter 3.3.1. This could happen by *dry snow on top of wet snow, fluctuating temperatures throughout the day, and balanced radiation and radiance*. Features used in the models in this thesis are daily measurements, not several measurements throughout the day. Hence it is difficult to capture these quick and sudden formations of PWLs. All of this thesis work is based on using *indirect features* of the temperature gradient, which is the key component of kinetic transformation in the snowpack. Ideally, one would like to have continuous temperature measurements from *within* the snowpack to forecast this problem. But this is not feasible as for now.

It is also worth repeating that the response in this thesis is whether or not PWL is *forecasted* or not. This is determined by an expert, usually a meteorologist or hydrologist at NVE. Since PWLs come with danger, it is possible to suspect that the forecasters sometimes choose to be conservative. By this we mean that if there has been a persistent weak layer over a given period, but if the forecasters are unsure if the layer has been destroyed by weak-gradient or wet snow metamorphism (Figure 3.11), then the forecasters might forecast the avalanche problem PWL to

avoid a false negative. In this thesis, the forecast is the *true prediction*, but one cannot eliminate human error. This was not added in the analysis, but commented here as a potential source of error.

So far in this discussion, we have sort of discarded the results from the random (R) training set due to overfitting. Nevertheless, this methodology could be implemented in these maritime regions as an aiding tool, especially when there is a lack of personnel. Then we would know the *status quo* on the snowpack for, *i.e.*, day i . Further, if the personnel is lacking on the following days ($i+1, i+2, \dots$), then a model that captures that specific *seasonal dependency* (and all preceding seasons) could be used in favor of the forecast. This does not need to be a difficult task, by sequentially updating the training data as the days go by. If the methods discussed here were to be implemented in regions where there are no avalanche observers in the field, only automatically gathered weather data, more data would certainly help. Nevertheless, we then get the problem of verifying if the predicted outcome is correct or false.

An important note is that the available data were five seasons from 2017/18 - 2021/22. This resulted in a total of approximately 900 days with data points. The methods used seem to suffer to generalize well, which could have its source in the lack of big data. This is a problem encountered in many machine learning problems and could be solved by gathering more data as the winters pass. Alternatively, one could train, *e.g.*, $M_{time-dependent}$ on seasons from all maritime regions in Norway to see if this captures a broader specter of feature values that will generalize better for unseen test data.

Chapter 8

Closing remarks and future research

This chapter will summarize this thesis, its findings, and closing remarks. Additionally, we will provide some suggestions for future research on this subject. These closing remarks will try to answer the goal and research questions asked in Chapter 1.1.

8.1 Closing remarks

In this thesis, we have tried to create different statistical and machine learning models to predict the existence or non-existence of PWL(s) on a given day. In doing so, many different learning techniques were used, like classification trees, pruning of trees, bagging, random forests, boosting, and neural networks. With the dataset at hand, consisting of five full winter seasons for Sunnmøre and Romsdal, we tested the models using different training sets (R , $4F$, $4L$, and $3/1$). This led to variable results, see tables in Chapter 6, indicating that the methods suffer from non-robustness. The boosted model ($M_{time-dependent}$) with training set $4L$ produced the best predictions, which included the outlier snowy season (2021/22) in the training set, see Figures 4.6 and 4.7.

In Chapter 1.1, we presented the overall goal as follows:

Goal: *Develop a robust statistical model to predict whether a maritime region will have a persistent weak layer(s) or not on a given day.*

In Chapter 6 we found that boosting methods from $M_{time-dependent}$ performed the best when excluding the random training set (R) that captures the seasonal dependencies. Boosting performed best when using the training set $4L$ (the four last seasons). However, it did not perform as well when trained on the two other training sets, $4F$ and $3/1$. The reasoning behind this is that the methods are non-robust, and hence very sensitive to what data it has been trained and tested on. Another explanation for this is that the model oversimplifies the problem at hand. Meaning that using this indirect approach to the temperature gradient by looking at snow depth and air temperature is not good enough. The transformations going on within the snowpack are simply too complex to mimic using only air temperature and snow depths over several days.

Kinetic transformation is the reason behind formation of facets and depth hoar, but the dataset made it difficult to predict days when the weak layer was surface hoar. These layers are also formed by kinetic transformation, but not from a temperature gradient within the snowpack. Surface hoar usually form on cold, clear nights, but we had no data on the cloud cover. This is an obvious shortcoming in this thesis. Also, the formation of faceted crystals *close to the surface* can occur very rapidly (Chapter 3.3.1), and these daily observations make them difficult to predict.

The main goal of this thesis was to see whether a statistical model could predict the existence

of PWL or not, and in doing so, some of the models performed decently. Although we would not recommend replacing the workflow of using professional observers in the field with the models proposed in the thesis. Nevertheless, the models could be used as aiding tools. An example is if a maritime region has similar temperatures and snow depths as, *e.g.*, 2017/18 season (the first season of the dataset), then using the boosted $M_{time-dependent}$ with training 4L could definitely aid the forecasters in determining if they should include the avalanche problem PWL or not.

Most studies done on similar subjects (Chapter 2) conclude that their model can be used to *aid* the already existing avalanche forecast, not replace it. The same goes for the findings in this thesis. This is probably due to the complexity of the continuous transformations happening within the snowpack at any given time. A lot of work has been put into discussing seasonal dependencies, and using the random training set makes the results not really representative, and therefore one has to use methods where training and test sets are from separate seasons. In real life, a method in the intersection between separate seasons and random training sets might be optimal, because when a forecasting season has started (1st of December to 31st of May), one should use weather data for the preceding seasons, but also the preceding days from that particular season. In practice, the forecaster might know whether or not a PWL existed in the preceding days, and this is information that could be put to use through, *e.g.*, weighted features. This could be implemented in a model that is daily updated with both weather data and whether or not PWL existed the preceding days, and would be helpful if a region is lacking observers. However, if PWL exists on a given day and the weather is stable and cold, the weak layer will persist until wet snow metamorphism destroys the layer.

To answer the goal and research questions, we conclude that the methods presented in this thesis cannot replace the existing workflow of the forecast due to inconsistency in the results. Nevertheless, we have presented a reproducible workflow, and when trained upon more data, the robustness will likely increase and can then aid the forecasting in maritime regions.

8.2 Future research

The results indicate that the models are non-robust, since the results were so variable based on which training set used. This non-robustness might originate from a lack of big data. This thesis had five winter seasons at hand, for both training and testing. The results would potentially be quite different if we had 10, 15 or 20 seasons at hand. Then the natural fluctuations in feature values would be captured, and hence not be as sensitive to outlier-seasons as the dataset used in this thesis. This would likely improve the overall robustness of the methods, and therefore be more reliable. Due to the complexity of the formation of PWLs, it would be preferable to have as much relevant data as possible. Since this thesis has focused the most on facets and depth hoar, and not so much on surface hoar, it would be interesting to have data on air humidity and cloud cover. In that case, one might be able to predict surface hoar by its own model, and facets and depth hoar with methods proposed in this thesis.

Throughout this thesis, the models have been *region-specific*. Meaning that the data used for training and testing both came from the same region. An idea could be to train a model on, *e.g.*, five different maritime regions, where each has five seasons of data. This would result in $5 \cdot 5 = 25$ seasons worth of training data and would remove the necessity of waiting several years to gather more data. Another advantage would be that we could create larger validation and test sets to perform cross-validation. Then the workflow presented in this thesis could easily be implemented on this larger dataset. A potential problem with this approach could be that a model trained on several different regions might capture more noise, and therefore generalize worse than a region-specific model.

In this thesis we focused on maritime regions, where the dataset were imbalanced with more 0s than 1s. It would be interesting to use the workflow from this thesis for intermountain or continental regions, where the occurrences of PWLs would differ from these maritime regions. This would be fairly easy to reproduce, with the code from Appendix A.2 and A.3. The data from other regions of interest could easily be retrieved by small adjustments in the code from Appendix A.1.

An additional proposal, is that one could consider creating decision trees manually in cooperation with snow scientists, forecasters, and snow observers from the given region of interest. This could be very difficult due to the complexity of the problem, but perhaps create insight to where the classification trees using binary recursive splitting "goes wrong".

The majority of this thesis focuses on classification trees, boosting, bagging, pruning, and random forests. Additionally, one MLP was tried for the different training sets. From our results, boosting and random forests (when m is tuned properly) outperforms the other methods. However, it would be interesting to compare these methods to other modern machine learning techniques, when handed the same dataset. The features used in this thesis are highly correlated, as seen in Figures 4.5 to 4.11, and an in-depth usage of *feature engineering* could be a good idea. A specific idea is to use graph attentional layers (GAT) to get "enhanced features", and perhaps increase predictive performance (Veličković et al., 2017).

Bibliography

- Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2), 123–140. <https://doi.org/10.1007/BF00058655>
- CAA. (2016). *Observation guidelines and recording standards for weather, snowpack and avalanches*. Retrieved August 23, 2022, from https://cdn.ymaws.com/www.avalancheassociation.ca/resource/resmgr/standards_docs/ogrs2016web.pdf
- Dkengne Sielenou, P., Viallon-Galinier, L., Hagenmuller, P., Naveau, P., Morin, S., Dumont, M., Verfaillie, D., & Eckert, N. (2021). Combining random forests and class-balancing to discriminate between three classes of avalanche activity in the french alps. *Cold Regions Science and Technology*, 187, 103276. <https://doi.org/https://doi.org/10.1016/j.coldregions.2021.103276>
- EAWS. (2017). *Typical avalanche problems*. Retrieved August 22, 2022, from https://www.avalanches.org/wp-content/uploads/2019/05/Typical_avalanche_problems-EAWS.pdf
- EAWS. (2022). *European avalanche danger scale (2018/19)*. Retrieved August 23, 2022, from https://www.avalanches.org/wp-content/uploads/2019/05/European_Avalanche_Danger_Scale-EAWS.pdf
- Fahrmeir, L., Kneib, T., Lang, S., & Marx, B. (2013). *Regression: Models, methods and applications*. Springer Berlin Heidelberg. <https://books.google.no/books?id=EQxU9iJtipAC>
- Faraway, J. J. (2006). *Extending the linear model with r: Generalized linear mixed effects and nonparametric regression models*. Chapman; Hall/CRC.
- Faris, H., Aljarah, I., & Mirjalili, S. (2016). Training feedforward neural networks using multi-verser optimizer for binary classification problems. *Applied Intelligence*, 45(2), 322–332. <https://doi.org/10.1007/s10489-016-0767-1>
- Friedman, J. H. (2002). Stochastic gradient boosting [Nonlinear Methods and Data Mining]. *Computational Statistics and Data Analysis*, 38(4), 367–378. [https://doi.org/https://doi.org/10.1016/S0167-9473\(01\)00065-2](https://doi.org/https://doi.org/10.1016/S0167-9473(01)00065-2)
- Granger, R. (2019). *Crystal growth physics in dry snow metamorphism: characterisation and modeling of kinetic effects* (Theses). Université Grenoble Alpes. <https://tel.archives-ouvertes.fr/tel-03092266>
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference and prediction* (2nd ed.). Springer. <http://www-stat.stanford.edu/~tibs/ElemStatLearn/>
- Ho, T. K. (1995). Random decision forests. 1, 278–282 vol.1. <https://doi.org/10.1109/ICDAR.1995.598994>
- Hosseini, M., Powell, M., Collins, J., Callahan-Flinton, C., Jones, W., Bowman, H., & Wyble, B. (2020). I tried a bunch of things: The dangers of unexpected overfitting in classification of brain data. *Neuroscience and Biobehavioral Reviews*, 119, 456–467. <https://doi.org/https://doi.org/10.1016/j.neubiorev.2020.09.036>
- James, G. (1998). *Majority vote classifiers: Theory and application*. Stanford University. <https://hastie.su.domains/THESES/gareth.james.pdf>
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning: With applications in r*. Springer New York. <https://doi.org/https://doi.org/10.1007/978-1-4614-7138-7>
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An introduction to statistical learning: With applications in r (2nd edition)*. Springer New York. <https://doi.org/https://doi.org/10.1007/978-1-0716-1418-1>

-
- Jamieson, J. B. (1995). *Avalanche prediction for persistent snow slabs*. PRISM. <https://doi.org/10.11575/PRISM/18207>
- Landrø, M. (2002). *Skredfare: Snøskred, risiko og redning*. Featureforlaget AS.
- Landrø, M., Hetland, A., Engeset, R. V., & Pfuhl, G. (2020). Avalanche decision-making frameworks: Factors and methods used by experts. *Cold Regions Science and Technology*, 170. <https://doi.org/https://doi.org/10.1016/j.coldregions.2019.102897>
- Marienthal, A., Hendrikx, J., Birkeland, K., & Irvine, K. M. (2015). Meteorological variables to aid forecasting deep slab avalanches on persistent weak layers. *Cold Regions Science and Technology*, 120, 227–236. <https://doi.org/https://doi.org/10.1016/j.coldregions.2015.08.007>
- Müller, K. (2020). *Snøomvandling*. Retrieved August 25, 2022, from <https://publikasjoner.nve.no/faktaark/2020/faktaark2020.01.pdf>
- NVE. (2016). *Bruk av regionale snøskredvarsler*. Retrieved August 22, 2022, from https://publikasjoner.nve.no/faktaark/2016/faktaark2016_04.pdf
- NVE. (2019). *Nve-forsvaret felthåndbok*. Retrieved October 25, 2022, from https://varsom.no/media/dlin3fyz/nve-forsvaret_feltha-ndbok_innmat_v1.pdf
- NVE. (2021). *Snø*. Retrieved August 22, 2022, from <https://nve.no/vann-og-vassdrag/vannets-kretsloep/sno/>
- NVE. (n.d.-a). *Matrise for å sette faregrad*. Retrieved August 24, 2022, from <https://varsom.no/snokred/snokredskolen/del-informasjon/matrice-for-a-sette-faregrad/>
- NVE. (n.d.-b). *Om snøskredvarslingen*. Retrieved August 22, 2022, from <https://www.varsom.no/snokred/snokredvarsling/om-snokredvarslingen/>
- NVE. (n.d.-c). *Skredkort - gjør turen tryggere*. Retrieved December 1, 2022, from <https://www.varsom.no/media/jzwlxao0/plakat-skredkort-norsk.pdf>
- NVE. (n.d.-d). *Slik lager vi snøskredvarsler*. Retrieved August 23, 2022, from <https://www.varsom.no/snokred/snokredvarsling/slik-lager-vi-snokredvarsler/>
- NVE. (n.d.-e). *Snøskredstørrelser*. Retrieved August 23, 2022, from <https://varsom.no/snokred/snokredskolen/snokredvarselet-forklaring/snokredstørrelser/>
- NVE/EAWS. (2018). *Typiske skredproblem*. Retrieved August 25, 2022, from https://varsom.no/media/fgan3dhv/typiske_skredproblemer-eaws.pdf
- Ridgeway, G. (2020). *Generalized boosted models: A guide to the gbm package*. Retrieved October 4, 2022, from <http://127.0.0.1:65473/help/library/gbm/doc/gbm.pdf>
- Rokach, L., & Maimon, O. (2010). Classification trees. In O. Maimon & L. Rokach (Eds.), *Data mining and knowledge discovery handbook* (pp. 149–174). Springer US. https://doi.org/10.1007/978-0-387-09823-4_9
- Temper, B. (2018). *Staying alive in avalanche terrain (3rd edition)*. Mountaineers Books.
- UAC. (n.d.). *Blog: Glide avalanche primer*. Retrieved August 22, 2022, from <https://utahavalanchecenter.org/blog/15571>
- Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2017). Graph attention networks. <https://doi.org/https://doi.org/10.48550/arxiv.1710.10903>
- Ville, B. D. (2013). Decision trees. *WIREs Comput Stats*, 5, 448–455. <https://doi.org/https://doi.org/10.1002/wics.1278>
- Widforss, A. (2021). *Teaching machines to recognise avalanche conditions* (Master's thesis). Luleå University of Technology. <https://www.diva-portal.org/smash/get/diva2:1588525/FULLTEXT01.pdf>
- Ying, X. (2019). An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, 1168, 022022. <https://doi.org/10.1088/1742-6596/1168/2/022022>

Appendix A

Code examples

Here we present how the data was retrieved from Varsom, and some examples of code that were used in the thesis. The examples showcase how we implemented the different machine learning methods for Romsdal on the random *R* training set. All examples are analogous for Sunnmore, and the other training sets. The code for finding optimal values of tuning parameters are excluded from this section, but described in Chapter 6.

A.1 Retrieving data from Varsom

This Python code (provided by NVE) was used to fetch the dataset for this thesis. It gathered weather information from Varsoms internal tool Avalanche Problem Solver (APS), and the avalanche problem and avalanche danger from Varsom.

```
 1 from regobslib import *
 2 import datetime as dt
 3 import numpy as np
 4 import pandas as pd
 5
 6 REGIONS = [SnowRegion.JOTUNHEIMEN,
 7             SnowRegion.TROMSO,
 8             SnowRegion.SUNNMORE,
 9             SnowRegion.ROMSDAL]
10
11 START_DATE = dt.date(2017, 11, 1)
12 STOP_DATE = dt.date(2022, 7, 1)
13
14 connection = Connection(prod=True)
15
16 varsom_query = connection.get_varsom(START_DATE, STOP_DATE, REGIONS)
17 weather_query = connection.get_aps(START_DATE, STOP_DATE, REGIONS)
18
19 for region in REGIONS:
20     danger_level_data = varsom_query[region].to_danger_level_series()
21     avalanche_problem_data = varsom_query[region].to_problem_frame()
22     weather_data = weather_query[region].to_frame(with_wind_dir=True)
23
24     id = f"{region.name}_{START_DATE}_{STOP_DATE}"
25     danger_level_data.to_csv(f"danger-levels_{id}.csv", sep=";")
26     avalanche_problem_data.to_csv(f"avalanche-problem_{id}.csv", sep=";")
27     weather_data.to_csv(f"weather_{id}.csv", sep=";", float_format = str)
```

A.2 R code

The following code shows how the classification trees were fitted, pruned, and how this was implemented to create a confusion matrix on its test data. Additionally, we showcase the code used to fit bagging, random forests, and boosting to the time-dependent model ($M_{time-dependent}$).

Classification trees

```
1 library(tree)           #Package for the trees
2 library(caret)          #Package for confusion matrix.
3 library(randomForest)   #Package for bagging and random forests.
4 library(gbm)            #Package for gradient boosting machines.
5
6 set.seed(563)           #For reproducible results.
7 romsdal.trainID = sample(1:906, 679, replace = FALSE) #Sampling random training set
     (R).
8
9 #timedependent_dataframe_romsdal is dataset with 60 features.
10 romsdal.train = timedependent_dataframe_romsdal[romsdal.trainID, ] #75% of data.
11 romsdal.test = timedependent_dataframe_romsdal[-romsdal.trainID, ] #The remaining
     25% of data.
12
13 ### Classification tree for Romsdal ####
14
15 #Creating the tree with deviance as splitting criterion.
16 tree.romsdal = tree(as.factor(pwl_romsdal) ~ ., data = romsdal.train,
17                      split = 'deviance')
18
19 #Create predictions on test set.
20 pred = predict(tree.romsdal, romsdal.test, type = "class")
21
22 #Create confusion matrix that calculates the predictive measurements. "1" (
23 #   existence of PWL) is positive.
24 romsdal_pred_confMat <- confusionMatrix(reference = as.factor(romsdal.test$pwl_
     romsdal), data = pred, positive = "1")
25
26 #Print to showcase the confusion matrix.
27 romsdal_pred_confMat
```

Pruning of classification trees

```
1 ### Pruning ####
2 tree.romsdal <- tree(as.factor(pwl_romsdal) ~ ., data = romsdal.train, control =
     tree.control(nrow(data_romsdalen),
      mincut = 2, minsize = 4, mindev = 0.001), split = 'deviance')
3 cv.romsdal <- cv.tree(tree.romsdal, K = 5)      #K=5 Cross validation.
4 plot(cv.romsdal$dev ~ cv.romsdal$size, type = "b", lwd = 2, col = "red",
5      xlab = "Tree Size", ylab = "Deviance")        #Find minimum of cross validation.
6
7
8 prune.romsdal <- prune.tree(tree.romsdal, best = 9) #Prune tree with size equal to
     the parameter "best", in this example "best = 9".
9
10 tree.pred = predict(prune.romsdal, romsdal.test, type = "class") #Predict on test
     set.
11
12 romsdal_confMat <- confusionMatrix(reference = as.factor(romsdal.test$pwl_romsdal),
     data = tree.pred, positive = "1") #Create confusion matrix that calculates the
     predictive measurements. "1" (existence of PWL) is positive.
13
14 romsdal_pred_confMat #Print to showcase the confusion matrix.
```

Bagging

```
1 set.seed(563)
2 #Parameter mtry = number of predictors.
3 bag.romsdal <- randomForest(as.factor(pwl_romsdal) ~., data = romsdal.train, mtry =
60, ntree = 500 , importance = TRUE, split = 'deviance', na.action=na.omit)
4 bag.romsdal
5
6 # Predict on test set.
7 yhat.bag <- predict(bag.romsdal, newdata = romsdal.test)
8 confusionMatrix(yhat.bag, reference = as.factor(romsdal.test$pwl_romsdal), positive
= "1")
```

Random forests

```
1 set.seed(563)
2 #mtry is set to the benchmark value (square root of number of predictors).
3 ranf.romsdal <- randomForest(as.factor(pwl_romsdal) ~., data = romsdal.train, mtry
= 8, ntree = 500 ,importance = TRUE, split = 'deviance', na.action = na.omit)
4
5 # Predict on test set.
6 yhat.ranf <- predict(ranf.romsdal, newdata = romsdal.test)
7 confusionMatrix(yhat.ranf, reference = as.factor(romsdal.test$pwl_romsdal),
positive = "1")
```

Boosting

```
1 set.seed(563)
2 # Parameters n.trees, interaction.depth, shrinkage, and bag.fraction were chosen
# according to the description in Chapter 6.
3 boost.romsdal <- gbm(pwl_romsdal ~., data = romsdal.train, distribution =
'bernoulli', n.trees = 4000, interaction.depth = 3, shrinkage = 0.05, bag.
fraction = 0.8)
4
5 # Predict on test set.
6 yhat.boost.romsdal <- round(predict.gbm(object = boost.romsdal, newdata = romsdal.
test, n.trees = 4000, type = 'response'))
7 pred_boost_romsdal <- confusionMatrix(as.factor(yhat.boost.romsdal), reference = as
.factor(romsdal.test$pwl_romsdal), positive = "1")
```

A.3 Python code

Here we present a simplified code of M_{MLP} , the neural network extension of the time-dependent model. This example showcase the random training set for Romsdal, but the code is similar for Sunnmore and the other training sets.

```
1 # Importing pandas, os, torch and torch.nn to create the neural network.
2 import os
3 import pandas as pd
4 import torch
5 import torch.nn as nn
6
7 ### FOR TRAINING ####
8 ! pip install pytorch-lightning
9
10 from pytorch_lightning import LightningModule, Trainer
11 from torch.utils.data import Dataset, DataLoader
12 import torch.nn.functional as F
13
14 from sklearn.model_selection import train_test_split # Creating test/training sets.
```

```

15 from sklearn.preprocessing import StandardScaler      # Standardize the data.
16 from sklearn.metrics import confusion_matrix        # To get confusion matrix.
17
18 # Import processed dataset that we exported from R.
19 df = pd.read_csv("timedependent_dataframe_romsdal.csv")
20 df = df.dropna() #Exclude rows with NA's. Meaning the first 30 days of
→ 2017/18-season.
21
22 ### PREPARE TRAINING ###
23 class CustomDataset(Dataset):
24     def __init__(self, X, Y):
25         super().__init__()
26         self.X = X
27         self.Y = Y
28
29     def __getitem__(self, index):
30         x = self.X[index].float()
31         y = self.Y[index].float()
32         return x, y
33
34     def __len__(self,):
35         return self.X.shape[0]
36
37 class CustomModel(LightningModule):
38     def __init__(self):
39         super().__init__()
40         # Nonlinear multi perceptron with TanH hidden layer, this was tuned as
→ described in Chapter 6.
41         self.model1 = nn.Sequential(nn.Linear(60, 60),
42                                     nn.Tanh(),
43                                     nn.Linear(60, 60),
44                                     nn.Tanh(),
45                                     )
46
47         self.model2 = nn.Sequential(nn.Linear(60, 60),
48                                     nn.Tanh(),
49                                     nn.Linear(60, 60),
50                                     nn.Tanh(),
51                                     )
52
53         self.model3 = nn.Sequential(nn.Linear(60, 60),
54                                     nn.Tanh(),
55                                     nn.Linear(60, 60),
56                                     nn.Tanh(),
57                                     )
58         # Sigmoid in outer layer.
59         self.clf = nn.Sequential(nn.Linear(60, 1), nn.Sigmoid())
60
61         self.threshold= 0.5 # Threshold turning predictions into 0's or 1's.
62
63     # Feedforward neural network.
64     def forward(self, x):
65         out1 = self.model1(x)
66         out2 = self.model2(x)
67         out3 = self.model3(x)
68         out = (out1 + out2 + out3) / 3
69
70         yhat = self.clf(out)
71         #yhat = yhat > self.threshold
72         return yhat
73
74     def training_step(self, batch, batch_nb):

```

```

75     x, y = batch
76     yhat = self.forward(x)
77     loss = F.binary_cross_entropy(yhat, y) #Cross-entropy used in this binary
    ↵ classification problem.
78     return loss
79
80 def configure_optimizers(self):
81     return torch.optim.Adam(self.parameters(), lr=0.001) #Optional learning rate,
    ↵ has to be tuned.
82
83 # Split features and predictors to separate dataframes.
84 X = df.iloc[:, :-1]
85 Y = df.iloc[:, [-1]]
86
87 scaler = StandardScaler()
88 # Standardizing the data.
89 X = scaler.fit_transform(X)
90
91 # Create training/test set. shuffle = True (random training set). random_state = 42
    ↵ for reproducible results.
92 X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25,
    ↵ random_state = 42, shuffle = True)
93
94 X_train_torch = torch.FloatTensor(X_train)
95
96 # To get on correct form to fit in nn.linear.
97 X_test_torch = torch.FloatTensor(X_test)
98 Y_train_torch = torch.FloatTensor(Y_train.values)
99
100 # Building some model. Build a linear model with 60 features as input and 60 as
    ↵ output.
101 linear = nn.Linear(60, 60)
102 linear
103
104 out = linear(X_train_torch)
105
106 # Nonlinear multilayer perceptron
107 # Now we're applying a nonlinear multiperceptron layers to capture non-linear
    ↵ relationships.
108
109 # Trying different models. Tuned as described in Chapter 6.
110 model1 = nn.Sequential(nn.Linear(60, 60),
111                         nn.ReLU(),
112                         nn.Linear(60, 60),
113                         nn.ReLU(),
114                         )
115
116 model2 = nn.Sequential(nn.Linear(60, 60),
117                         nn.ReLU(),
118                         nn.Linear(60, 60),
119                         nn.ReLU(),
120                         )
121
122 model3 = nn.Sequential(nn.Linear(60, 60),
123                         nn.ReLU(),
124                         nn.Linear(60, 60),
125                         nn.ReLU(),
126                         )
127
128 out1 = model1(X_train_torch)
129 out2 = model2(X_train_torch)
130 out3 = model3(X_train_torch)

```

```

131
132 # Take the mean of the three models.
133 out = (out1 + out2 + out3) / 3
134
135 # Using a classifier to end up with (60 by 1), to get a response. nn.Sigmoid used in
136 #→ output layer.
137 classifier = nn.Sequential(nn.Linear(60, 1),
138                             nn.Sigmoid())
139
140 # Find predictions.
141 yhat = classifier(out)
142
143 # Threshold.
144 yhat_bool = yhat.detach().numpy() > 0.5
145 # Confusion matrix on our training set.
146 confusion_matrix(Y_train.values, yhat_bool)
147
148 # Initiate our model.
149 custom_model = CustomModel()
150
151 # Use DataLoader on our time-dependent dataset.
152 train_loader = DataLoader(CustomDataset(X_train_torch, Y_train_torch), batch_size=32)
153 #Batch_size should be tuned.
154
155 # Initialize a trainer.
156 trainer = Trainer(
157     accelerator = "gpu",
158     devices = 1,
159     max_epochs = 1000, #Epoch has to be tuned to optimize training, but avoid
160     #→ overfitting.
161 )
162
163 # Train the model.
164 trainer.fit(custom_model, train_loader)
165
166 # Prediction.
167 yhat = custom_model.forward(X_test_torch)
168
169 # Implement threshold.
170 yhat_bool = yhat > 0.5
171
172 #Create confusion matrix to calculate predictive measurements on our test set.
173 confusion_matrix(Y_test.values, yhat_bool)

```
