# Prediction of snow depth during winter in Helsinki

## 1 Introduction

Weather has always been an aspect that is hard to predict and fluctuates considerably. By combining machine learning and real-life data of weather, I aim to find a particular weather pattern. If the outcome of this project comes up to expectation, people can gain a rough estimate of what weather would look like in the near future based on available weather measurements and they can therefore prepare beforehand for what could happen.

The topic of this project is discussed further in the part Problem Formulation. The methods used in this project are linear regression and polynomial regression, which, together with data process, are in further details in section Methods. Sections Result and Conclusion will wrap up and conclude the better method between the two I used.

## 2 Problem Formulation

In this project, I try to find out if there is any correlation between precipitation amount/minimum temperature and snow depth during winter season in Helsinki. The datapoints are the measurements of weather observations, in particular minimum temperature, precipitation amount and snow depth, that are collected on a daily basis during the interval of December - March of the following year. My data source is the observations published on Finnish Meteorological Institute services website [1] at the observation station in Helsinki Kaisaniemi. The dataset in this project includes measurements of 14 consecutive winters: 2007-2021 as the data of 2021-2022 are not yet available. In the following stages of the project, the features would be minimum temperature and precipitation amount, while the label would be the snow depth.

## 3 Methods

### 3.1 Dataset

About my dataset in this project, I use daily measurements in winter, to be more precise from December to March of the following year, during the interval 2007-2021. The yearly data are downloaded directly from the website of Finnish Meteorological Institute and afterwards combined during the data process stage.

For the features, I decide to pick precipitation amount and minimum temperature out of the datapoints, while choosing snow depth to be my label. Both my features and label appear under the type of Double. There are several missing values in the features so I use Simple Imputer methods in Scikit-learn [2] to replace the missing values with the mean of the features.

The combined dataset includes total of 1688 datapoints of 1688 days over 7 consecutive years, with features being the two columns “Precipitation amount (mm)” and “Minimum temperature (deg C)”, while label being the column “Snow depth (cm)”.

Heuristically, as the temperature goes down, the amount of snow increases, which leads to increasing snow depth. And when the temperature is low enough, the rain would become snow rain, which will also affect the snow depth. So I suspect if there is any correlation between these two and the snow depth during winter where the temperature is extremely low and the amount of rainfall is also considerable.

The whole dataset is splitted into three subsets: training set, validation set and test set. The function that I use is the train_test_split from the Scikit-learn [3], which is implemented two times to produce the three subsets. As I would want to reserve a large number of datapoints for the training set, it would be best for the validation size to be 20-30% of the dataset. And by trial and error, I have found that 25% would be the optimal size for validation set. So the final number is 75%, 12,5% and 12,5% respectively.

## 3.2 Linear Regression

My first method would be linear regression as I intially expect when the minimum temperature decreases and the precipitation amount increases, the snow depth would also increase. This method has the form of

$$\hat{y} = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n$$

with $\hat{y}$ being the dependent variable (also known as the value of predicted label), $x_i$ being the independent variables (values of different features), $\beta_0$ being the constant term (or the intercept) and $\beta_i$ being the coefficients for different features, which can also be referred as slope. The method is expected to produce a line that corresponds to the data. The method implementation instruction can be found on Scikit-learn [4].
Because the quality of the linear predictor is usually to be measured by the squared error loss (or the mean squared error) [5], by minimizing which I can get to the coefficients $\beta_i$ of the regression, so I would use the squared loss error as my loss function. The loss function has the form as below:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

with $y_i$ being the real label value from the dataset and $\hat{y}_i$ being the predicted label value using linear regression. I use this loss function directly from the Scikit-learn [6] website.

## 3.3 Polynomial Regression

The second method to be chosen for this project is polynomial regression. Polynomial regression is a form of regression analysis in which the relationship between the independent variable *x* and the dependent variable *y* is modelled as an *n*th degree polynomial in *x* [7]. The reason for choosing this method is because my model takes in two features, and based on the result of previous method, perhaps simple linear model is not enough for this dataset. Furthermore, a broad range of functions can be fit under polynomial regression, so I apply this model to my dataset with the list of degrees d = {2, 3, 4, 5, 7, 10} to find out the best fit degree.

According to the book “Machine Learning: The Basics”, since polynomial is also a linear predictor, the quality of polynomial regression is measured by the squared loss error[5]. So I would still use the abovementioned mean squared error as my loss function for the second method, which appears as:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

## 4 Results

Of the two tables below, the upper one (1) contains the training and validation error of linear regression, while the lower one (2) is the training and validation errors of polynomial regression in the order of degrees d = {2, 3, 4, 5, 7, 10} respectively.

| Training error | Validation error |
|---|---|
| 202.32 | 194.75 |

(1)

| Degree | 2 | 3 | 4 | 5 | 7 | 10 |
|---|---|---|---|---|---|---|
| Training error | 200.99 | 199.27 | 197.63 | 196.91 | 194.45 | 191.33 |
| Validation error | 191.17 | 189.94 | 191.23 | 194.79 | 194.63 | 1208.57 |

(2)

Looking at the results, it can be easily seen that both the training errors and validation errors are very large. For most of them, the validation error is smaller than the training error, which implies that the models may be underfitting. Based on the large number results, it also indicates that the models used in the project are not the best ones to find the relationship between my features and labels, perhaps I would need a more complicated model to calculate their correlation.

However, to compare between the models that I used, until the degree of 7, the polynomial regression shows better fit for the dataset than the linear regression, because both the training errors and validation errors of polynomial are smaller than those of linear. So polynomial regression with small degrees ($d \leq 7$) is more suitable for the described featureas and label.

Among the polynomial regression models with different degrees, the one with degree 3 proves to have the best performance because it has the smallest validation error according to the Machine Learning: The Basics book [5]. Although the error shows no evidence of effectiveness of the regression, it it currently the best to be found among the models in my project.

With polynomial regression of degree 3 to be the chosen model, I come to the final test error is 230.61, which turns out to be the smallest test error. The test set has been mentioned before in the section Dataset, which is 12,5% of the original dataset.

## 5 Conclusion

After going through two different models, I arrive at the polynomial regression at degree 3 which has the test error of 230.61, with considerably large training error and validation error (199.27 and 189.94 respectively). It implies that the problem does not really meet the condition since the errors are very large and the gaps between training error and validation error are not small either, so there should be much room for improvement.

In order to improve the method, there are several ways listed below.

Data visualization could have been used to gain a rough insight into the dataset, and it could help to choose a more suitable method than polynomial regression.

Polynomial regression is a simple approach and it is very sensible to outliers. And since weather is an unstable aspect, there might have been some outliers in the dataset, which make polynomial regression less effective.

In conclusion, there is much space for this problem to be developed. When the problem is solved satisfactorily, it would become very handy for predicting the weather without complicated tools.

## 6 References

[1] https://en.ilmatieteenlaitos.fi/download-observations

[2]https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html

[3]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[4] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

[5] https://github.com/alexjungaalto/MachineLearningTheBasics/blob/master/MLBasicsBook.pdf

[6] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html

[7] https://en.wikipedia.org/wiki/Polynomial_regression

## 7 Appendices

- Jupyter notebook (Code for stage 3): Pages below

# Code2

March 29, 2022

[2]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
```

[3]:
```
master_df = pd.DataFrame()
master_df = master_df.append(pd.read_csv('2007-2008.csv'))
master_df = master_df.append(pd.read_csv('2008-2009.csv'))
master_df = master_df.append(pd.read_csv('2009-2010.csv'))
master_df = master_df.append(pd.read_csv('2010-2011.csv'))
master_df = master_df.append(pd.read_csv('2011-2012.csv'))
master_df = master_df.append(pd.read_csv('2012-2013.csv'))
master_df = master_df.append(pd.read_csv('2013-2014.csv'))
master_df = master_df.append(pd.read_csv('2014-2015.csv'))
master_df = master_df.append(pd.read_csv('2015-2016.csv'))
master_df = master_df.append(pd.read_csv('2016-2017.csv'))
master_df = master_df.append(pd.read_csv('2017-2018.csv'))
master_df = master_df.append(pd.read_csv('2018-2019.csv'))
master_df = master_df.append(pd.read_csv('2019-2020.csv'))
master_df = master_df.append(pd.read_csv('2020-2021.csv'))
master_df
```

[3]:
```
     Year   m   d   Time Time zone  Precipitation amount (mm)  \
0    2007  12   1  00:00       UTC                        8.1
1    2007  12   2  00:00       UTC                       10.2
2    2007  12   3  00:00       UTC                       15.7
3    2007  12   4  00:00       UTC                        4.0
4    2007  12   5  00:00       UTC                        3.8
..    …   ..  ..    …         …                         …
116  2021   3  27  00:00       UTC                       -1.0
117  2021   3  28  00:00       UTC                        0.5
118  2021   3  29  00:00       UTC                        5.9
119  2021   3  30  00:00       UTC                        0.3
120  2021   3  31  00:00       UTC                       -1.0
```

```
     Snow depth (cm)  Minimum temperature (degC)
0                1.0                        -3.9
1                8.0                        -3.5
2                4.0                         0.5
3                4.0                         0.0
4                3.0                        -1.3
..               …                         …
116              7.0                        -0.8
117              5.0                         3.0
118             -1.0                         2.5
119             -1.0                         3.8
120             -1.0                         1.9

[1698 rows x 8 columns]
```

[4]:
```
master_df = master_df.dropna(axis=0)

data = master_df.assign(Date = master_df["Year"].astype(str)+'-'+master_df["m"].
 ↪astype(str)+'-'+master_df["d"].astype(str))
data = data.drop(['Year','m','d','Time zone'],axis=1)
data = data[['Date','Time','Precipitation amount (mm)','Snow depth␣
 ↪(cm)','Minimum temperature (degC)']]

imp1 = SimpleImputer(missing_values=-1.0, strategy='mean')
imp2 = SimpleImputer(missing_values=-1.0, strategy='mean')
imp1.fit(data.iloc[:,[2]])
imp_pre = imp1.transform(data.iloc[:,[2]])
imp2.fit(data.iloc[:, [3]])
imp_snow = imp2.transform(data.iloc[:, [3]])

data['Precipitation amount (mm)'] = imp_pre
data['Snow depth (cm)'] = imp_snow
data
```

[4]:
```
          Date   Time  Precipitation amount (mm)  Snow depth (cm)  \
0    2007-12-1  00:00                   8.100000         1.000000
1    2007-12-2  00:00                  10.200000         8.000000
2    2007-12-3  00:00                  15.700000         4.000000
3    2007-12-4  00:00                   4.000000         4.000000
4    2007-12-5  00:00                   3.800000         3.000000
..         …     …                        …                …
116  2021-3-27  00:00                   2.984278         7.000000
117  2021-3-28  00:00                   0.500000         5.000000
118  2021-3-29  00:00                   5.900000        22.872521
119  2021-3-30  00:00                   0.300000        22.872521
120  2021-3-31  00:00                   2.984278        22.872521
```

```
     Minimum temperature (degC)
0                          -3.9
1                          -3.5
2                           0.5
3                           0.0
4                          -1.3
..                          …
116                        -0.8
117                         3.0
118                         2.5
119                         3.8
120                         1.9

[1688 rows x 5 columns]
```

[5]:
```
features_1 = data["Precipitation amount (mm)"].to_numpy().reshape(-1,1)
features_2 = data["Minimum temperature (degC)"].to_numpy().reshape(-1,1)
features = np.column_stack((features_1,features_2))
labels = data["Snow depth (cm)"].to_numpy()

X = np.array(features)
y = np.array(labels)

X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size=0.5,␣
 ↪random_state=42)
```

[6]:
```
regr = LinearRegression()

regr.fit(X_train, y_train)

y_pred_train = regr.predict(X_train)
tr_error = mean_squared_error(y_train, y_pred_train)

y_pred_val = regr.predict(X_val)
val_error = mean_squared_error(y_val, y_pred_val)

y_pred_test = regr.predict(X_test)
test_error = mean_squared_error(y_test, y_pred_test)

print('The training error is: ', tr_error)
print('The validation error is: ', val_error)
print('The test error is:', test_error)
```

```
The training error is:  202.31961750074944
The validation error is:  194.7504941263769
The test error is: 229.5996901359503
```

[12]:
```python
degrees = [2, 3, 4, 5, 7, 10]
tr_errors, val_errors = [], []

for i, degree in enumerate(degrees):

    lin_regr = LinearRegression(fit_intercept=False)
    poly = PolynomialFeatures(degree=degree)
    X_train_poly = poly.fit_transform(X_train)
    lin_regr.fit(X_train_poly, y_train)

    y_pred_train = lin_regr.predict(X_train_poly)
    tr_error = mean_squared_error(y_train, y_pred_train)
    X_val_poly = poly.fit_transform(X_val)
    y_pred_val = lin_regr.predict(X_val_poly)
    val_error = mean_squared_error(y_val, y_pred_val)

    tr_errors.append(tr_error)
    val_errors.append(val_error)

print(tr_errors)
print(val_errors)

poly = PolynomialFeatures(degree=3)
X_train_poly = poly.fit_transform(X_train)
lin_regr.fit(X_train_poly, y_train)
X_test_poly = poly.fit_transform(X_test)
y_pred_test = lin_regr.predict(X_test_poly)
test_error = mean_squared_error(y_test, y_pred_test)

print("The test error is: ",test_error)
```

```
[200.99254250460658, 199.26862298671614, 197.6315879558421, 196.91132207795778,
194.44896978441366, 191.327040799731]
[191.1662239177094, 189.93566218333322, 191.2260694027094, 194.79125935197322,
194.62865136508336, 1208.5727118412037]
The test error is:  230.60669170098907
```