Predicting sea ice extent on the Northern Hemisphere with different regression methods

1. Introduction

The cryosphere is the frozen part of the Earth which includes sea ice, glaciers, ice sheets on Greenland and Antarctica and more. Snow and ice are an important part of the global climate system. The cryosphere protects Earth from getting too warm by acting like a highly reflective shield. Therefore, the extent of the cryosphere, and furthermore the extent of sea ice, is directly linked to global warming and climate change which are some of the biggest worries of todays' world.

The main goal here is to predict the changes in the sea ice extent during the year. Specifically sea ice is very much interlinked with the current season of the area so there will be variance in the extent depending on the time of the year. First, at section 2, we will look at the problem formulation where data points, features and labels will be stated. Second, at section 3, we will proceed to the machine learning methods used in this report. At section 4, we will look at the results given by these methods and lastly, section 5 will conclude this report. References can be found at section 6 and the appendices at section 7.

2. Problem formulation

In this project I will be using machine learning methods to predict the extent of sea ice on Earth's Northern Hemisphere. The data that is used is daily sea ice extent data that can be found on Kaggle.com[1]. The data has been conducted based on downloadable data from National Snow & Ice Data Center's website nsidc.org[2]. This data that I am going to use has daily markings of the sea ice extent from the year 2014 to the year 2021. The data also includes sea specified ice extents such as East Siberian Sea but in this case only the total sea ice extent of the whole Northern Hemisphere will be used.

The data points are the extent of sea ice on the Northern Hemisphere taken at certain days. The dates are in the following form: first is the digits of the year and after that digits that indicate the number of the day starting from the beginning of the year are added. So for example the format for the date 3.1.2015 would be 2015003. Only the day of the year (in this example day 3) will be the feature of the data point. The labels will be the extent of the sea ice for that day.

3. Methods

3.1 Dataset

For this project, data points from the year 2019 were selected. So altogether there are 365 data points, one data point for each day of the year 2019. In the original raw data the dates are in the format described above in section 2 (Problem formulation). This format was modified to only range from 1 to 365 (d = 1,2,...,365) so that 1 indicates the first day of the year and 365 the last day of the year. The column with the ice extent of the Northern Hemisphere was selected for the feature because we are looking at the total amount of sea ice. For the training and validation sets a ratio of 1:3 is used so that the training set is 66% and the validation set is 33%. With a larger validation set the validation error grows larger and the model has a tendency of overfitting. A set of polynomial functions with ranging degree (i = 3,4,...,10) are fitted to the training set.

The sea ice extent increases and decreases throughout the year, depending on the season. Therefore, classification models will not suit this problem. Instead, methods such as polynomial, decision tree and ridge regression are a better fit for this problem. Each one of these will be briefly discussed in the next few subcategories and the outcome of each method will be shown in the fourth section, Results.

3.1. Polynomial Regression

First method to be used is polynomial regression. Polynomial regression is a good fit for a wide range of curvatures. [3] The sea ice extent is highly dependent on the day of the year and changes quite rapidly. Hence, a polynomial of a higher degree can provide us a good fit. What is to be considered is that the polynomial regression is very sensitive for outliers and the presence of a couple of outliers can badly affect the outcome. In relation to this, MSE (mean square error) loss function is used. The MSE loss function gives us an advantage in ensuring that the trained model has no outlier predictions with huge errors.

The MSE loss is calculated for both training and validation sets. The MSE loss function can be defined as following [4]:

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (y - \hat{y}_i)^2$$

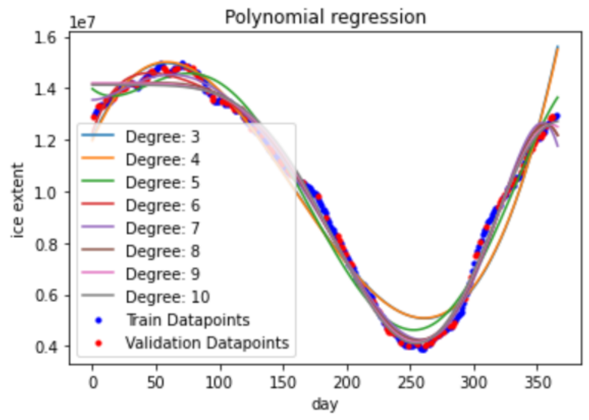
Here $\hat{y}$ is the predicted value and y is the real lablel value. The error losses of both training and validation sets are plotted in order to study the fit of the model for each degree.

3.2. Ridge Regression

Secondly, ridge regression will be used. The decision to use ridge regression instead of for example lasso regression was because lasso regression is a better fit for more complex datasets. Lasso regression also helps with feature selection, which in this case is not necessary since we only have one feature [5]. Ridge regression does parameter shrinkage and also works better when there are multiple significant parameters of approximately the same value. Ridge regression uses the L2 regularization technique, that is it adds a squared magnitude of coefficient as a penalty term.

4. Results

4.1. Polynomial Regression and Ridge Regression



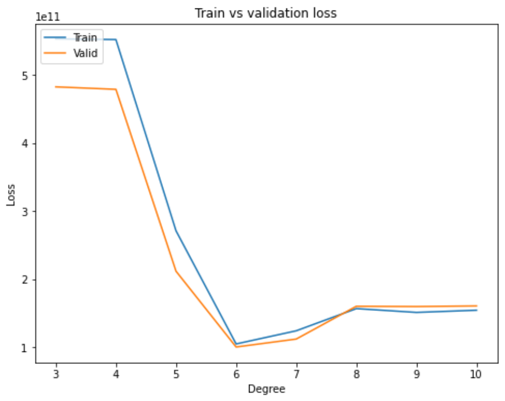
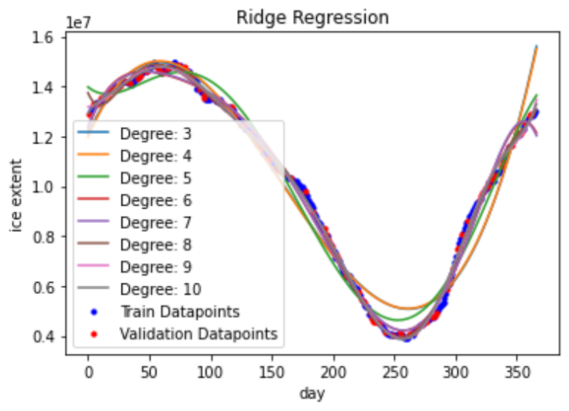


Figure 1. (left) The Polynomial regression fit for degrees 3 to 10.
Figure 2. (right) Training and validation loss for polynomial regression fit of degrees 3 to 10.

Here we can see that the polynomial regression with the degree six has the lowest value for both training and validation loss. As the degree increases onwards from the degree seven we can see the effect of overfitting. Validation error grows larger than the training error. As well as for the smaller degrees such as three and four, we can see the effect of underfitting. The training error is much larger than the validation error.



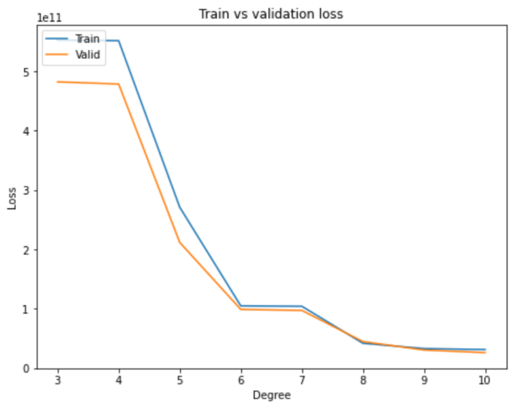


Figure 5. (left) Ridge Regression for polynomials with degree ranging from 3 to 10
Figure 6. (right) Train and validation errors for corresponding ridge regressions.

Here we can see that the train and validation errors decrease as the degree of the polynomial continues to increase. The validation error exceeds the train error when the polynomial degree grows greater than eight. The best results, ergo the best fit, is provided by the ridge regression of polynomial degree seven.

4.2 The Final Model

The choice to use the methods listed before was based on comparison of different methods and their performance. It was clear that polynomial regression and ridge regression gave the best results. But from these, the ridge regression fit of degree seven gave the best performance.

Training and validation errors of both methods' best fit:

Method	Polynomial degree	Train error	Validation error
Polynomial Regression	6	104819105753.2971	100347964984.41884
Ridge Regression	7	103958310024.2501	97012384893.39749

Here we can see that indeed the ridge regression of polynomial degree seven has the smallest train and validation errors where the validation error does not exceed the train error.

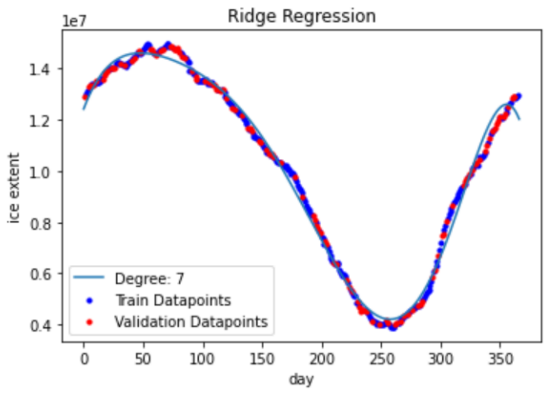


Figure 7. The final, most effective model fit: ridge regression fit of polynomial degree seven.

5. Conclusion

The most effectively fitting model for predicting sea ice extent was found to be the ridge regression. There is not too much of a difference between the results of the polynomial regression compared to the ridge regression. Both seem to underfit with too low degrees (3 and 4) and overfit with too high degrees (8, 9 and 10). Both of these models work great with rapidly changing data points and fit the curvatures. However, there is always a risk of overfitting and therefore the MSE (mean squared error) was used.

The accuracy of the final model could be improved by using data points from multiple years. This model was made based on the observations of the year 2019 and therefore is highly based on those measurements. To gain a more precise model for future ice extent predicting one might use more data.

6. References

[1] https://www.kaggle.com/lsind18/ims-daily-northern-hemisphere-snow-and-ice
[2] https://nsidc.org/data/masie
[3]https://towardsdatascience.com/introduction-to-linear-regression-and-polynomial-regression-f8adc96f31cb
[4]
https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-squared-error
[5]
https://towardsdatascience.com/ridge-and-lasso-regression-a-complete-guide-with-python-scikit-learn-e20e34bcbf0b

7. Appendices

The code is listed below.

(The data: https://www.kaggle.com/lsind18/ims-daily-northern-hemisphere-snow-and-ice)

MLproject

March 31, 2022

1 Machine Learning project

1.1 Predicting sea ice extent on the Northern Hemisphere

First, polynomial regression method is used to create a hypothesis for the ice extent.

[1]:
```
%config Completer.use_jedi = False  # enable code auto-completion
import numpy as np    # library for numerical computations (vectors, matrices,␣
↪tensors)
import pandas as pd    # library for data manipulation and analysis
import matplotlib.pyplot as plt    # library providing tools for plotting data
from sklearn.preprocessing import PolynomialFeatures    # function to generate␣
↪polynomial and interaction features
from sklearn.linear_model import LinearRegression    # classes providing Linear␣
↪Regression with ordinary squared error loss
from sklearn.linear_model import Ridge              # class providing Ridge␣
↪regression
from sklearn.metrics import mean_squared_error    # function to calculate mean␣
↪squared error
from sklearn.model_selection import train_test_split   # function to train test␣
↪and split data
```

[2]:
```
## Lets read the raw data in 'masie_4km_allyears_extent_sqkm.csv' and store it␣
↪in variable 'RawData'

RawData = pd.read_csv('masie_4km_allyears_extent_sqkm.csv')

## test to print first 5 columns of data
RawData.tail(5)
```

[2]:
```
      yyyyddd  Northern_Hemisphere  Beaufort_Sea  Chukchi_Sea  \
5445  2021010          13239856.28     1070689.2    966006.16
5446  2021011          13330573.75     1070689.2    966006.16
5447  2021012          13519114.70     1070689.2    966006.16
5448  2021013          13570166.17     1070689.2    966006.16
5449  2021014          13599214.40     1070689.2    966006.16

      East_Siberian_Sea  Laptev_Sea   Kara_Sea  Barents_Sea  Greenland_Sea  \
5445         1087119.86   897827.01  837196.81    376514.30      656234.73
5446         1087119.86   897827.01  857797.21    393368.17      645859.62
5447         1087119.86   897827.01  869798.48    442932.04      656394.09
5448         1087119.86   897827.01  838589.77    454291.09      657007.55
5449         1087119.86   897827.01  841300.29    467377.63      638469.08

      Baffin_Bay_Gulf_of_St._Lawrence  Canadian_Archipelago  Hudson_Bay  \
5445                        958212.72             854597.27  1260470.92
5446                        966717.47             854597.27  1260470.92
5447                       1010092.02             854597.27  1260470.92
5448                       1040976.57             854597.27  1260470.92
5449                       1052875.74             854597.27  1260470.92

      Central_Arctic  Bering_Sea  Baltic_Sea  Sea_of_Okhotsk  Yellow_Sea  \
5445      3165806.68   392892.34    11933.33       668812.50    26054.97
5446      3158716.27   414705.61    12949.03       708207.50    26054.97
5447      3156891.89   436788.77    13741.72       754703.67    31574.13
5448      3159836.55   467534.28    13741.72       766276.76    25713.99
5449      3171497.06   478783.84    20955.89       754390.89    26708.67

      Cook_Inlet
5445     7686.37
5446     7686.37
5447     7686.37
5448     7686.37
5449     8343.77
```

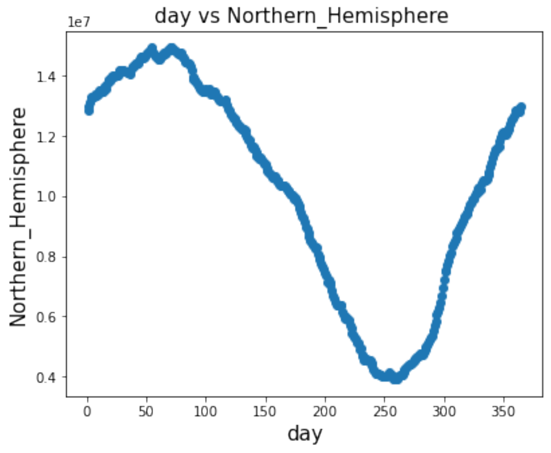
[3]:
```
## Filtering the data: taking only the date ('yyyyddd') and␣
↪'Northern_Hemisphere' i.e. drop everything else

RawData['yyyyddd'] -= 2019000 ## subtracting 2019000 in order to get only the␣
↪daysof the year 2019

newdata = RawData.drop(['Beaufort_Sea', 'Chukchi_Sea',
       'East_Siberian_Sea', 'Laptev_Sea', 'Kara_Sea', 'Barents_Sea',
       'Greenland_Sea', 'Baffin_Bay_Gulf_of_St._Lawrence',
       'Canadian_Archipelago', 'Hudson_Bay', 'Central_Arctic', 'Bering_Sea',
       'Baltic_Sea', 'Sea_of_Okhotsk', 'Yellow_Sea', 'Cook_Inlet'],axis=1)

## change first column name
newdata.columns = ['day', 'Northern_Hemisphere']

## selecting datapoints from year 2019 (1-365)

mask = newdata.day > 0
newdata2 = newdata[mask]
mask2 = newdata2.day < 366
```

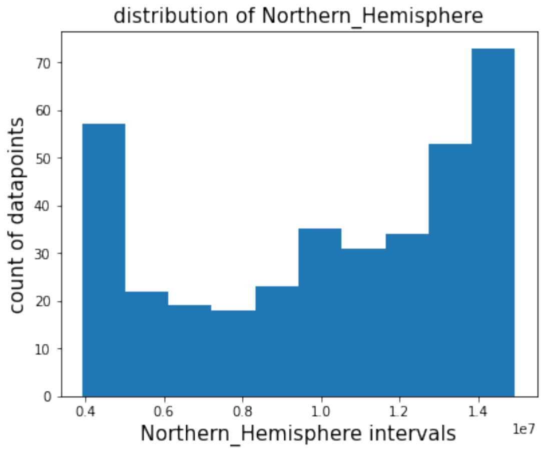
```
df = newdata2[mask2]

df.head(5) ## checking that the data is in the wanted format
```

```
[3]:       day  Northern_Hemisphere
     4705    1          12871795.41
     4706    2          13012673.16
     4707    3          13106826.22
     4708    4          13291005.47
     4709    5          13307331.11
```

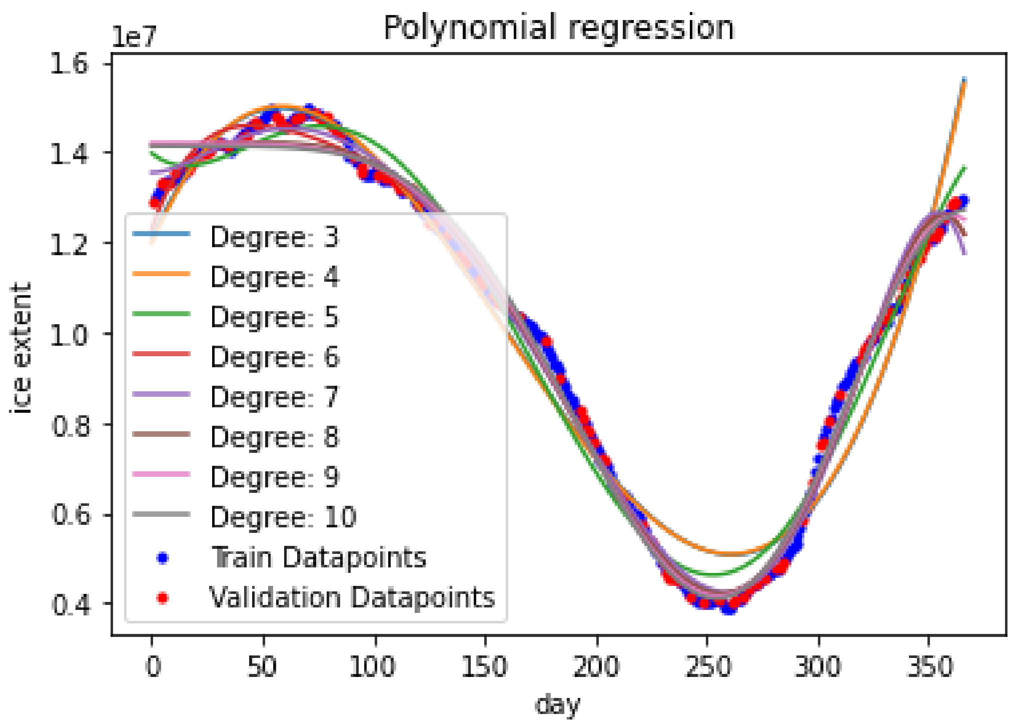
```
[4]: ## Selecting features and labels

     X = df["day"]
     y = df["Northern_Hemisphere"]

     X.to_numpy()
     y.to_numpy()

     X = np.array(X).reshape(-1,1)
     y = np.array(y)
```

```
[5]: # Visualizing data
     fig, axes = plt.subplots(1, 2, figsize=(14,5)) # create a figure with two axes␣
      ↪(1 row,2 columns) on it
     axes[0].scatter(df['day'],df['Northern_Hemisphere']) # plot a scatter plot on␣
      ↪axes[0]
     axes[0].set_xlabel("day",size=15)
     axes[0].set_ylabel("Northern_Hemisphere",size=15)
     axes[0].set_title("day vs Northern_Hemisphere ",size=15)

     axes[1].hist(df['Northern_Hemisphere']) # plot a hist plot to show the␣
      ↪distribution of Northern_Hemisphere
     axes[1].set_title('distribution of Northern_Hemisphere',size=15)
     axes[1].set_ylabel("count of datapoints",size=15)
     axes[1].set_xlabel("Northern_Hemisphere intervals",size=15)
     plt.show()
```

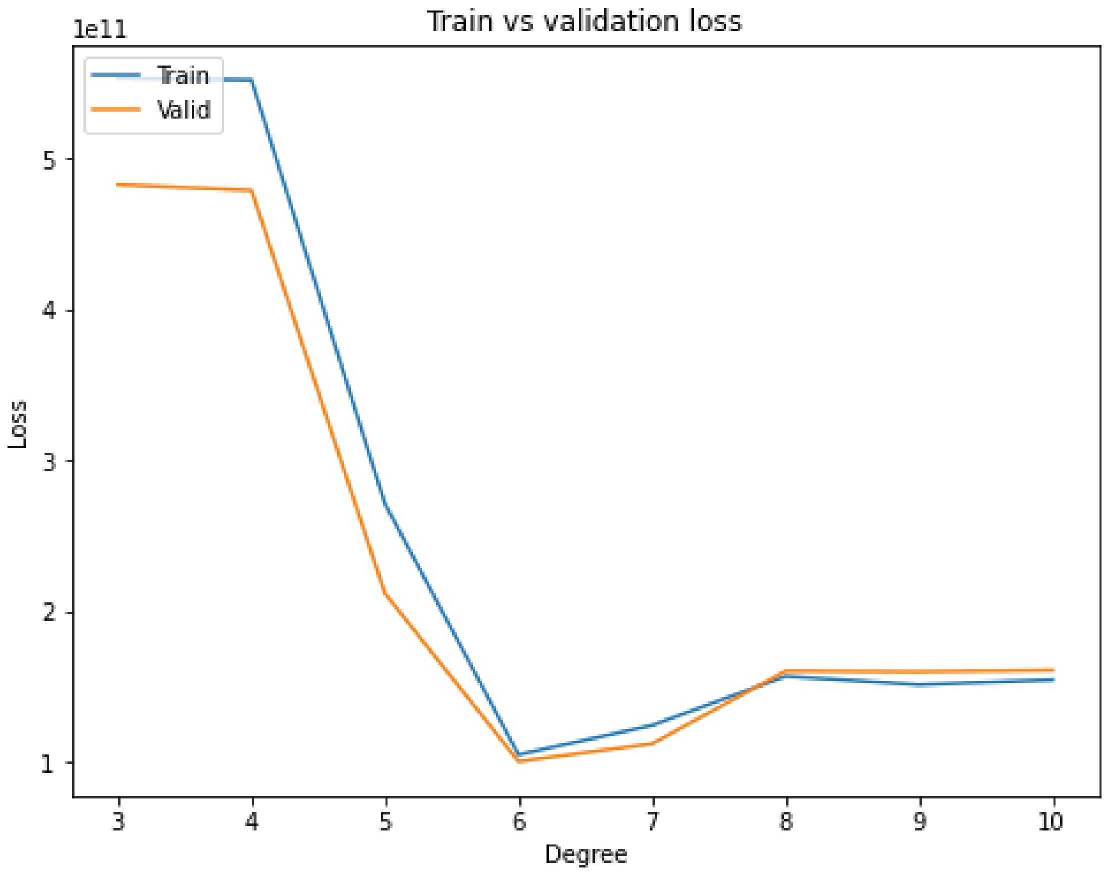




[6]:
```
# Fitting a Polynomial Regression model


# Split the dataset into a training set and a validation set
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.33,␣
 ↪random_state=42)

## define a list of values for polynomial degrees
degrees = [3, 4, 5, 6, 7, 8, 9, 10]

# we will use this variable to store the resulting training errors for each␣
 ↪polynomial degree
tr_errors = []
val_errors = []

for i in range(len(degrees)):    # use for-loop to fit polynomial regression␣
 ↪models with different degrees

    poly = PolynomialFeatures(degrees[i])
    lin_regr = LinearRegression()

    X_train_poly = poly.fit_transform(X_train)    # fit and transform the raw␣
 ↪features
    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.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)
    X_fit = np.linspace(0, 366, 1000)     # generate samples
    degree_label = "Degree: " + str(degrees[i])
    plt.plot(X_fit, lin_regr.predict(poly.transform(X_fit.reshape(-1, 1))),␣
 ↪label=degree_label)     # plot the polynomial regression model

plt.scatter(X_train, y_train, color="b", s=10, label="Train Datapoints")    #␣
 ↪plot a scatter plot of y(Northern_Hemisphere) vs. X(day) with color 'blue'␣
 ↪and size '10'
plt.scatter(X_val, y_val, color="r", s=10, label="Validation Datapoints")    #␣
 ↪do the same for validation data with color 'red'
plt.xlabel('day')     # set the label for the x/y-axis
plt.ylabel('ice extent')
plt.legend(loc="best")     # set the location of the legend
plt.title('Polynomial regression')     # set the title
plt.show()     # show the plot
```



[7]:
```
plt.figure(figsize=(8, 6))

plt.plot(degrees, tr_errors, label = 'Train')
plt.plot(degrees, val_errors,label = 'Valid')
plt.legend(loc = 'upper left')
plt.xlabel('Degree')
plt.ylabel('Loss')
plt.title('Train vs validation loss')
plt.show()
```

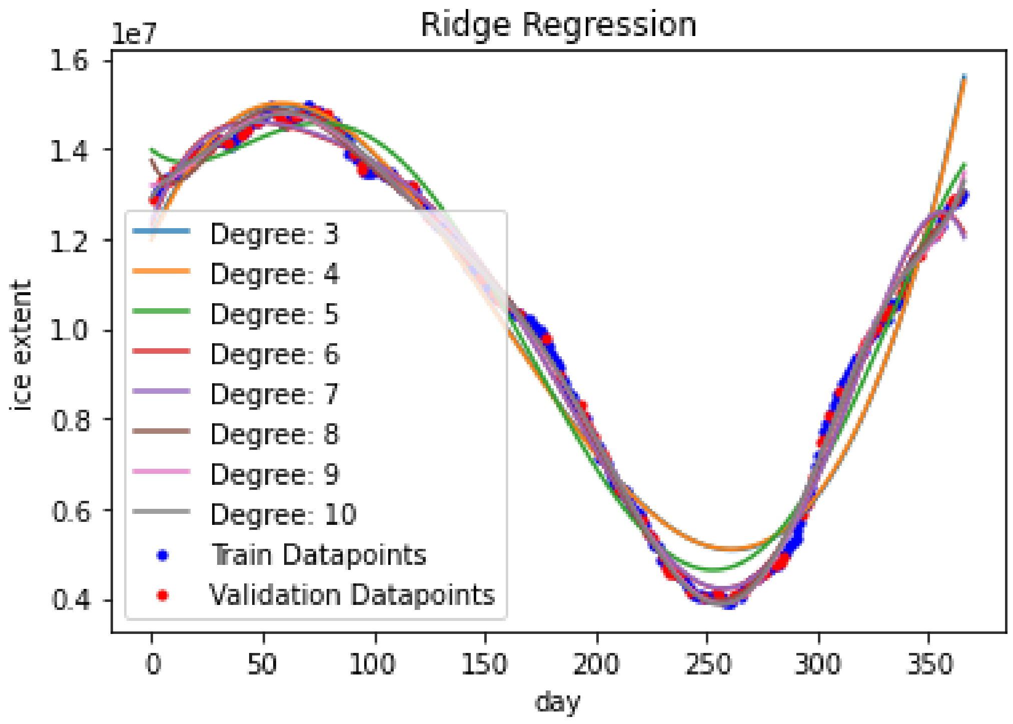


[8]:
```
print("Degree: " + str(degrees[3]))
print("Train: " + str(tr_errors[3]))
print("Validation: " + str(val_errors[3]))
```

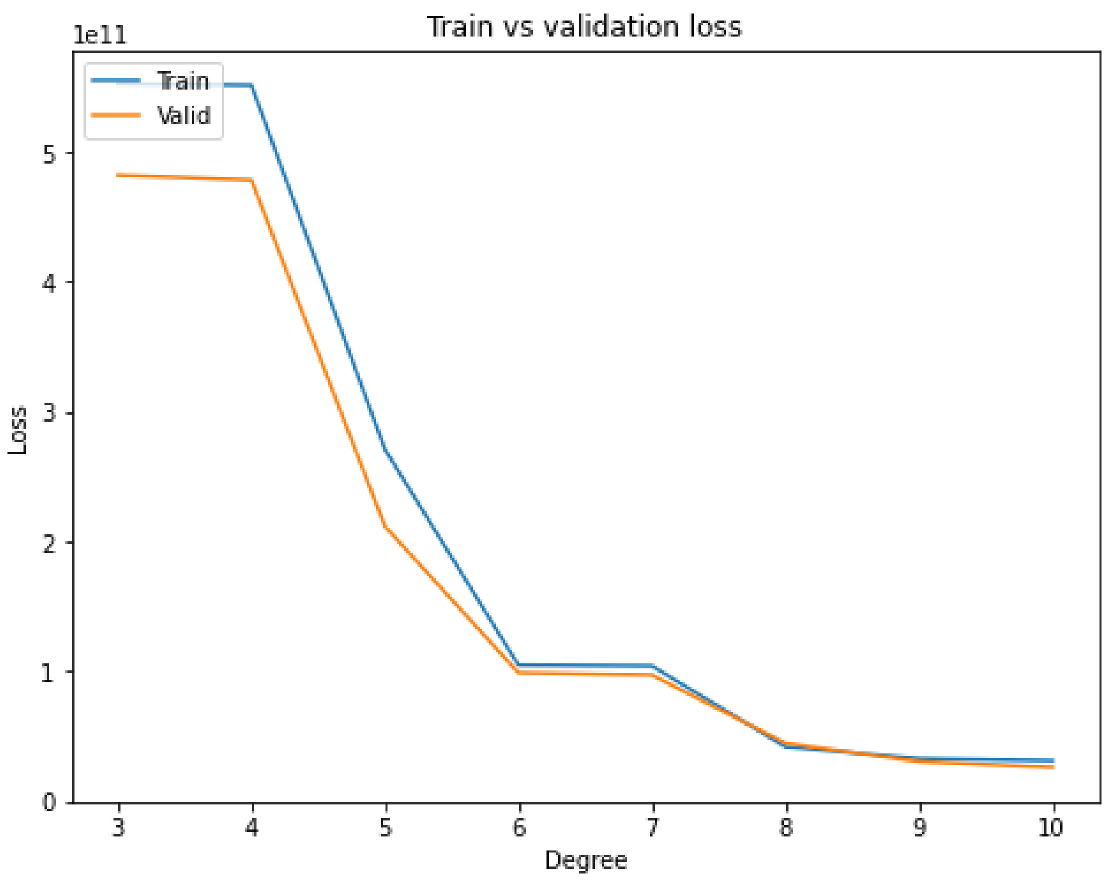
```
Degree: 6
Train: 104819105753.2971
Validation: 100347964984.41884
```

[9]:
```
## Fitting a Ridge Regression model


## define a list of values for polynomial degrees
degrees = [3, 4, 5, 6, 7, 8, 9, 10]
```

```
# we will use this variable to store the resulting training errors for each␣
↪degree
tr_errors = []
val_errors = []

for i in range(len(degrees)):    # use for-loop to fit ridge regression models␣
↪for different polynomial degrees

    poly = PolynomialFeatures(degrees[i])
    tree_regr = Ridge(alpha=0.0, normalize=True)

    X_train_poly = poly.fit_transform(X_train)    # fit and transform the raw␣
↪features
    tree_regr.fit(X_train_poly, y_train)

    y_pred_train = tree_regr.predict(X_train_poly)
    tr_error = mean_squared_error(y_train, y_pred_train)
    X_val_poly = poly.transform(X_val)
    y_pred_val = tree_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)
    X_fit = np.linspace(0, 366, 1000)    # generate samples
    degree_label = "Degree: " + str(degrees[i])
    plt.plot(X_fit, tree_regr.predict(poly.transform(X_fit.reshape(-1, 1))),␣
↪label=degree_label)    # plot the ridge regression model

plt.scatter(X_train, y_train, color="b", s=10, label="Train Datapoints")    #␣
↪plot a scatter plot of y(Northern_Hemisphere) vs. X(day) with color 'blue'␣
↪and size '10'
plt.scatter(X_val, y_val, color="r", s=10, label="Validation Datapoints")    #␣
↪do the same for validation data with color 'red'
plt.xlabel('day')    # set the label for the x/y-axis
plt.ylabel('ice extent')
plt.legend(loc="best")    # set the location of the legend
plt.title('Ridge Regression')    # set the title
plt.show()    # show the plot
```



[10]:
```
plt.figure(figsize=(8, 6))

plt.plot(degrees, tr_errors, label = 'Train')
plt.plot(degrees, val_errors,label = 'Valid')
plt.legend(loc = 'upper left')

plt.xlabel('Degree')
plt.ylabel('Loss')
plt.title('Train vs validation loss')
plt.show()
```



```
[11]: print("Degree: " + str(degrees[4]))
      print("Train: " + str(tr_errors[4]))
      print("Validation: " + str(val_errors[4]))
```

```
Degree: 7
Train: 103958310024.2501
Validation: 97012384893.39749
```

```
[ ]:
```