Chair in
Information and Service Systems

UNIVERSITÄT
DES
SAARLANDES

Topic 3: Environment
Determine the CO2 emission of machine learning computation

Lecture	Data Science SoSe 2022
Participants	Abdul Mueed, Muhammad (7009503), muab00001@stud.uni-saarland.de Paulus, Mina (2563573), s8mmpaul@stud.uni-saarland.de Franke, Lars (2566269), s8lsfran@stud.uni-saarland.de Zender, Dennis (2550212), s9dezend@stud.uni-saarland.de
Submission date	22-07-2022
Chair	Univ.-Prof. Dr.-Ing. Wolfgang Maaß Chair in Information and Service Systems, Campus A5 4, 66123 Saarbrücken

Executive Summary

Research shows, that CO2 emissions of machine learning computation are becoming a more and more important factor of environmental pollution, which needs to be faced in the next years. Since the number of processed data is rising exponentially, there is currently no end in sight.

Companies therefore need to become aware of how CO2 emissions caused by machine learning can be contained and reduced. As a first step, it is necessary to be able to determine and predict the CO2 emissions caused by machine learning.

In this paper, we have addressed this issue and developed an approach to determine the amount of CO2 emission. In addition, we give an outlook on further research possibilities in order to further develop the topic of emission reduction. As a data base we use a combination of different data sets of previous work done around the field of CO2 emissions in machine learning and developed a model to determine these emissions.

Contents

List of Abbreviation

Machine Learning	ML

List of Figures

1 Introduction

In a world of growing challenges with the goal of creating newer and better machine learning models, which are more useful than the ones that already exist, the amount of data to be processed is becoming increasingly more. To beat these challenges, more CPUs and especially GPUs need to be used to train the models over a longer period. This leads to ever-growing costs regarding the amount of energy needed to run the GPUs while training the models. Besides rising costs, the needed high energy consumption also causes high CO2 emissions.

Due to these economic and ecological factors, it is important to identify crucial aspects of training models and how it is possible to reduce the amounts of carbon emissions that the training emits and the costs of the training process.
Current research identified a few crucial points to look at while thinking about reducing the carbon footprint of training models: These factors include e.g., “the location of the server used for training and the energy grid that it uses, the length of the training procedure, and even the make and model of hardware on which the training takes place.” (Lacoste et al. 2019, p. 1) Lacoste et al. built an application to approximate this carbon footprint. In order to approximate these emissions, there are applications like the Machine Learning Emissions Calculator, a tool to make it easier to understand the environmental impact of training ML models (Lacoste et al. 2019, p. 1). Other Papers predict that the carbon footprint of machine learning training will plateau within the next years, and then shrink because of the rising costs and the rising awareness of the users and companies regarding the importance of climate chance (Patterson et al. 2022, p. 1-4). In Summary, the current research agrees with the fact, that energy consumption of machine learning will be a bigger problem in the future due to the circumstances, that the usage of machine learning is increasing exponentially (Hao 2019).

Although the problem is not new, current research does show multiple research gaps. The main problem currently is that there is yet no solution or any further research about how to make it easy to predict the energy consumption without knowing the time needed for training the model. Current research only offers the possibility of some kind of simple calculation of the amount of energy used during the training.

1 Data Set

For our service use case we decided to combine data from three different places. The first source is some data from the Machine Learning Emissions Calculator, which is based on the work of Lacoste et al. to quantify the Carbon Emissions of Machine Learning (Lacoste et al. 2019; Lacoste et al. 2019a). The second data set we used for our work is a data set which maps the carbon footprint of electricity worldwide. With this set we can use accurate hourly based data of the energy consumption and CO_2 emissions in more than 50 countries (Corradi et al 2021). The third data set we include is the Experiment impact tracker, “which is meant to be a simple drop-in method to track energy usage, carbon emissions, and compute utilization of your system” (Henderson et al.).

To use the different data in our set, we first updated the yearly emission averages to the 2021 averages, while Henderson et al. used older data sets from 2017 or 2018 (Henderson et al.). In a second step we took the geographic zones of Corradi et al. As Polygons on a (-180,180) x (-90,90) map to create a world map with all the regions we could manage to use data from. The GPU specific information about the energy consumption (watts/h) was collected from Lacoste et al. (Lacoste et al. 2019).

All data we used is available to the public, since all publications are also available on GitHub as open-source projects. However, in general, there is little data available online to further improve this application, as we wished to have some data about training time and CPU usage as well, to predict the emission even before the model was trained. As of now, one can only predict the emission if they already trained the model or a similar one, since otherwise they would not know how long it would take to train it. Therefore, the CO_2 is already emitted.

2 Procedure and Analysis

In the following we will describe our process of creating a prototype for this project, and afterwards we briefly explain the methods used to analyze the model and the results.

2.1 Procedure

We started approaching the problem by addressing the use case of the CO_2 emission pipeline. After doing some research, the main idea of our approach was to not just give the user information about the CO_2 emitted, but instead to find the main reasons behind the amount of CO_2 emitted.

By knowing the reasons for the emissions, we could then try to get some data that would increase the accuracy of the emission calculation.

Lacoste et al. created a calculator, that takes information about the GPU, the hours used to train the machine learning model, if any the cloud provider used, and the region where the model was trained (Lacoste et al 2019a). While all this information has an impact on the CO_2 emission of training a model, it is just an estimation with partial information. By giving the type of GPU used to train the model, the total power usage of the GPU is not exactly known, as it differs depending on the use case and the given circumstances. Furthermore, the CPU might also be used in the process, which is not considered in their calculation. Therefore, Henderson et al. suggested that while might being inaccurate, the calculator created by Lacoste et al. could be used to get an "ahead of time" calculation on CO_2 emissions.

This gave us the idea to differentiate between the CO_2 emission calculation done before a model is trained, and the calculation after a model finished training. While being more inaccurate, we felt that it is more important to get the emission estimation before training the model, because otherwise the CO_2 is already emitted. Therefore, the first approach for our application was to create a CO_2 emission calculator, that does the calculation in the same way Lacoste et al. did in their calculator (Lacoste et al 2019). Since we wanted to use machine learning instead of a simple formular for our application, we calculated the CO_2 emissions for the data with their formular and used the obtained value as the target variable for a simple linear regression model. The data we used for training was all the data that was attached to our project in the kick-off presentation slides I.e., the data Lacoste et al. Used for their calculator. Everything was programmed in a Jupyter notebook (https://jupyter.org/). Parallel, while working on the data preparation and the model we also worked on a user interface, where one can select the GPU, the region, and the provider by clicking on buttons, and getting the amount of CO_2 emitted by a machine learning model trained on this input.

Now we had a simple calculator for the emissions that we wanted to improve on. We looked for some other factors that might contain information contributing to CO_2 emissions and found an interesting paper by Li et al., where they researched the energy efficiency of deep convolutional neural networks with different architectures and deep learning frameworks. Although, it would be a useful addition to our calculator, to be able to select an architecture and framework when working with a deep convolutional neural network, we found the main problem with our topic, which is the data collection. As Schwartz et al. state in their work, “the AI research community has paid relatively little attention to computational efficiency”, and therefore there is very little data available containing information about the power usage of trained models.

When there is no data, we must create the data ourselves, however after working on it for some time, we realized that this is not applicable in our scenario. We took some randomized models from the framework created by Henderson et al. which gave us information about the training time of each model. The architecture of the models was taken from the most used model architectures from Tensorflow (https://www.tensorflow.org/), such that one can then select the model architecture as an input in our calculator. However, the training time was long, and since we needed a lot of data to train on, this was not applicable for the course of this project. Furthermore, the information would be only useful for the same graphic card we used for training. Unfortunately, we had to discard the idea of creating data by creating models we train ourselves and went a step back to the calculator we already created and looked for ways to improve what we already got, since we already had some useful data there.

The data that was given to us contained information about the GPUs, yearly emission averages per world region, and about the emission of cloud providers like Google Cloud Platform (https://cloud.google.com/?hl=de), Microsoft Azure (https://azure.microsoft.com/de-de/) and Amazon Web Services (https://aws.amazon.com/de/).

We decided that we wanted to generalize the calculator to not only cloud providers but also just machine learning training on regular machines at home or in other companies, we extended the given data with a world map (Corradi et al 2021), that splits the world up into meaningful regions, defined by polygons, each region corresponding to a unique power consumption average.

Then, we generated samples across the world map, and took their actual 2021 average emissions, and used this to train a Random Forest model. We also tried several other models, but the Random Forest led to the best results. The model is then able to take a string that contains information about the location, like “Berlin, Germany”, map it to a GPS location using a library called geopy, and accurately predict the CO_2 emission on that location.

Then, we used the simple formula from Lacoste et al. to compute the actual CO_2 emission based on the GPU energy consumption, the training time and the predicted average emission per location. Even though we also trained a linear regression model to predict the emission based on GPU energy consumption, training time and average emission per location, we decided to not do this with machine learning, since the formula is a rather simple one and training a machine learning model for this would, as we learned in this project, result in unnecessary CO_2 emission.

Since we changed the model, we also had to change the application a little bit, because the inputs changed. We removed the buttons for the region and for the provider, and instead one must write the region where they are training the model into a text field. This is not a problem, because with this new model, the information about the region and the provider is redundant.

2.2 Analysis

Initially we tried to organize the impact of CO_2 using the data which we were given. From that we could make an initial map of the world that would display impact of CO_2 for every single spot on the world, leaving only the sea part of the world, and regions that do not contain data for this task.

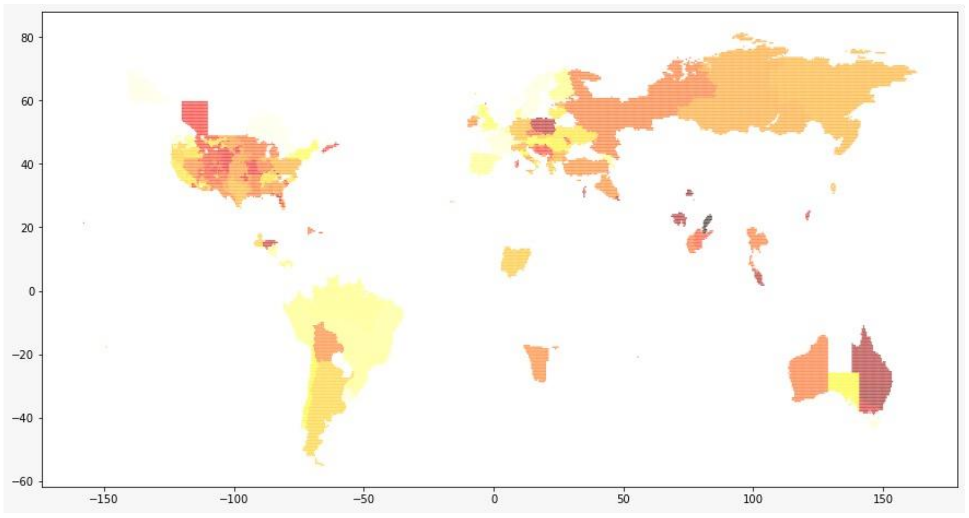

Fig. 1: The CO_2 emission of each region that has data on power providers

As we can see clearly, only the land parts with data are displayed and water areas are hidden, since no CO_2 Emission could arise from there, as there is no power provider with given data in these regions. Furthermore, we used different colour scheme to show the impact of CO_2. Ranging from light colours for countries with low emission rate as compared to dark areas with high CO_2 emissions. This are the actual CO_2 emissions extracted from the data given in form of CO_2.

3 Results and Discussion

In this section we shortly present the results and the key features of our model, before discussing it and giving a short outlook for future work.

3.1 Results

3.1.1 Performance

From all models that were used to solve the task, Random Forest Regression achieved the best R2 score. While the Decision Tree Regression and the Neural Network reached a solid score of slightly above 95%, Random Forest Regression obtained an accuracy of 96.7%. The other regression models, namely SVM Regression and Linear Regression had an accuracy of below 50%. With same procedure as above, we mapped our CO_2 impact prediction with Random Forest Regression of each part of world (leaving the sea and the regions not containing any data) into a graphical map. This map could be used to understand critical points of CO_2 emissions. Since the accuracy of the model was really high it was already expected to look nearly identical to the map shown in the analysis chapter.

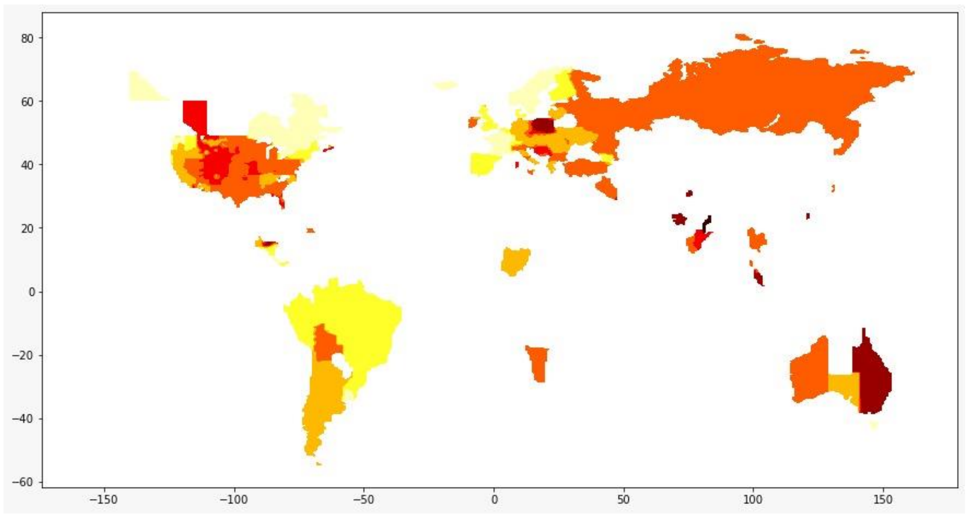

Fig 2: The predicted CO_2 emission of each region that has data on power providers

3.1.2 Key Features of our Model

We have three main key features in our model that overpowers the existing models in research.

1) Our model provides prediction of CO_2 emission, for every location on the world where data could be found. Looking on existing models in research, we could only find models that predicts the CO_2 emission for locations where cloud providers have their servers, limiting the target audience to users of cloud providers only. Our model overcomes this problem and using the location provided by the user, it can provide the CO_2 emission for every individual location of the world, that we have basic average data from.
2) Using our model, we could develop a world map for CO_2 emission, highlighting each critical areas and locations.
3) The accuracy of the model is very high. This is due to using a lot of training points for training the Random Forest Regression. Around 50000 points on the world map were

used for training. These results are promising and could be used in research and further relevant areas, for studying the impact of CO_2 very precisely and having very mild error rate.

3.2 Discussion

Although, the problem given in this task is a regression problem because the value of the impact is continuous, classification-based models like Random Forests or Decision Trees were expected to perform well, as the impact of the regions could be seen as a class label, such that the problem is equivalent to a multi-class classification problem. Furthermore, when plotting the real data as shown below, one can clearly see that it was expected for the Linear Regression Model and the SVM Regression too have poor performance, since the different impacts can hardly be divided into their corresponding regions by using linear functions.

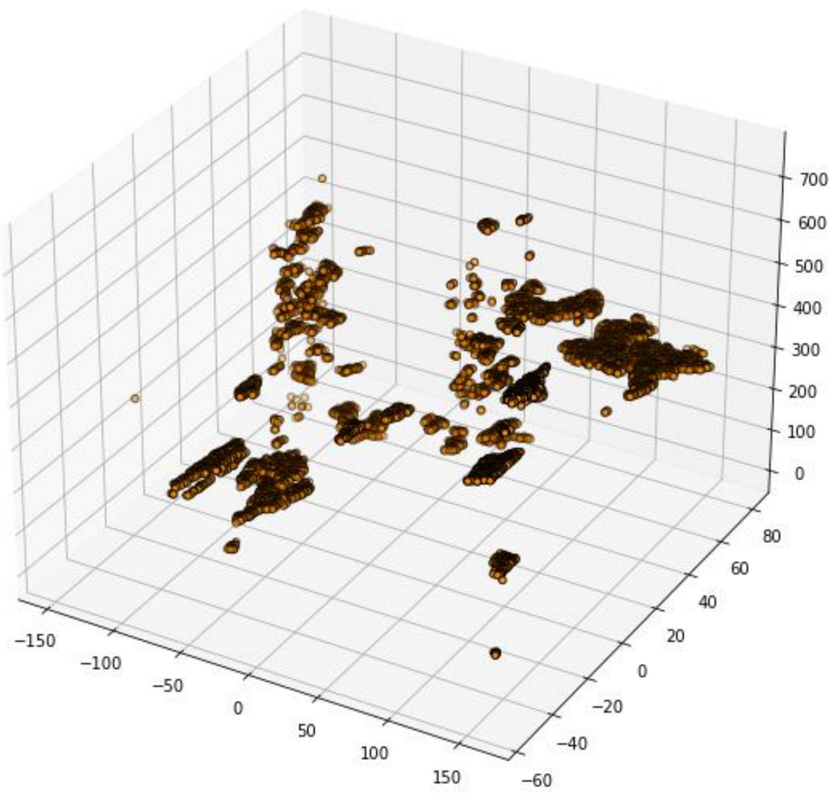

Fig 3: The impact of all training points

The model has a good performance, nevertheless the region where a person trains the model is only one part of many that impacts the CO_2 emissions. However, with the current model at least a pre-training estimation of the emissions is given, that could already indicate where to work on to reduce the emissions.

3.3 Outlook

With the exponential growing speed of machine learning models, it will get more and more relevant to not only build better models, but also to build more efficient models. If AI researchers paid more attention to efficiency, they could not just reduce the CO_2 emissions emitted by their models, but they could also strongly contribute to the creation of better machine learning models that predict the CO_2 emissions of their models. Especially since there is little to none open source data that documents the CPU and GPU power usage when training the model, as well as the time used to train the model. With more data out there, one could integrate all this data into the calculator which should drastically increase the accuracy of the CO_2 emission prediction.

Besides integrating the said information into the calculator, information about the model itself could also be used to increase the performance. For example, as already mentioned in the procedure part, the architecture of different popularly used neural networks and neural network types could be used as an input to get a better complexity estimation of the trained model. Besides for neural networks, this will also work for other machine learning models, but instead of the architecture of common models like SVMs, the parameters given to the model could be used, as they usually do not have an architecture like a neural network does.

Lastly, the training size plays a big role in the time needed to train a model. By giving information to reduce the training size optimally, the training time, and thus the CO_2 emissions could be further reduced.

Bibliography

Alsop, T. (2021) Market revenue of artificial intelligence chips from 2020 to 2026. URL https://www.statista.com/statistics/1283358/artificial-intelligence-chip-market-size/

Biewald, Lukas (2019). Deep Learning and Carbon Emissions. URL= https://towardsdatascience.com/deep-learning-and-carbon-emissions-79723d5bc86e.

Corradi, O., Qvist, F., Nedergaard, M., Segonne, P. (2021). Electricity Maps. URL= https://github.com/electricitymap

Deloitte Global Estimates (2017). Hitting the accelerator: the next generation of machine-learning chips. URL= https://www2.deloitte.com/content/dam/Deloitte/global/Images/infographics/technologymediatelecommunications/gx-deloitte-tmt-2018-nextgen-machine-learning-report.pdf

Hao, K. (2019). Training a single AI model can emit as much carbon as five cars in their lifetimes. MIT Technology Review. URL = https://www.technologyreview.com/2019/06/06/239031/training-a-single-ai-model-can-emit-as-much-carbon-as-five-cars-in-their-lifetimes

Henderson, P., Hu, J., Romoff, J., Brunskill, E., Jurafsky, D., & Pineau, J. (2020). Towards the systematic reporting of the energy and carbon footprints of machine learning. Journal of Machine Learning Research, 21(248), 1-43.

Henderson, P., Hu, J., Derczynski, L., Teixeira, R. Experiment-impact-tracker. URL= https://github.com/Breakend/experiment-impact-tracker.

Lacoste, A., Luccioni, A., Schmidt, V., Dandres, T. (2019). Quantifying the Carbon Emissions of Machine Learning. arXiv:1910.09700.

Lacoste, A., Luccioni, A., Schmidt, V., Dandres, T. (2019a). Dataset Machine Learning Emissions Calculator. URL= https://github.com/mlco2/impact

Li, D., Chen, X., Becchi, M., & Zong, Z. (2016, October). Evaluating the energy efficiency of deep convolutional neural networks on CPUs and GPUs. In 2016 IEEE international conferences on big data and cloud computing (BDCloud), social computing and networking (SocialCom), sustainable computing and communications (SustainCom)(BDCloud-SocialCom-SustainCom) (pp. 477-484). IEEE.

Patterson, D., Gonzalez, J., Hölzle, U., Le, Q., Liang, C., Munguia, L.-M., Rothchild, D., So, D., Texier, M., Dean, J. (2022). The Carbon Footprint of Machine Learning Training Will Plateau, Then Shrink. arXiv: arXiv:2204.05149.

Schwartz, R., Dodge, J., Smith, N. A., & Etzioni, O. (2020). Green ai. Communications of the ACM, 63(12), 54-63.

Appendix

In the following we present first some snippet from our code, and afterwards an explenation, what the snippet is doing. Each snippet does not represent one cell of the .ipynb file we submitted, but instead we split the cells up into smaller groups that are easier to summarize. Note that we took out the prints and comments for this documentation to make it better readable for the reader.

```python
import csv
import matplotlib.pyplot as plt
import pandas as pd
from geopy.geocoders import Nominatim
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import svm
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from sklearn import tree
from sklearn.neural_network import MLPRegressor
import numpy as np
import geojson
from numpy import save, load
import os.path
import sys
from geojson import Feature, FeatureCollection
import json
import geojson
from shapely.geometry import shape
from shapely.geometry.point import Point
from shapely.geometry.polygon import Polygon

```

At first, we have all the imports, which is not really that interesting.

```python
1.  path=r'world.geojson'
2.  with open(path,'r') as data_file:
3.      data1 = json.load(data_file)
4.  
5.  feature_collection = FeatureCollection(data1['features'])
6.  
7.  
8.  path=r'Data/co2_emission_per_zone.json'
9.  with open(path,'r') as data_file:
10.     data = json.load(data_file)
```

In the first three lines we load a map that contains polygons defining each region. These polygons are then gathered in a list in line 5. From line 8 to line 10 we load the data for the average CO2 emission per region from 2021.

```
1.  polygons=[]
2.  zone_names=[]
3.  wrong_zones = []
4.
5.  for feature in feature_collection["features"]:
6.      poly: Polygon = shape(feature['geometry'])
7.      if(feature['properties']['zoneName'] in data):
8.          polygons.append(poly)
9.          zone_names.append([feature['properties']['zoneName'], feature['properties']['countryKey'], feature['properties']['countryName']])
10.     else:
11.         if(feature['properties']['zoneName'] not in wrong_zones):
12.             wrong_zones.append(feature['properties']['zoneName'])
```

While the polygons define all the regions where we have data from, there are also regions whose data is not available. These regions are sorted out here. We append a region to the list “wrong_zones” defined in line 3 when we don't have data for that specific region. Later we use that list to make sure that our model does not train on regions that we don't have any data for.

```
1.  gpus = []
2.  with open("Data/gpus.csv","r",encoding="Latin1") as gpus1:
3.      tempgpus = csv.reader(gpus1)
4.      for row in tempgpus:
5.          if(row[0] != 'name'):
6.              gpus.append([row[0], row[2]])
7.  df_gpus = pd.DataFrame(gpus, columns = ['name', 'tdp_watts'])
8.  df_gpus.tdp_watts = pd.to_numeric(df_gpus.tdp_watts, errors = 'coerce')
```

To calculate the CO2 emissions, the Thermal Design Power (TDP) watt value from the used graphic card is needed. In this code, the GPU data and thus, the TDP watt value is imported. The if clause in line 5 is there to get rid of the first row with the feature names. Lines 7 and 8 convert the array obtained from importing the data into a Pandas data frame that is suitable as input for a machine learning model.

```
training_points = []
x = np.linspace(-180,180,750)
y =  np.linspace(-90,90,350)
for j in range(len(y)):
    for i in range(len(x)):
        point = Point(x[i], y[j])
        poly_nr = -1
        for m in range(len(polygons)):
            if(polygons[m].contains(point) and not zone_names[m] in wrong_zones):
                poly_nr = m
                break
        if(poly_nr != -1):
            training_points.append([[x[i],y[j]], zone_names[poly_nr]])
```

Here, the preparation for the training data starts. In lines 2 and 3 we create 750 evenly spaced x-values between -180 and 180, and 350 evenly spaced y-values between -90 and 90 respectively. The x-values represent the degree of the longitude, while the y-values represent the degree of the latitude on the world map. The list “training_points” declared in line 1 should later contain all the combinations of x and y values defined previously, that are positioned inside a region where data is available. All possible x-y combinations are defined as points in line 6 by going through the loops. In the for-loop in line 8 it is checked whether a point is in a region with data or not. If the region had data, the point is added to the “training_points” list.

```
X = [x[0] for x in training_points]
y = [data[t[1][0]]['hydro discharge']['value'] for t in training_points]

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.25, random_state =
42)
```

Now that we have all the viable points, we can start preparing the data for our model. For this, we concatenate every training point, of the format [longtitude, latitude], which corresponds to a point on our world map, with the corresponding average hydro discharge of 2021 of the corresponding region.

In the following are some of the models we tried out for this task. The structure of each code snippet in the following is really similar, therefore we will just go ahead and document it once, after every model is listed.

```
regr = RandomForestRegressor(max_depth=30, random_state=100)
regr.fit(X_train,y_train)
y_pred = regr.predict(X_test)
score = r2_score(y_test, y_pred)
```

Random Forest Regression Model

```
regr = svm.SVR()
regr.fit(X_train,y_train)
y_pred = regr.predict(X_test)
score = r2_score(y_test, y_pred)
```

SVM Regression Model

```
regr = LinearRegression()
regr.fit(X_train,y_train)
y_pred = regr.predict(X_test)
score = r2_score(y_test, y_pred)
```

Linear Regression Model

```
1.  regr = tree.DecisionTreeRegressor()
2.  regr = regr.fit(X_train,y_train)
3.  y_pred = regr.predict(X_test)
4.  score = r2_score(y_test, y_pred)
```

Decision Tree Regression Model

```
1.  regr = MLPRegressor(random_state=1, max_iter=500)
2.  regr = regr.fit(X_train, y_train)
3.  regr.predict(X_test)
4.  score = r2_score(y_test, y_pred)
```

Neural Network

In the first line of these code snippets, we define the model. By inputting the previously obtained training data to the fit function of each model, the model is trained such that it can predict values on the test data in line 3. In line 4 the accuracy is then calculated by comparing the predicted values with the actual values.

```
1.  fig = plt.figure(figsize=(15, 8))
2.  for pol in polygons:
3.      if pol.geom_type == 'MultiPolygon':
4.          for g in pol.geoms:
5.              x,y = g.exterior.xy
6.              plt.plot(x,y)
7.      elif pol.geom_type == 'Polygon':
8.          x,y = pol.exterior.xy
9.          plt.plot(x,y)
10. plt.show()
11. plt.close(fig)
```

At last, we plot the result to a world map to get a good representation of what we achieved. Therefore, we plot all the polygons in the for-loop in line 2. A multi polygon is a region consisting of several other polygons, which we need to plot by plotting all the other polygons. This is done in the for-loop in line 4.

```
1.  fig = plt.figure(figsize=(15, 8))
2.  ax = fig.add_subplot(1,1,1)
3.  maximp, minimp = 0,1000
4.  for d in data:
5.      if(data[d]['hydro discharge']['value'] > maximp): maximp = data[d]['hydro dis-
    charge']['value']
6.      if(data[d]['hydro discharge']['value'] < minimp): minimp = data[d]['hydro dis-
    charge']['value']
7.  normal_imps = []
8.  col = []
9.  xs = []
10. ys = []
```

```
for t in training_points:
    imp = data[t[1][0]]['hydro discharge']['value']
    n_imp = 1 - (float(imp-minimp)/maximp-minimp)
    normal_imps.append(n_imp)
    col.append(plt.cm.hot(n_imp))
    xs.append(t[0][0])
    ys.append(t[0][1])
ax.scatter(xs,ys,c=col,s=0.1)
ylim = ax.get_ylim()
xlim = ax.get_xlim()
plt.show()
plt.close(fig)
```

We first plot the training data, to later compare it to our predicted result. We begin by searching the data for the maximum and minimum hydro discharge values, to later use them for normalization. Then, we search the data for the actual average impact, normalize it and append it to our lists "normal_imps" / "xs" / "ys" for plotting, as well as already saving the corresponding colormap value, to save us some computing time and avoid a second for-loop later.

```
1.  x = np.linspace(-180,180,400)
2.  y =  np.linspace(-90,90,400)
3.  X, Y = np.meshgrid(x,y)
4.
5.  xyvals = []
6.  for j in range(len(y)):
7.      for i in range(len(x)):
8.          xyvals.append([x[i], y[j]])
9.  T = regr.predict(xyvals)
```

After plotting the map with the real impact values, we want to plot them with the predicted values. In the first two lines we create evenly spaced values in the given ranges again like previously. The for-loops in lines 6 and 7 are then needed put the evenly spaced values in the correct format for the prediction which is done in line 9. Line 3 is only needed to plot the data later on, as a mesh grid is needed.

```
1.  def isInAPolygon(x,y):
2.      for m in range(len(polygons)):
3.          if(polygons[m].contains(Point(x,y))):
4.              return True
5.
6.  for i in range(len(T)):
7.      if (not isInAPolygon(xyvals[i][0],xyvals[i][1])):
8.          T[i] = 0
```

For better comparability in the images, we set the impact of the regions where we have no data to 0, to make sure they are displayed white. The for-loop in line 6 takes care of that and sets the impact for all regions without data to zero. To find out if a region contains data, we wrote a small function defined in line 1. This function checks whether a point is in a polygon or not and returns true or false respectively.

```python
1.  normal_T = []
2.  max_T = max(T)
3.  min_T = min(T)
4.  normal_T = [1 - ((float(i)- min_T)/(max_T-min_T)) for i in T]
5.  normal_T= np.ma.masked_array(normal_T,mask=[t==1 for t in normal_T])
6.  normal_T = normal_T.reshape(-1,400)
```

We want to create a plot that shows the impact of each region by colouring them. Since colour values are in a range from 0 to 1, we have to normalize the impact before plotting it. The normalization is done in line 4. Now “normal_T” contains the normalized impact of every point, however we need to do 2 further steps before plotting it.
First, a mask has to the list, that is true for all impact values of regions that had no data. The reason for this is, because the function we are going to use later does not plot masked values, and instead leaves them white. If we would not apply this mask, all regions without data would still be coloured, since the function we are going to use for plotting fills out all areas it can find with the colours of their surrounding contours.
The last step needed before plotting is reshaping the list. The points defined by x and y in the mesh grid have dimension 400x400, but the normalized predicted results list “normal_T” has dimension 1600x1. Therefore, we have to reshape it such that the dimension fits.

```python
fig = plt.figure(figsize=(15, 8))
ax = fig.add_subplot(1,1,1)
ax.set_facecolor("white")
ax.contourf(X, Y, normal_T, cmap=plt.cm.hot)
ax.set_ylim(ylim)
ax.set_xlim(xlim)
plt.show()
```

All that is left to do in the end is to plot the data.

Declaration of Authorship

I affirm that I have produced the work independently, that I have not used any aids other than those specified and that I have clearly marked all literal or analogous reproductions as such.

Saarbrücken, 22.07.2022

Abdul Mueed, Muhammad

Paulus, Mina

Franke, Lars

Zender, Dennis