Machine Learning Elective

Project: Name-Localizer

Lecturer: Meghan Kane

Student: Matthias Dietrich (755634)

Motivation & Project Idea

Before I started studying Interactive Media Design, I completed an apprenticeship as a geomatic, which is pretty much a more contemporary version of the former cartograph. I was working a lot with geodata and cartographic visualizations. That is why I wanted to do ML project that relates to a geographic or cartographic topic.

I started searching for open source geodata, that I could use for my project. My former employer, the Federal Agency for Cartography and Geodesy, provides different open source datasets, like terrain models or administrative boundaries. I decided to use the dataset named “Geographische Namen 1:250 000 (GN250)”, that can be found *here*.

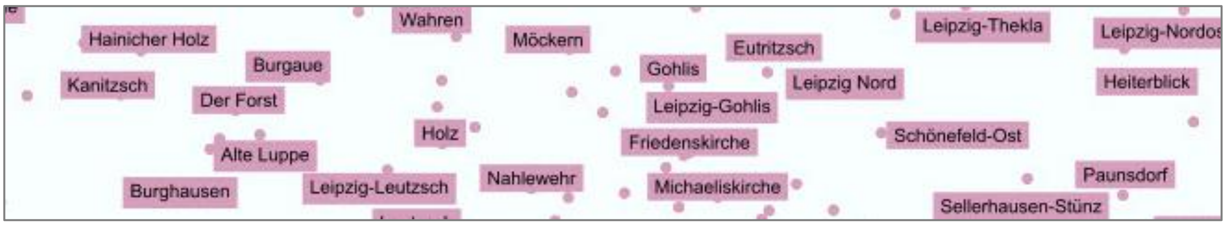


Figure 1: GN250 Dataset

The dataset contains the names of different object areas with coordinates and additional attributes. If you travel through Germany, you will notice that the places in certain regions often have similar names. The project idea was to find out to what extent the name of a locality allows conclusions about its geographical position.

Four Stages

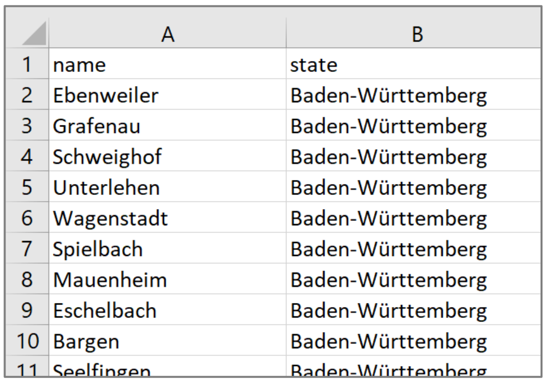
Like every machine learning process, this project can be separated in four stages that build on each other. In the following I would like to give a brief insight into how I proceeded in these four stages.

Stage 1: Framing the problem

The goal of this project was to have a ML model that can *predict the geographical position of a location based on its name.* This could be done with different approaches. My first idea was using the names together with the coordinates of existent locations to train a model and then be able to predict coordinate-pairs based on a given name. This would have meant that the model would have to learn from three different attributes: the name of a location together with the X- and Y-coordinate. Lastly, I decided to follow a more beginner-friendly approach. I used the federal state in which the existent locations are located to classify their geographic position. “Frankfurt” for example would be labelled to be in “Hessen” or “München” would be labelled to be in “Bayern”. This approach is also referred to as *multi-label classification*. Luckily the found “GN 250” dataset also contains the state in which the location is located. I was now able to move on to the next stage.

Stage 2: Preparing the data

As I already mentioned, the “GN 250” dataset includes many different names of different objects. I was only interested in the location-objects so at first, I had to reduce the dataset to only include the data rows of type “AX_Ortslage”. Secondly, I removed every additional attribute besides the “NAME” and the “BUNDESLAND” attribute. I also renamed these attributes to “name” and “state”. The dataset now looked like you can see in the image on the right.



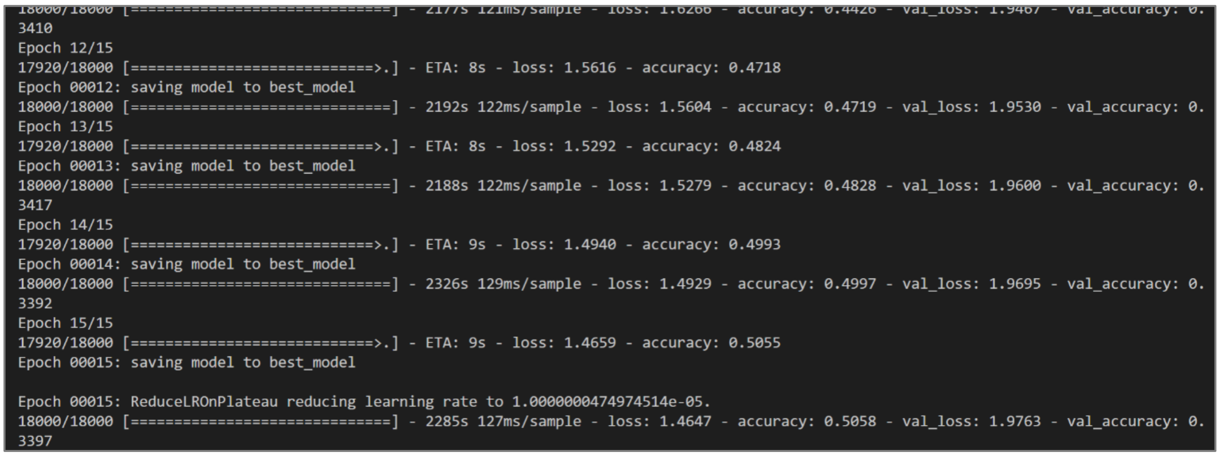
	A	B
1	name	state
2	Ebenweiler	Baden-Württemberg
3	Grafenau	Baden-Württemberg
4	Schweighof	Baden-Württemberg
5	Unterlehen	Baden-Württemberg
6	Wagenstadt	Baden-Württemberg
7	Spielbach	Baden-Württemberg
8	Mauenheim	Baden-Württemberg
9	Eschelbach	Baden-Württemberg
10	Bargen	Baden-Württemberg
11	Seelfingen	Baden-Württemberg

Figure 2: Training Data

Now I needed to make sure that I have the same number of locations for every state, so that every state is represented equally inside the dataset. First, I removed the city-states “Hamburg”, “Bremen” and “Berlin”, since they have only a few locations. Now the state with the fewest locations was the “Saarland” with a number of 391 locations. Every other state has more than 2000 locations within its boundaries. I decided to also remove the “Saarland” from my dataset, because otherwise I would have had to reduce the amount of locations for every other state drastically, which would have resulted in a rather poor dataset. Now I was able to reduce the dataset once again to have 2000 rows of data per each of the remaining 12 states (altogether 24.000 locations). I saved this dataset as a .csv file.

Stage 3: Training the model

For my project I chose a recurrent neural network with a Long short-term memory architecture. I used Python together with TensorFlow. I orientated myself on a similar project that can be found *here*. The benefit of a LSTM network is that they are capable of learning long-term dependencies. For example, they are able to identify relations between the first characters and the last characters of a word. I used a bidirectional LSTM Layer, so the output depends on the past and the future characters of a given name. I also added two dropout layers to prevent Overfitting. While training I reached a maximum accuracy of 33% in 15 epochs, which I considered as enough for the purpose of my project.



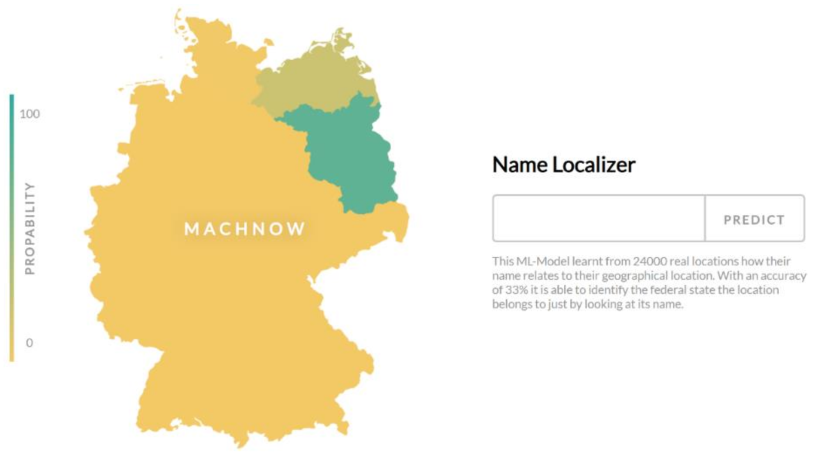
```
18000/18000 [==============================] - 2177s 121ms/sample - loss: 1.6266 - accuracy: 0.4426 - val_loss: 1.9467 - val_accuracy: 0.
3410
Epoch 12/15
17920/18000 [============================>.] - ETA: 8s - loss: 1.5616 - accuracy: 0.4718
Epoch 00012: saving model to best_model
18000/18000 [==============================] - 2192s 122ms/sample - loss: 1.5604 - accuracy: 0.4719 - val_loss: 1.9530 - val_accuracy: 0.
Epoch 13/15
17920/18000 [============================>.] - ETA: 8s - loss: 1.5292 - accuracy: 0.4824
Epoch 00013: saving model to best_model
18000/18000 [==============================] - 2188s 122ms/sample - loss: 1.5279 - accuracy: 0.4828 - val_loss: 1.9600 - val_accuracy: 0.
3417
Epoch 14/15
17920/18000 [============================>.] - ETA: 9s - loss: 1.4940 - accuracy: 0.4993
Epoch 00014: saving model to best_model
18000/18000 [==============================] - 2326s 129ms/sample - loss: 1.4929 - accuracy: 0.4997 - val_loss: 1.9695 - val_accuracy: 0.
3392
Epoch 15/15
17920/18000 [============================>.] - ETA: 9s - loss: 1.4659 - accuracy: 0.5055
Epoch 00015: saving model to best_model

Epoch 00015: ReduceLROnPlateau reducing learning rate to 1.0000000474974514e-05.
18000/18000 [==============================] - 2285s 127ms/sample - loss: 1.4647 - accuracy: 0.5058 - val_loss: 1.9763 - val_accuracy: 0.
3397
```

Figure 3: Training Log

Because of the rather low accuracy I had to face the problem that most ML models do not behave deterministic. This is usually not a problem, because a high accuracy will lead the model to return the same output for the same input. In my case though, the same input resulted in different outputs, which was not ideal. So, I had to prevent this non-deterministic behaviour. To accomplish that I used seeds. Seeds can be used on various functions that return random outputs. Whenever one of these functions is called, using the same seed (e.g "1"), the function will return the same output. I applied these seeds to the dropout layers of the ML model, the python random function as well as the numpy.random function. I also had to set the PYTHONHASHSEED to ensure a completely determinsitic behavior. In general it is also important to train the model on a CPU, instead of a GPU, to avoid non-determinism. Another way to avoid non-determinism is to limit the amount of threads, that the application uses, to only one.

Stage 4: Predicting

With a trained and deterministic model, I was now able to predict the state based on a string input. I decided to create a small web application for handling the input and visualizing the output. For that, I used a python backend with a flask webserver that will render a HTML template. The output of the ML model will be presented on a map of Germany. The higher the predicted probability for a specific state is, the more green it will appear on the map. So instead of returning only the state with the highest propability, the output is some kind of heatmap, as you can see in the image below.



Name Localizer

This ML-Model learnt from 24000 real locations how their name relates to their geographical location. With an accuracy of 33% it is able to identify the federal state the location belongs to just by looking at its name.

Learnings & Course Summary

Before this course, I had only a poor understanding of machine learning. During this course I learned a lot about the basic principles of machine learning. Starting with the four stages of machine learning, I learned how to approach a ML project from scratch and what I need to be aware of before starting off. I got a basic understanding of how a ML model is structured, what kind of networks exist and how they are able learn. I also got an insight for what purposes a ML model can be used for and for which cases it is not applicable. I really liked the practical, project-based approach of this course, because for me it is the most interesting way to explore new technologies. Although the course took place under special circumstances due the Covid-19 pandemic, I felt well supervised by the lecturer through online classes and 30 minute-meetings. Summarized I really enjoyed attending the course and I have learned a lot from it.