Species Prediction based on Environmental Variables using Machine Learning Techniques

Stefan Taubert[1], Max Mauermann[1], Stefan Kahl[1], Danny Kowerko[2], and Maximilian Eibl[1]

[1] Chair Media Informatics,
Chemnitz University of Technology, D-09107 Chemnitz, Germany
[2] Junior Professorship Media Computing,
Chemnitz University of Technology, D-09107 Chemnitz, Germany
[3] {stefan.taubert, max.mauermann, stefan.kahl, danny.kowerko, maximilian.eibl}@informatik.tu-chemnitz.de

Abstract. In our working notes we discuss different machine learning approaches for predicting species for different geolocations based on environmental variables. We used convolutional neural networks (CNN) for image classification as well as data extraction with tree boosting as models. Furthermore, we analyzed different approaches for data extraction, as well as data augmentation and group forming to improve our models performance.

Keywords: Species Prediction · Machine Learning · Environmental Variables · GeoLifeCLEF

1 Description of the dataset and task

The training dataset provided for the 2018 GeoLifeCLEF task includes around 260.000 occurrences of plants from different species. Each occurrence is associated with a 33-channel TIFF image. Each channel represents a map of values for a different Environmental Variable (EV), centered around the location of the occurrence. Additionally, a .csv file containing single values for each EV at the center of each occurrence is given. Since each occurrence is associated with a species, the goal of the GeoLifeCLEF task was to train a model, that is able to predict the species based on these EV.

2 Dataset Analysis and Pre-processing

In this section we will introduce the different analysis methods we applied to assess the dataset. This provides an understanding of how the dataset was organized and structured. We also present various methods of pre-processing we utilized to extract the data, that was used in our approaches.

2.1 Extracting single values from images

Since the dataset was given in two versions - images and a .csv-file containing single values - we planned to develop approaches for both types. After some analysis of the .csv-file, we noticed, a large amount of cells without values. These cells render their respective rows unusable. To fill out the missing values, we first tried to calculate them by averaging over the 4 centered pixels of the respective images, since the single values are supposed to be the EV values at the exact location of the occurrence. We also did this calculation for the images, for which values existed in the .csv file, but realized that it was not possible to reproduce these values. Because of this discrepancy we decided very early to not use the given single values from the .csv-file, but rather to create our own .csv-file by extracting the single values from the images like mentioned above.

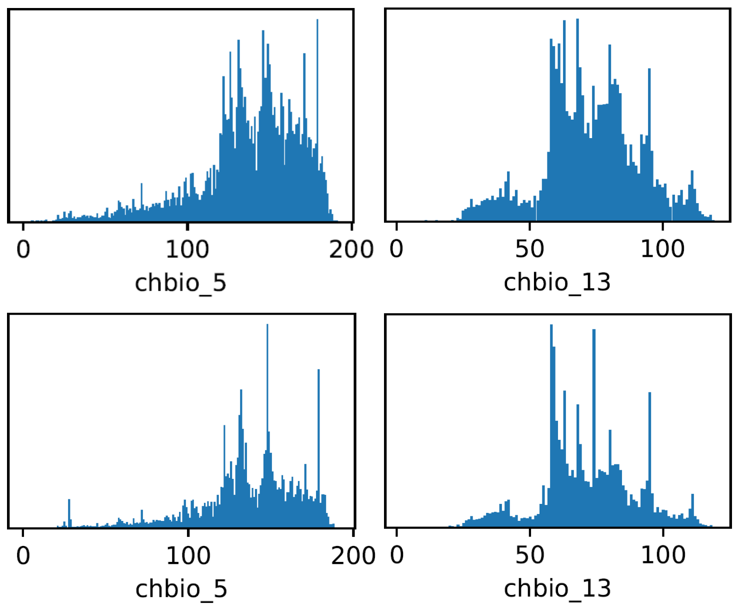

Fig. 1. Comparing test- and trainset: The value occurrences for the channels **chbio_5** and **chbio_13** of the testset (top) and trainset (bottom) show that test- and trainset have different value occurrences. We will show later that this is one of the reasons that our test- and validation scores differed by a great margin.

We also used another variation of this extracting algorithm, were we averaged the single values over the whole image instead of only using the 4 centered pixels. Both variations were used in different submissions we made. We did not transfer the values of each EV into their respective ranges, but rather kept them at the range from 0 to 255, like in the images.

2.2 Most common values for each class

After this pre-processing step, we encountered a lot of raw numbers associated with different classes. They were not intuitively understandable, since the number of classes was also very high. For this reason we tried to create a fingerprint for each class. We did this by counting the single values for each class and creating a diagram, which shows how often each value occurs in every class. This way we gained some insight which values are discriminative for a specific class.

2.3 Finding the most discriminative channels

The relatively high number of 33 different channels indicated that not all channels are equally useful. We decided it would be worth to cut some of them, because they are not sufficiently discriminative. To achieve this, we searched for combinations of channels based on the extracted single values, that were sufficient to fully seperate the entire training set. We came across multiple combinations, one of which was used later when training a multi-model CNN.

2.4 Creating Heatmaps for all classes

Since the TIFF images of the dataset are no photographic images, it was hard to grasp the features that identify a specific class. In order to get a better understanding of what characterizes different classes, we created an heatmap for each class, which contains the average values of all images for that classes. Furthermore, all values were normalized to the range between 0 and 1.

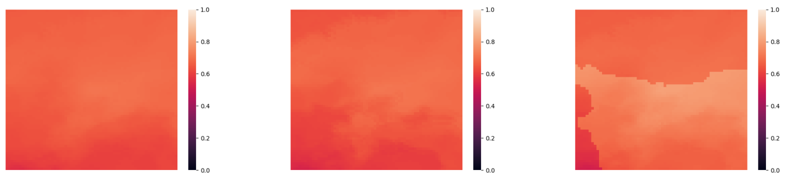

Fig. 2. Heatmaps for the 3 most common classes in the training dataset, for channel 12 (annual percipation)

However, the heatmaps for the most common classes looked very similar, which further proved our assumption that the given data is not distinctive enough to predict classes properly.

2.5 Splitting a validation set

We used 10% of the training dataset, to create a separate validation set. We used this split to estimate, how well our models and different approaches performed and to generate intermediate checkpoints for some of our models.

3 Submissions

In this section, we will comment on our submissions made during the challenge. For each submission, we will describe the approach we used and we will discuss why some approaches produce better results then others.

3.1 Random Guessing

The most basic model we used was based on random guessing. We extracted all 3336 possible species and assigned them randomly to each row of the testset. The score of this submission was 0.0016.

3.2 Probability Model

First, we counted the occurrences of each species in the trainset for the probability model. Then we assigned the species with descending probabilities to each row of the testset (so every row had the same prediction). That means that the most common species was at rank one, the second most common species at rank two and so on. The score of this submission was 0.0134.

3.3 Vector Model

The vector model calculated the differences between the rows of the trainset and the testset and predicted each testrow with the species of the trainrow with the smallest difference. Therefore we took all 33 features and shaped a 33-dimensional vector V out of each row. We calculated the difference D as followed:

$$\begin{aligned}
\overrightarrow{V_n} &= \text{Vector of } n\text{-th trainrow}\\
\overrightarrow{V_m} &= \text{Vector of } m\text{-th testrow}\\
\overrightarrow{V_{nm}} &= \overrightarrow{V_n} - \overrightarrow{V_m}\\
D_{nm} &= |\overrightarrow{V_{nm}}|
\end{aligned}$$

Then we sorted all D_{nm} values in descending order and looked which species S_{nm} were assigned. As the species occurred multiple times for each m we always kept the first occurrence in S_{nm}. The result was a sorted list of 3336 species for each row m. At the end we took the first 100 species of the lists and ranked them ascending for the final submission. The score of this submission was 0.0271.

3.4 Convolutional Neural Networks

When using the TIFF-images directly for training, we decided to use convolutional neural networks. In this section we will discuss the different approaches we tried during the challenge. We used the Keras framework[5] to implement all models that were used.

VGG-like-models The first approach was a simple feed forward CNN, similar to the popular VGG model[7], which won the ImageNet contest in 2014. We choose this relatively simple structure to better understand how the model works with the given data.

The exact architecture we used consisted of five convolutional layers followed by a global average pooling layer and the classification layer. We implemented max pooling after the first two and every second convolutional layer, which divides the spatial dimensions of the image by half. Additionally, we doubled the number of filters in the convolutional layers after each pooling layer .

During training we also recognized, that our model was suffering from severe overfitting. We decided to use heavy regularization in form of l2-regularization in the convolutional layers and dropout after each global pooling layer.

We also used two approaches for this architecture. One model that is trained with all 33 channels of the image and one ensemble, which consists of 6 individually trained models, each one using one of 6 channels we assumed to be the most discriminative ones.

To further improve our models, we tested classic affine transformations as data augmentation on the images that were used for training. First, we only used random flipping (vertical or horizontal) and random rotations. We later added random crops to the range of possible augmentations, to see if they would lead to further improvements.

DenseNet We included the more complex DenseNet[6], which is included in Keras, despite the low scores of our CNN approach. We decided to not use any pre-trained weights. We assumed that this would not be helpful, since the images of our dataset did not share any features with the photographic images from ImageNet. We submitted a single DenseNet which was trained without any of the augmentations that were mentioned above. Additionally, we trained the DenseNet as a single model, not using the ensemble approach we used with the VGG-like models.

Training the CNN All of our CNN were trained with checkpoint learning (or early stopping). This means that we used the validation set to calculate a validation loss after each epoch of training. Every time the validation loss decreased, we saved the models weights. This way, we only used the best possible weights, that occurred during the training, to create our submissions.

Results of the CNN The following table shows the results of our submissions, that were accomplished with the CNN. The table shows the results for the single model and the multi-model approach for the VGG-like network. Rows indicate which data augmentations were used.

We noticed that the result had very high fluctuations and that it was not clearly visible which augmentation methods did provide an improvement, when looking at the test set scores. For the single model, only flips and rotations

Table 1. The final settings for the used architectures.

augmentations	VGG-like single	VGG-like multi	DenseNet
none	0.0096	0.0153	0.0099
flip, rotate	0.0153	0.0144	-
flip, rotate, crop	0.0103	0.0096	-

seemed to provide an improvement and the ensemble seemed to work best without any augmentations. The crops did not lead to any improvement and the DenseNet is about as good as the VGG-like single model.

However, these scores are the result of the official test set evaluation. When validating on our separate validation set, we generally found all augmentations to be helpful.

3.5 XGBoost

One other approach besides neuronal networks was the usage of Extreme Gradient Boosting (XGBoost)[1][2]. This section describes our two strategies for building a model with XGBoost. The traindata was created from the images and the occurrences csv-file. We took all 33 channel values and the parameter latitude and longitude for each row in the datasets and saved those information in a new csv-file. For all runs we created one validationset out of the trainset (10% of the data) to get a hint when we overfit the model. The validationset contained no species which are not occurred in the trainset. The model parameters were adjusted every run and we used early stopping rounds to stop the training when the model overfitted. XGBoost has one parameter max_depth which defined how deep the boosted trees were build. That means that the higher the depth was defined the more complex the resulting model was. We tried different depths to get a understanding how complex our model has to be.

Single Model The first strategy was to build one single model for the prediction. The evaluation we choose for this multi-class classification was the merror metric which was basically equal to the top-1-accuracy. We submitted the following variations:

Table 2. The results show that the value 10 for max_depth was to big and the model was to complex whereas value 2 was that one with the best score.

Variation	max_depth	Test-Score
1	10	0.0085
2	3	0.0344
3	2	0.0348

Multi-Model The second strategy was to predict each species separately. Therefore we created 3336 different models. Each model was evaluated with the logloss metric. We tried following parameter settings:

Table 3. Multi-model evaluation with different parameter settings. Scores increased after calculating the pixel values solely from the four pixels of the center of the images.

Variation	*max_depth*	Pixel Count	Test-Score	Validation-Score
1	3	64x64	0.0338	0.0942
2	2	64x64	0.0352	0.0963
3	2	2x2	0.0358	0.0884

Groups Model To get a possibly better score, we created an advanced model which built several groups with species that are similar. The idea was to group species because we argued that it is likely that more than one species occur at one location. In order to find similar species, we looked at the rows for each species in the trainset and calculate a representative value per species. One way of doing so was to look at each feature column and estimate which value occurred most. The second method was to calculate the mean value for each feature column. This way, we calculated a simple fingerprint for each species.

In the next step we calculate the difference between all species with the same method like in the Vector Model. The result was a 3336x3336 matrix in which the distance between each species was held. We had to round the feature values to n digits and we ignored species which occurred less then a count of o.

After that, we defined the parameter t for threshold which indicates how great the distance between two species could be to consider them in the same group. After this, we created a network in which all species relations were contained. The following figure illustrates some of the problems that arise with that approach.

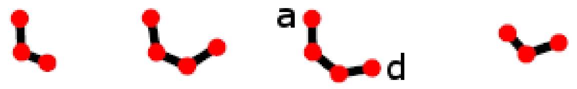

Fig. 3. Each point represents a species and the connections are distances between the species which are within t. The main problem is that species **a** has a difference of $3t$ to **d** in the worst case and is not really similar.

Therefore we defined a additional parameter *min_edges* which defines how much neighbours a species needs to be in a group. As result we got less groups but groups with species which are more similar.

At both runs we used all 64x64 pixels and round the data to 2 digits. For the mapping back to the species later in the submission we took all species of

Fig. 4. Now the species in the groups are more similar at a *min_edges* of 3.

an group and sorted them descending with regard of their probability in the trainset.

Table 4. The score of the run in which the mean was calculated was slightly better than the other run but not as good as the run without any groups.

Variation	*max_depth*	*min_edges*	*t*	*o*	Method	Test-Score
1	4	1	3	10	most common	0.0220
2	2	3	9	0	mean	0.0326

At variation 1 we have got 2148 groups in which were 266 species in groups with an size greater one. So we trained each group with 2148 models. At Variation 2 we have got 3301 groups and 39 species in groups with a size greater one.

We realized that it was not an improvement to build groups because the test scores were not as good as the scores without groups. The reason could be that the machine learning algorithm detected by itself which species are similar and it therefore was more correct then with building the groups manually.

4 Sourcecode

We published our source code at GitHub[4]. The code is written in python and uses different libraries. At the project page are instructions how to execute the code. We used an i7 and a P6000 for model training.

5 Summary

As the other participants and our results show, predicting plant species based on environmental variables can be a challenging task. Furthermore it is especially challenging when it comes to predicting only a single species, since there are many species that may occur under the same environmental conditions. Anyway we evaluated various machine learning techniques to get an idea of which work best for this problem.

[4] https://github.com/stefantaubert/lifeclef-geo-2018

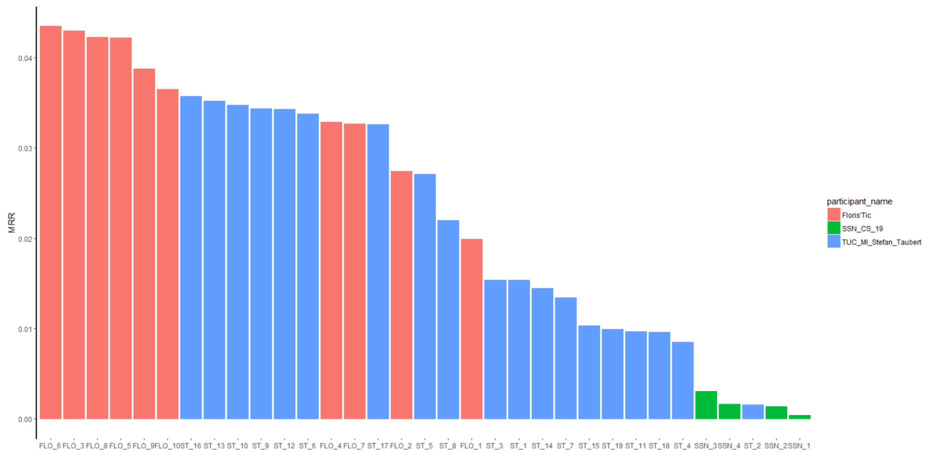

Fig. 5. As in the result graph is shown the overall scores were really low because of similar EV.

Our approaches were divided into two kinds, one using the 33-channel images and CNN, the other one using single values, that were extracted from these images. For the CNN part we mainly used a traditional feed forward network in two variations: one using all 33 channels of the images, the other one was an ensemble of 6 individually trained models, each one using one of 6 selected channels. After using the unprocessed versions of the images we also added different affine transformations as data augmentation. On the test score augmentations did not lead to an improvement, but when evaluating them on our local validation set they had proven to be useful. In general, at their best, our CNN were merely better then a probability based approach, which implies that more optimization would be needed to make this technique work properly.

Our single value approach used boosted trees with the XGBoost framework. The first parameter we tried to optimize with this approach was finding the right depth for our trees. We started with a very high depth, but we found out, that less deeper and therefore less complex models yield better results. After that we also changed the way in which we extract the single values from the images. It has shown, that averaging only over the 4 centered pixels leads to better results then averaging over the whole image. Lastly we also tried out forming groups of similar species and predicting them instead of single species. We counted predicting a group as predicting all species within this group ordered along their frequency in the train set. This data processing step did not lead to an improvement of our score.

In general we have to state, that our local validation score constantly differed from the challenges test score, which led us to the conclusion that the species of the tasks dataset are not completely distinguishable.

References

1. Tianqi, C., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, ACM (2016)
2. Friedman, J. H.: Greedy function approximation: a gradient boosting machine. In: Annals of statistics, 1189-1232. (2001)
3. Botella, C., Bonnet, P., Joly, A.: Overview of GeoLifeCLEF 2018: location-based species recommendation. In: CLEF working notes 2018. (2018)
4. Joly, A., Goëau, H., Botelle, C., Glotin, H., Bonnet, P., Planque, R., Vallinga, W.P., Müller, H.: Overview of LifeCLEF 2018: a large-scale evaluation of species identification and recommendation algorithms in the era of AI. In: Proceedings of CLEF 2018. (2018)
5. Chollet, F. et al.: Keras. (2015)
6. Huang, G., Liu, Z., Weinberger, K. Q., van der Maaten, L.: Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition (Vol. 1, No. 2, p. 3). (2017)
7. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: arXiv preprint arXiv:1409.1556. (201