Learning a rule

Christoph Netz & Theo Pannetier

I. Introduction

A dataset may be linearly separable by many different solutions. The Rosenblatt perceptron algorithm is designed to find one of these solutions in version space. But are all of these solutions equally suitable? There may be solutions that correctly separate the dataset, but do so only with a narrow margin between examples of different labels. Such a solution will be very sensitive to noise when applied to new data points. Therefore, we search for the hyperplane with the largest possible margin separating the two classes, giving rise to the perceptron of optimal stability. This perceptron typically lies in the center of version space and provides a good generalization ability.

The Minover Algorithm can be used to determine the perceptron of optimal stability. It iteratively updates the weight vector with Hebbian terms based on the example $\xi^{\mu(t)}$ of minimal stability. If the dataset is linearly separable, it converges and yields the perceptron of optimal stability. Here, we demonstrate how a student perceptron learns a rule by means of the Minover Algorithm that was produced by a teacher perceptron.

II. Solution

We first initialize a so-called teacher perceptron w^* with random values in N dimensions. Here, we drew each weight in a normal distribution with $\{\mu = 0; \sigma = 1\}$ and normalized them such that $\|w\|^2 = N$. This teacher perceptron is then used to determine the labels $\boldsymbol{S}^\mu \in \{-1, 1\}$ of P randomly generated N-dimensional input vectors $\{\boldsymbol{\xi}\}_{\mu=1}^{P}$.

We then seek to obtain a student perceptron that classifies the data accurately, and try to minimize the discrepancy between the student and teacher perceptron with regard to novel input data, that is, we seek to minimize the generalization error ϵ.

The student perceptron is initialized with weigths w(t=0) = 0. We loop over P and determine the example $\xi^{\mu(t)}$ with the lowest stability based on w(t=0):

$$\kappa^{\mu(t)} = \min_{\nu} \left\{ \kappa^{\mu}(t) = \frac{w(t)\xi^{\nu(t)} S_R^P}{|w(t)|} \right\}$$

Next, the student perceptron is updated based on the Hebbian term $w(t+1) = w(t) + \frac{1}{N}\xi^{\mu(t)}\boldsymbol{S}^{\mu(t)}$. The implementation of these two steps in C++ is shown in Fig. 1. The procedure is iterated over t until $t = t_{max}$, where $t_{max} = n_{max} * P$.

The pseudocode for the Minover algorithm is presented in Box 1.

At the end of the training process, we calculate the generalization error $\epsilon_g(t_{max})$. We compute the average generalization error for nD replicates of the data set. We repeat this procedure for $N = 10, 20, 100$ and for $\alpha = 0.1, 0.2, ..., 5.0$ with $\alpha = \frac{P}{N}$.

To fasten the computation, we allowed the minover algorithm to be escaped before t_{max} if the error did not change by more than 0.005 units for 1 consecutive steps. This significantly shortened the runtime, as optimal stability was attained in relatively few steps.

Box 1. Pseudocode for the perceptron

- Initialize $\boldsymbol{w}(0) = 0$
- repeat
- for each μ element from 1 to P
- compute $\kappa^{\nu}(t) = \frac{w(t)\xi^{\nu(t)}S^{\nu}}{|w(t)|}$
- end for
- select $\mu(t)$ with $min_{\nu}\{\kappa^{\nu}(t)\}$
- update $w(t+1) = w(t) + \frac{1}{N}\xi^{\mu(t)}\boldsymbol{S}^{\mu(t)}$
- until stopping criterion is reached or $t = t_{max}$

```
// Loop through points to find example of minimal stability
for (int nu = 0; nu < P; ++nu) {

  double dotProductStudent = dataset[nu].getDotProduct(studentVector);
  double kappa = dotProductStudent * dataset[nu].get_truelabel();

  if (kappa < dMinStability) {
    indexMinStability = nu;
    dMinStability = kappa;
  }
}

dataset[indexMinStability].updateWeights(studentVector);
```

```
// Update weight vector in argument using the input's feature vector
void updateWeights(vector<double> &weightVector) {
  for (int i = 0; i < weightVector.size(); ++i) {
    weightVector[i] += featureVector[i] * label / featureVector.size();
  }
}
```

Fig. 1. C++ code snippets for 1) determining the example from the dataset of minimal stability and 2) updating the perceptron.

III. Results

The perceptron stability increases over training time, as the distance between the hyperplane defined by the student vector and the closest example increases in the feature space. However it does so not continuously, with the stability decreasing in some steps. For very large datasets (large αN values, the algorithm did not return positive stabilities. Linear separability is guaranteed as the data is labeled based on the teacher perceptron. Therefore the algorithm was not able to find the existing correct classification of the dataset within runtime t_{max} (results not shown). We also tested whether the initialization of the teacher network had an effect on the training process by setting the weights to different random or constant values (e.g. setting all weights to 1). It seems

that the teacher network can be initialized to weights of any value without having an effect to the training of the student network. The magnitude of the teacher weights was also found to be irrelevant here, it is rather the tuning of weights to all other vector weights that is decisive. On the other hand, a different random generation of the input data might have a much stronger effect, but we did not test for that. After the algorithm has reached t_max or fulfilled the implemented stopping criterion, we went on to calculate the generalization error:

$\epsilon_g(t_{max}) = \frac{1}{\pi} arccos(\frac{w(t_{max}) * w^*}{|w(t_{max})||w^*|})$

We observe that the generalization error drops as α increases (fig. 2). With increasing N dimensions, the curve becomes flatter. Probably, the increased N increases the version space overproportionally to α.

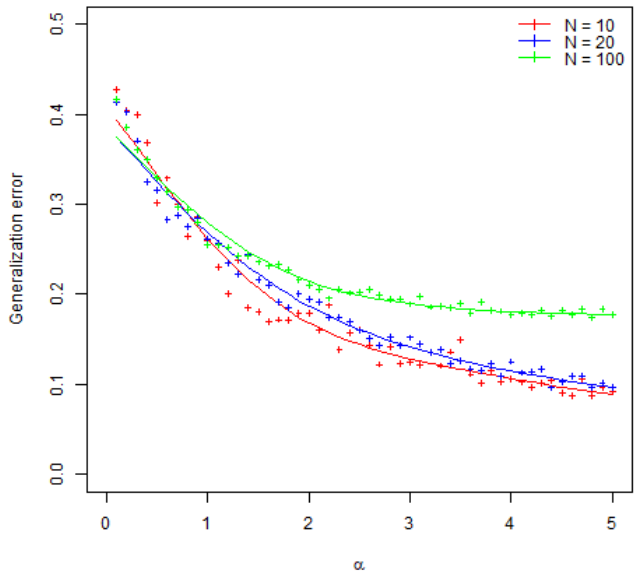

Fig. 2. Generalization error ϵ_g in relation to $\alpha = \frac{P}{N}$. The values for α range from 0.1 to 5.0 in increments of 0.1, N = 10, 20, 100. The generalization error drops with increasing values of α as the version space decreases. With increasing dimensions N the decrease becomes flatter.

IV. Discussion

A perceptron algorithm seeks to determine a true classification of data (here defined by the teacher perceptron) based on a training set presented to it. Since the training set can only contain a sample of all possible points, a student perceptron that accurately classifies all data in the training set may still yield a wrong classification when presented with novel input. This discrepancy between the teacher and the student is measured by the generalization error.
The minover algorithm optimizes the stability of the perceptron and thus constitutes a strategy to reduce the generalization error. In a geometric interpretation, the algorithm tries to place the student perceptron close to the center in the space of possible perceptrons (the volume space) to maximize the probability that a randomly drawn new example falls on the right side of the classification.

???Here we have found that this strategy reduces the generalization error of the student perceptron compared to the Rosenblatt perceptron (Fig. 2, to compare with the figure discussed in class).??? This is especially true for low to medium values of α, where we observe the algorithm to learn well before the carrying capacity. We interpret this as a consequence of a large version space when there is only few examples. Stability optimization in such a case allows to reduce the average distance to all the possible teacher perceptrons in the version space. By contrast, when the number of examples is high, optimized stability shows little effect as the version space is already highly constrained.
Interestingly, when no correct classification was reached (that is, in the case of large P, see Results section), stability optimization still yields a perceptron that is closer to the teacher (Fig. 2, blue and green curves) ???CN than what??.

V. Conclusion

Our program has shown that the concept of optimal stability can be used to train a student network with the Minover algorithm. In doing so it is important to provide enough examples P to the training process, otherwise the generalization error ϵ_g will be very large. Based on our results we can recommend to choose P such that $\alpha > 2$. One should note that the total number of dimensions N can also influence ϵ_g in relation to α. If we had more time it would have been very interesting to compare the Minover algorithm to the Rosenblatt Perceptron algorithm. We expect the Rosenblatt Perceptron algorithm to perform much worse especially for low values of α, as it can only find one random solution in version space, whereas the Minover algorithm allows us to converge to the center of version space.

VI. Code availability

The C++ code used to generate the data and the R code used for the plots can both be found at https://github.com/TheoPannetier/NNCI_winter2018

VII. Contributions

TP wrote the code, debugged and analysis, CN debugged and wrote the report. TP reviewed the report. Both authors contributed equally to this assignment.