Co-funded by the Horizon 2020 Framework Programme of the European Union

Grant agreement No. 671555

ExCAPE
Exascale Compound Activity Prediction Engine

Future and Emerging Technologies (FET)

Call: H2020-FETHPC-2014
Topic: FETHPC-1-2014
Type of action: RIA

Deliverable D1.9

Report: Report + Code 6

Platt Scaling

Due date of deliverable: 31.08.2018
Actual submission date: 31.08.2018

Start date of Project: 1.9.2015 Duration: 36 months

Responsible Consortium Partner:	RHUL
Contributing Consortium Partners:	UL
Name of author(s) and contributor(s):	Andreas Mayr, Günter Klambauer
Internal Reviewer(s):	Xiangju Qin
Revision:	V3

Project co-funded by the European Commission within Horizon 2020 Framework Programme (2014-2020)		
Dissemination Level		
PU	Public	PU
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Document revision tracking

This page is used to follow the deliverable production from its first version until it has been reviewed by the assessment team. Please give details in the table below about successive releases.

Release number	Date	Reason of this release and/or validation	Dissemination of this release (task level, WP/ST level, Project Office Manager, Industrial Steering Committee, etc)
0	19.04.2018	First draft	CO
1	05.06.2018	Code finished	CO
2	10.07.2018	Theory part draft	CO
3	31.07.2018	Final version	PU

Glossary

No glossary is given.

Link to Tasks

Task number	Work from task carried out	Deviations from task technical content, with motivation and summary of impact
1.4.5	Platt scaling for probabilistic confidence estimation for supervised algorithms developed in Task 1.2 will be tackled in this subtask.	No deviation.
1.2	Scalable supervised machine learning approaches for prediction of the compound activities based on large and imbalanced Chemogenomics datasets	No deviation.

Contents

1 Executive Summary

In this deliverable we provide code for Platt scaling and its integration with other algorithms from the ExCAPE project. As described by some publication ([Lin et al., 2007]), the suggested pseudo-code of the original publication ([Platt, 1999]) may have numerical problems and the optimization algorithm may also be problematic. We therefore create implementations for both pseudo-codes and finally decide to use the implementation of [Lin et al., 2007] for ExCAPE. We finally provide a Python package for the [Lin et al., 2007] implementation.

2 Probabilistic Confidence Estimation

Machine Learning methods often work well for predicting certain target variables, e.g. they do differentiate with high predictive accuracy between two different classes. Usually these methods return some prediction scores, which can be considered to be more or less arbitrary numbers, but are monotonic in a way, such that one class has typically lower scores than the other class. Sometimes these scores are even more constrained, e.g. for logistic regression or deep neural networks with sigmoid activations, where these scores are between 0 and 1. Some methods even try to argue that they are predicting probabilities. Although these prediction scores might fulfill some formal conditions for probabilities, e.g values are between 0 and 1, they are not trustworthy as they can overfit to the training set (see [Mayr et al., 2016]).

Often, there is however the demand for a probabilistic prediction, e.g. in drug discovery only a limited number of drug candidates can be measured and one is possibly interested in the very safe predictions for activity against a certain biological target. The situation can be even vice versa and one might be interested in the unclear predictions in order to possibly gain additional insight.

As the predictions of different targets might be scaled differently, it is further difficult to compare predictions across different targets. This is especially relevant in drug discovery, where often a huge number of possible biological targets are considered. However e.g. a value of 0.7 might indicate activity for a certain target, while for another target it might mean, that it is clearly inactive.

Another demand for probabilistic predictions arises, when predictions of different classifiers should be combined to an ensemble classifier. If a probabilistic ensemble approach as e.g. in [Mayr et al., 2016] is used, there is the need for probabilistic predictions.

3 Platt scaling: Theoretical Background

The core idea of Platt scaling [Platt, 1999] is to fit a logistic sigmoid function to pairs of data points (x_i, y_i), where $x_i \in \mathbb{R}$ is the prediction score of a machine learning algorithm and y_i is the true target. The logistic sigmoid function is parameterized by 2 parameters $A \in \mathbb{R}$ and $B \in \mathbb{R}$ and the fitted probability p, that a certain data point i is a member of the positive class $y_i = 1$ in a binary classification problem, is given by the following formula:

$$p(y_i = 1 \mid x_i) = \frac{1}{1 + \exp(Ax_i + B)} \tag{1}$$

The probability p, that the data point is from the negative class $y_i = 0$, is then:

$$p(y_i = 1 \mid x_i) = 1 - p(y_i = 0 \mid x_i) = \frac{\exp(Ax_i + B)}{1 + \exp(Ax_i + B)} \tag{2}$$

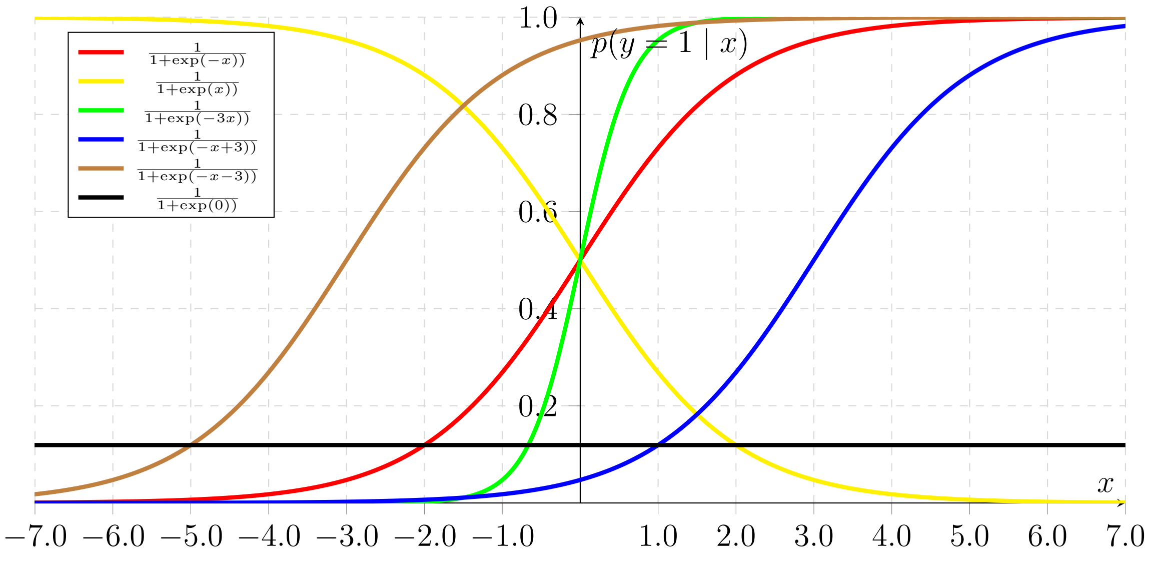

Figure 1: Logistic Function with different parameters

Figure 1 visualizes logistic functions with different parameters influencing the shape and position of the curves. The figure shows, that the fitted probabilitiy $p(y = 1 \mid x)$ can be considered a monotonically increasing or decreasing function from the prediction score x. The sign of the slope parameter A decides, whether the function is increasing or decreasing. As the absolute value of the slope gets higher and higher, the shape of the function, converges to a step function. In the case, the slope is zero, the function is a constant function and returns the same probabilitiy independent of the prediction score. The intercept parameter B moves the curve along the x-axis.

[Platt, 1999] fits the parameters A and B by minimizing the negative log-likelihood of some training data points i (which is noted to be equivalent to a cross-entropy error function):

$$\min_{A,B} -\sum_i (y_i \ \log(p(y_i = 1 \mid x_i; A, B)) + (1 - y_i)\ (1 - \log(p(y_i = 1 \mid x_i; A, B)))) \tag{3}$$

Here an open question is, what the training data points for Platt scaling are. [Platt, 1999] gives some arguments, why it may be not be a good idea to use the same training data points as were used to train the classifier, e.g. for SVMs some training samples are at the margin and therefore are exactly 1, which may be unusual for test samples. Therefore the suggestion is to use hold-out data, that was not used for training the classifier (on which Platt scaling is applied) itself. Alternatively cross-validation approaches are suggested. The approach here would be to produce prediction scores with the trained classifier and then train the logistic sigmoid function with the prediction scores and the true labels.

In order to avoid overfitting, e.g. in the case, where positive and negative samples are perfectly discriminated by the prediction score and A would get very high absolute values, Platt suggests to use regularization.

4 Platt scaling: Implementation and Integration with prediction algorithms

4.1 Introduction

[Platt, 1999] suggested some pseudo-code for training the logistic sigmoid function. This pseudo-code is a model-trust algorithm, which is based on the Levenberg-Marquardt algorithm. [Lin et al., 2007] however argued that there might be some numeric and convergence difficulties, when using Platt's pseudo-code. They suggested a more robust algorithm and gave some related pseudo-code. We created a C program for this pseudo-code, extended it by inserting a statement, that sets $A = 0$, if it is detected to be positive during the iterations as a form of prior knowledge, and checked the functionality on random and generated prediction scores (see below, section 4.2).

4.2 Emprical Tests

We applied some empirical tests to an implementation according to [Platt, 1999] and an implementation according to [Lin et al., 2007] concerning completely random predictions and predictions that come from a machine learning algorithm (see Figures 2 and 3) (with the additional inserted $A = 0$ in both implementations). The blue logistic curve gives the result from the implementation according to [Lin et al., 2007], while the yellow logistic curve gives the result according to [Platt, 1999]. The vertical lines mark the prediction score, for which the probabilities of the fitted curves give 0.5. The red and green points show predictions, where the colour indicates known activity/inactivity respectively. The number of data points for each class is given in the legend. The title of the figures gives the cross entropies for the implementation according to [Platt, 1999] and [Lin et al., 2007] respectively and whether the fitting procedure was successful (in a sense, that the algorithm converged (H=...), etc. and in a more soft version (S=...)). In case it was not successful in a soft way, default parameters may have been returned. In general, with the tested cases and our concrete implementation, it seemed that the implementation according to [Lin et al., 2007] worked better. Exception is the upper left plot in Figure 2, but compared to the other plots in Figures 2 and 3, the absolute difference seemed minor. We therefore consider, the implementation according to [Lin et al., 2007] as the standard implementation for the ExCAPE project.

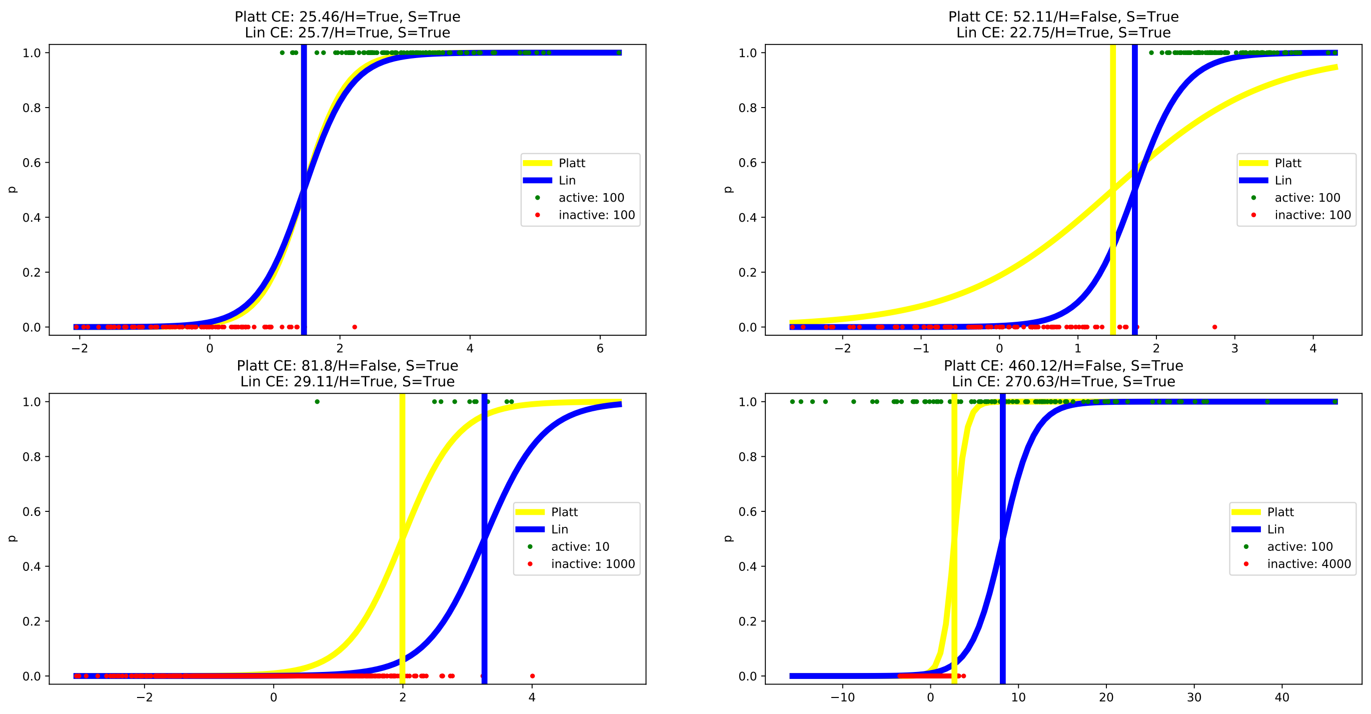

Figure 2: Platt scaling: Implementations according to Platt and Lin on random data

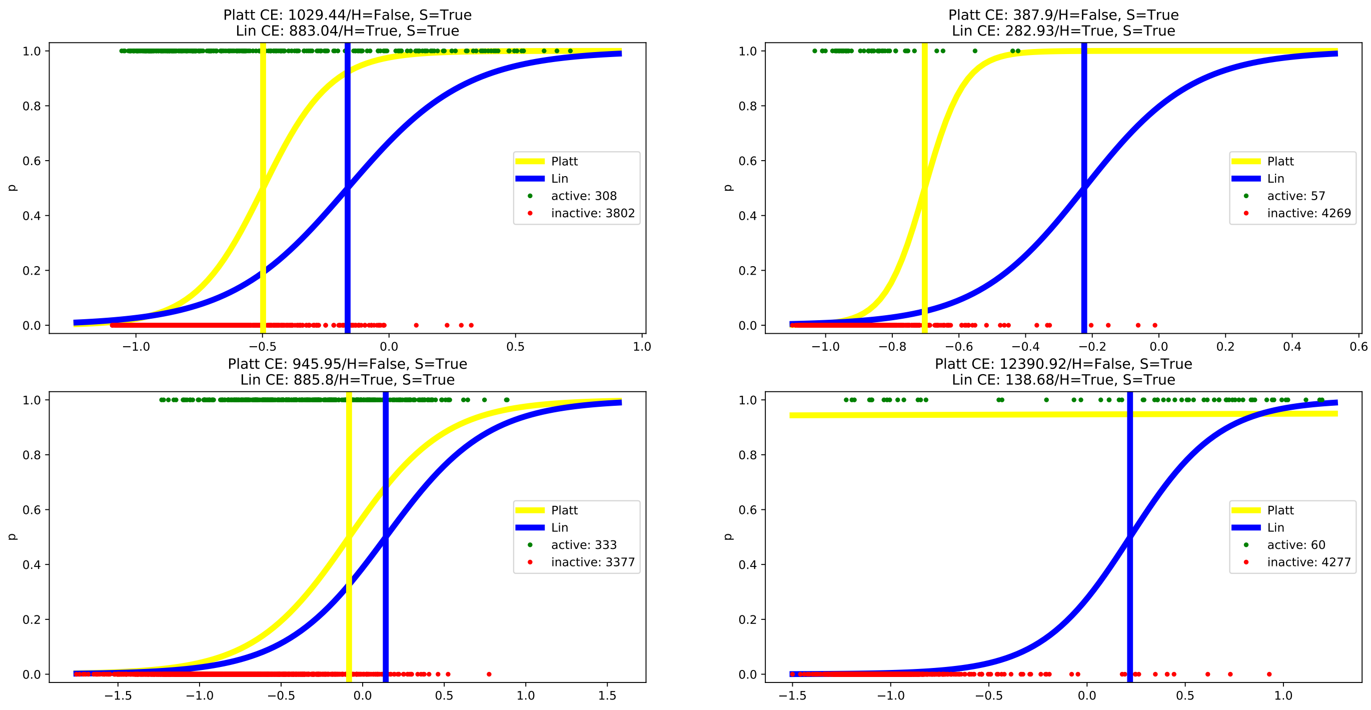

Figure 3: Platt scaling: Implementations according to Platt and Lin on predictions from machine learning algorithm

4.3 Python Package

Using our C program (according to the implemenation suggestions of [Lin et al., 2007]), we created a Python interface and a further Python package (`platt-1.0.tar.gz`) for it, which we attach to this document. Listing 1 shows exemplarily, how this package can be used. `newRes["success"]` returns, whether there is an indication that something went wrong (`false`) or whether this seems not the case (`true`). None-success can e.g. happen, if there are too few data points (e.g. no active sample) and therefore the algorithm did not converge, etc. None-success can also happen, if the algorithm did not converge (concerning the convergence criterion) during the (predefined) maximum number of iterations. Additionally, we included an attribute `newRes["successSoft"]`, to indicate success in a more soft version. Even, if the algorithm itself did not converge, we look, whether the found parameters seem extremely strange (`successSoft=false`) or not (`successSoft=true`), e.g. whether all predicted values on the Platt scaling training data is equal or not. If `successSoft=false`, we change A and B in some cases to default values.

```
import numpy as np
import platt
import matplotlib.pyplot as plt

np.random.seed(123456789)
target=np.concatenate([np.repeat(False, 4000), np.repeat(True, 10)]).astype
    (np.int32)
out=target+np.random.normal(0, 0.3, len(target))

plattRes=platt.plattScaling(out, target)

if plattRes["success"]:
    plattMean=-(plattRes["B"]/plattRes["A"])*plattRes["norm"][1]+plattRes["
       norm"][0]
    plattLeft=((np.log(1.0/0.01-1.0)-plattRes["B"])/plattRes["A"])*plattRes
       ["norm"][1]+plattRes["norm"][0]
    plattRight=((np.log(1.0/0.99-1.0)-plattRes["B"])/plattRes["A"])*
       plattRes["norm"][1]+plattRes["norm"][0]

    xmin=min(min(out), min([plattMean, plattLeft, plattRight]))
    xmax=max(max(out), max([plattMean, plattLeft, plattRight]))
    xvals=np.linspace(xmin, xmax, 100)
    yvals=platt.predictProb(plattRes, xvals)

    plt.plot(xvals, yvals, color='blue', linestyle='-', lw=5)
    plt.plot(out[target>0.5], np.ones(len(out))[target>0.5]*1, color='green
       ', marker='.', lw=0.0, label="active:␣"+str(np.sum(target>0.5)))
    plt.plot(out[target<0.5], np.ones(len(out))[target<0.5]*0, color='red',
        marker='.', lw=0.0, label="inactive:␣"+str(np.sum(target<0.5)))
    plt.axvline(x=plattMean, color='blue', linestyle='-', lw=5)
    plt.ylabel("p")
    plt.show()
```

Listing 1: Platt scaling usage example

4.4 Integration with prediction algorithms

The developed Python package basically works independently of an used prediction algorithm. The integration with prediction algorithms is therefore in the way, that the prediction algorithms just need to provide final prediction scores as well as prediction scores from an hold-out or cross-validation set together with the true labels. Given this information, the platt scaling training function (`platt.plattScaling`) can be applied to prediction scores from the hold-out or cross-validation set and their associated true labels in order to obtain the parameters A and B. Using the platt scaling prediction function (`platt.predictProb`), calibrated probablistic predictions are obtained.

5 Example: Application of Platt Scaling on ExCAPE-DB

As an example, we apply Platt scaling to a Deep Neural Network, that we train on a part of ExCAPE-DB (data_release_v5, which was developed by WP3: Task 3.1.2). Especially, we used the targets corresponding to affinity level 6. We took a certain fixed assignment of samples to folds and trained a network on an outer fold and two inner folds corresponding to this outer fold. Therby we took the provided ECFP features. In order to reduce the number of features we applied some feature selection.

For this demonstration purpose of Platt scaling, we trained a neural network with a fixed network architecture and a fixed set of hyperparameters (improvements may be expected by nested hyperparameter search). Esspecially, we used a network with scaled exponential linear units (see Task 1.2.3, deliverable D1.7 and [Klambauer et al., 2017]). We adapted the dropout functionality according to [Mayr et al., 2018] and used 3 hidden layers with 2048 units. The outer fold had 622729 training samples and 332657 test samples, while the two inner folds had 308169 and 314560 samples for training and testing. The number of targets considered was 526. The sample vs. target matrix is quite sparse, which means that for many entries there is no known ground truth.

We used the test set predictions of the two inner folds to train the Platt scaling parameters for the individual targets. Thereby, we use only those predictions, where we have a known ground truth. The found parameters were then used to apply Platt scaling to the respective outer fold target predictions. In total the search for Platt scaling coefficients was succesful for 516 targets out of the 526 targets.
In the following, we show the results, we get, when applying Platt scaling and then using a threshold of 0.5 for an activity call. Plot 4 shows, that the accuracy is, as one would expect, good, if there is strong imbalancedness, as the classifier may already be very good by just predicting always only one class. There are however several targets, where one might argue, that the classifier has learnt something.

Plot 5 shows the found Platt scaling coefficients over the different targets, ordered by imbalance. A dependence on the imbalance can be observed.

Plots 6 and 7 show the number of positive and negative predictions for each target respectively. As expected, for strong imbalancedness, there are only few positive predictions. However these ones might be interesting.

As concrete examples for found Platt scaling coefficients, we include plot 8 and plot 9, which show scatter plots, that compare, the raw neural network prediction scores with the platt-scaled scores for different imbalance levels and different numbers of samples, where the targets were quite arbitrarily selected (e.g. only had to fulfill that tey are at 25 % of all targets concerning imbalancedness). The found Platt-scaling coeffiencts are given above the plots.

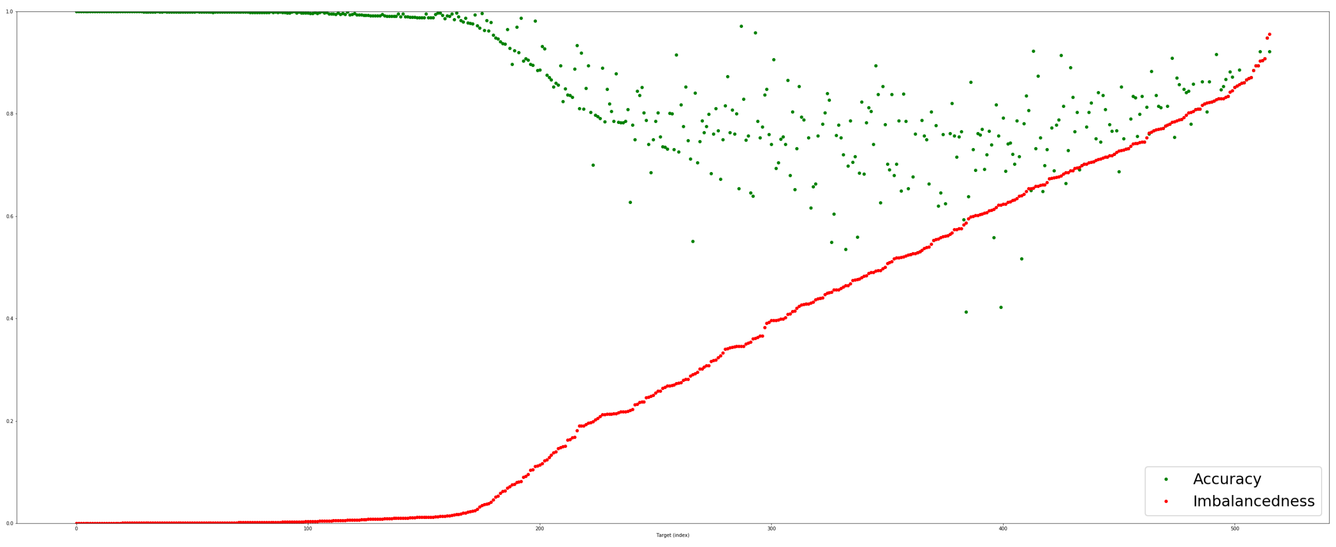

Figure 4: Platt-scaled accuracies and related imbalancedness for each target

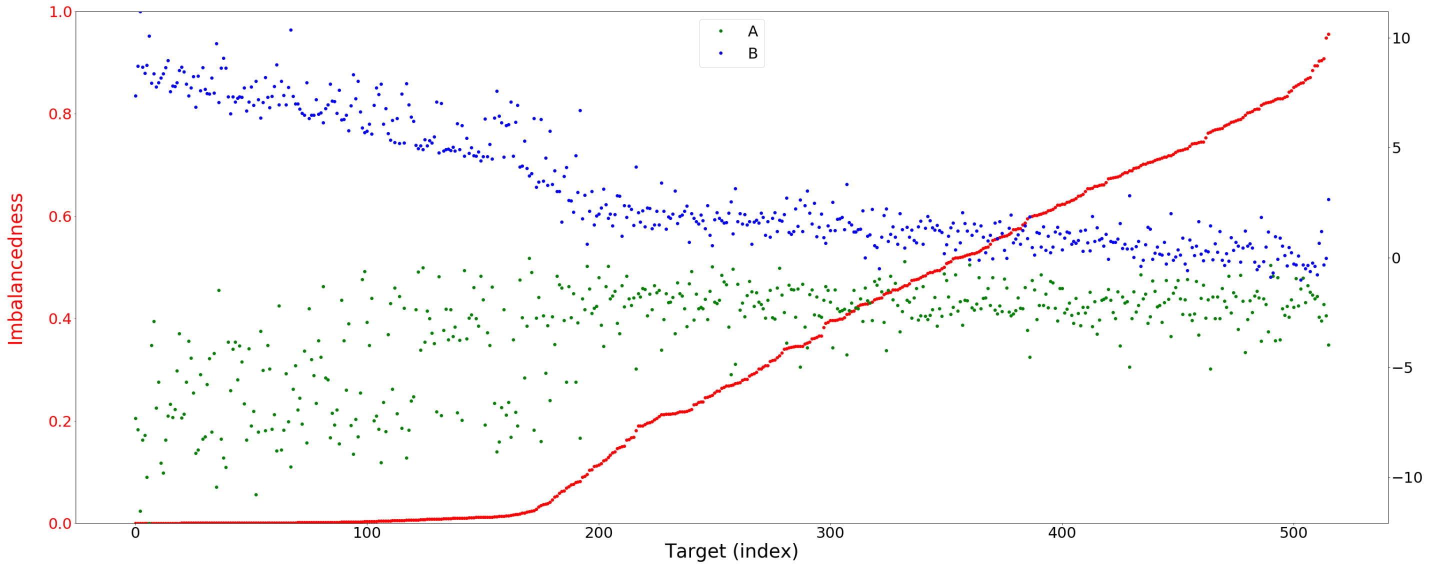

Figure 5: Found Platt coefficients

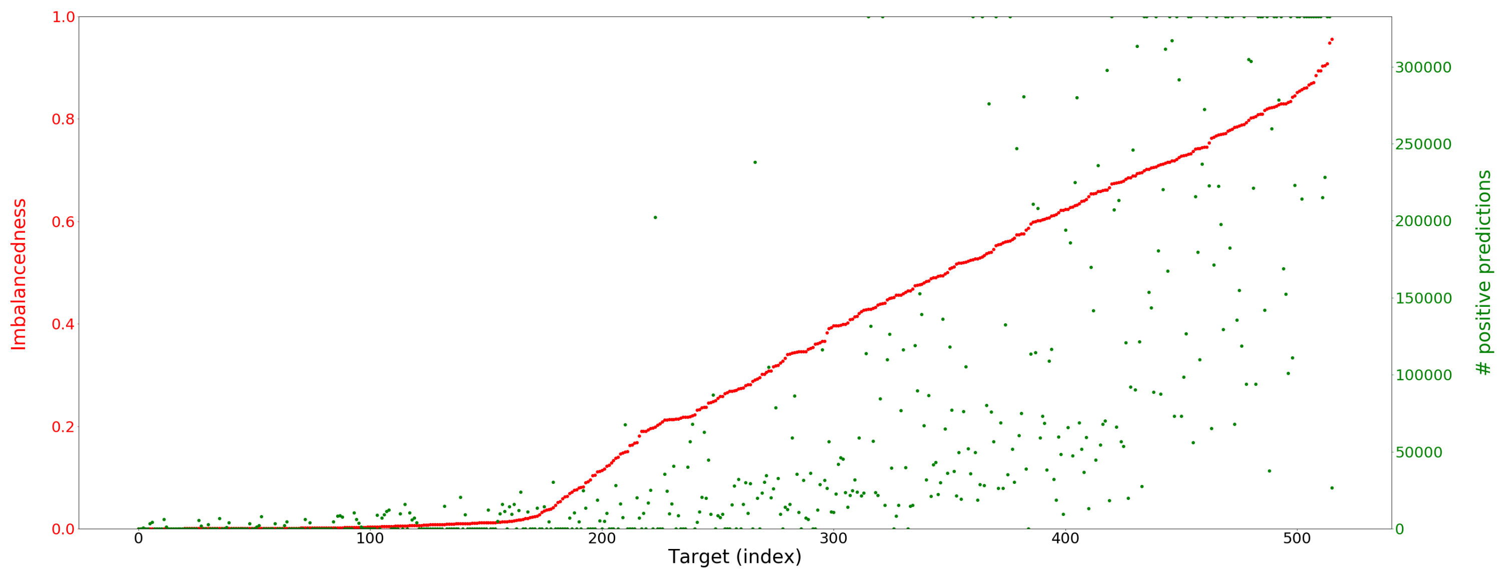

Figure 6: Platt-scaled nr. of positive predictions

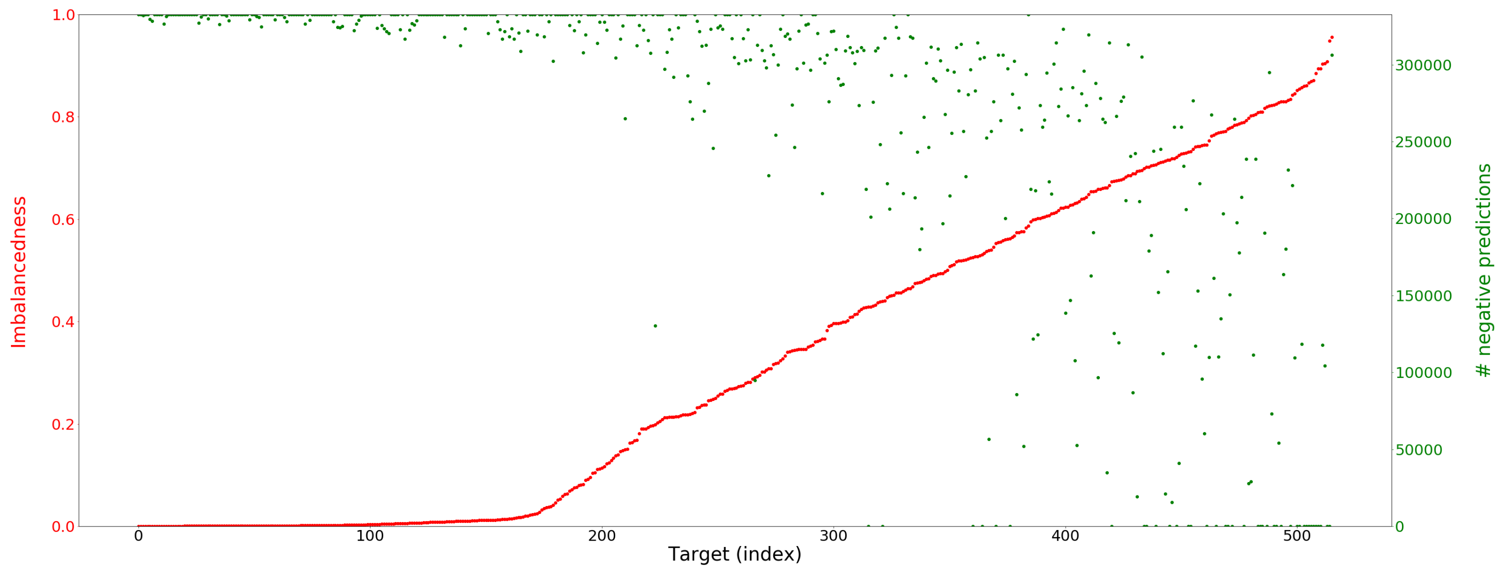

Figure 7: Platt-scaled nr. of negative predictions

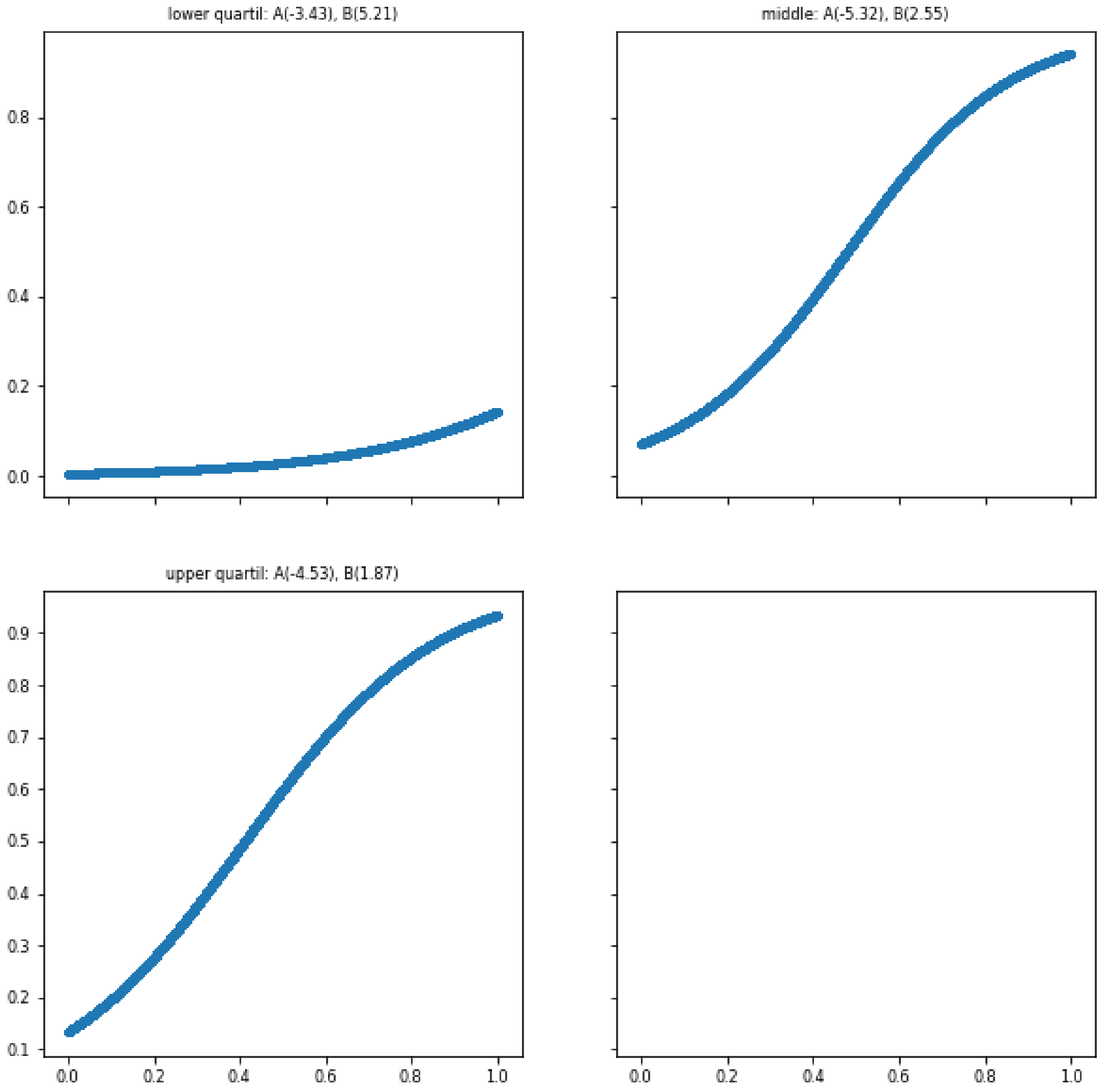

Figure 8: Raw prediction scores vs platt-scaled scores for different imbalance levels

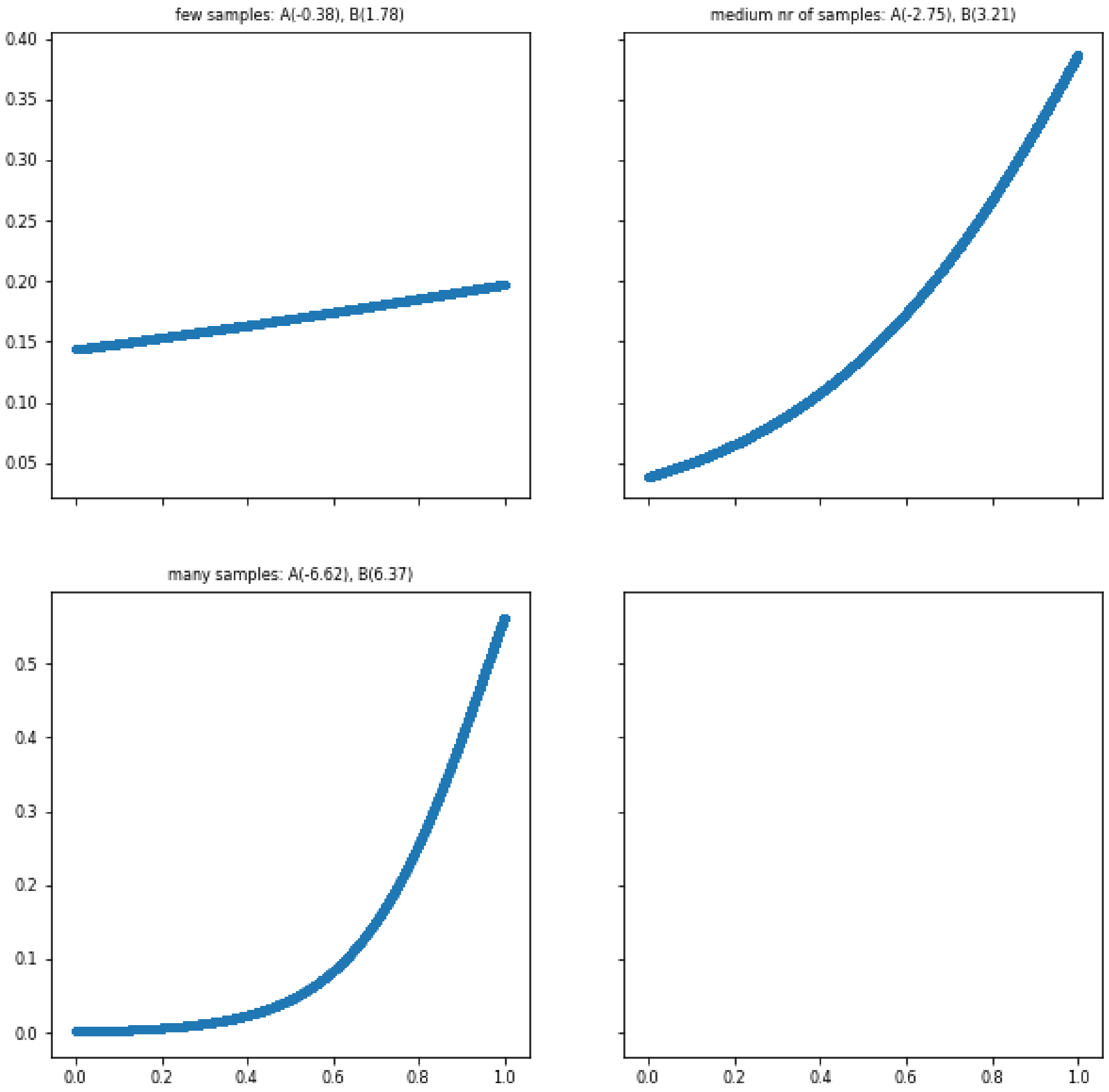

Figure 9: Raw prediction scores vs platt-scaled scores for different number of training samples

6 Conclusion

We created a Python implementation for Platt scaling, applied some empirical tests and decided to use an implementation according to the suggested pseudo-code of [Lin et al., 2007]. This implementation can directly be applied to prediction scores and scores from an hold-out set or cross-validation set from other machine learning algorithms together with the corresponding true labels.

References

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, pages 971–980, 2017.

Hsuan-Tien Lin, Chih-Jen Lin, and Ruby C. Weng. A note on platt's probabilistic outputs for support vector machines. *Machine Learning*, 68(3):267–276, Oct 2007.

Andreas Mayr, Günter Klambauer, Thomas Unterthiner, and Sepp Hochreiter. DeepTox: Toxicity Prediction using Deep Learning. *Frontiers in Environmental Science*, 3(80), 2016.

Andreas Mayr, Günter Klambauer, Thomas Unterthiner, Marvin Steijaert, Jörg K. Wegner, Hugo Ceulemans, Djork-Arné Clevert, and Sepp Hochreiter. Large-scale comparison of machine learning methods for drug target prediction on ChEMBL. *Chem. Sci.*, 9:5441–5451, 2018.

John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large-Margin Classifiers*, pages 61–74. MIT Press, 1999.