An Introduction to one-class classification with the oneClass package

Benjamin Mack

August 17, 2014

Contents

1 One-class classification

The purpose of a **one-class classifier** is identical to the purpose of a supervised binary classifier. New data is classified to belong to one of two classes based on a classification model trained from labeled samples, for which the class membership is known. In contrast to the supervised classifier, the training data of the one-class classifier only contains labeled samples from the class of interest, i.e. the **positive class**. In the case of the binary classifier also the other class, or the **negative class**, has to be represented with the training set. Collecting a representative training set for the negative class can be very costly and time-consuming due to the fact that the negative class is the aggregation of all other classes without the positive class. Thus, a one-class classifier is particularly useful when only one or a few classes have to be mapped and when the acquisition of representative labeled data for the negative class is expensive or not possible at all.

The convenience of not requiring negative training data comes at a price. One-class classification is challenging due to the limited information contained in the training set. Unlabeled training data can be necessary for some classification problems in order to learn more accurate predictive models. However, the process is still uncertain and the classification outcome has to be treated with caution.

The package oneClass shall serve the requirements of **two potential users**, the analyst and the developer. These are extrem characters and in reality one will usually be located somewhere in between. The **analyst** is faced with a particular one-class classification problem, i.e. a set of positive training samples and the unlabeled data to be classified. It is assumed that no complete and representative test set is available for the purpose of validation and testing. In such a situation a careful evaluation of the classification outcome based on the available (positive and unlabeled) data is required in order to select the most promising final model and threshold. The function **trainOcc**() is a wrapper for carets[1] [1] **train**() function which is called with one of the one-class classification methods implemented in oneClass (see Section4). **trainOcc**() returns an object of class `trainOcc` which inherits from class `train`. Thus, the extensive infrastructure of caret is available, such as parallel processing and different methods for pre-processing, resampling, and model comparision. Furthermore, the oneClass infrastructure comprises one-class classification specific methods, such as performance metrics based on positive and unlabeled data (see Section1.2) and diagnostic plots, which further support the handling of the one-class classification methods and particularly understanding their outcome in the absence of representative test data. In Section4 a one-class classification task is solved step by step in order to show which outcomes should be screend by the analyst in order to detect deficient settings, input data, or model outcomes and improve the model if necessary (Section 4). Hopefully the package is helpful for solving one-class classification problems more effectively and conveniently.

The **developer** is interested in the developement of new or optimization of existing methods. The package oneClass builds upon the powerful package caret and tries to adapt its philosophy. The package caret allows the user to embed own **custom functions** and **performance metrics** in the rich infrastructure of the caret package. Furthermore, convenient functions are available for testing the classifier outcome with positive/negative (PN) test sets (Section 5).

1.1 One-class classifiers

The oneClass package is a user-oriented environment for analyzing one-class classification problems. It implements three commonly used classifiers, the one-class SVM (OCSVM) [2] and biased SVM (BSVM) [3, 4] via the package kernlab [5], and a one-class classifier based on calculating a density ration with a maximum

[1] See also `http://caret.r-forge.r-project.org/`.

entropy approach (MAXENT) [6, 7] via the package dismo [8]. As mentioned before these classifiers are implemented as custom functions for **train**() for the package caret [1]. The one-class SVM is a P-classifier, i.e. the classification model is trained with positive samples only. Nevertheless, unlabeled samples can be used to calculate PU-performance and support model selection. The biased SVM and Maxent are PU-classifiers, i.e. they are trained on positive and unlabeled data. Computationally, P-classifiers are usually computationally less complex than PU-classifiers. However, PU-classifiers often perform better in terms of classification accuracy because with the information contained in the unlabeled training data models can be build which better fit the particular classification problem to be solved (Section **??** and Section 5).

1.2 PU-performance metrics

As with other pattern recognition and machine learning algorithms, it is crutial to parameterize the one-class classification methods carefully. The parameterization or model selection is usually performed via a grid-search. The grid points are combinations of discrete parameter values. The performance of the model is evaluated for all grid points and the parameters are chosen which optimizes the performance metric. In the case of supervised classification the performance metric, such as the overall accuracy or kappa coefficient. Such metrics have to be derived from complete validation data comprising the positive and negative class. They are therefore unidentifiable in a one-class classification situation.

Some performance metrics have been defined which can be derived from positive and unlabeled data (PU-performance metrics). From PU-data we can estimate two interesting probabilities: From the positive training samples we can estimate the probability of classifying a positive sample correctly, also known as the true positive rate (TPR). From the unlabeled samples we can estimate the probability of classifying a sample as positive, which we call the probability of positive prediction (PPP). Given we have two models with the same TPR but with different PPP it is valid to say that the model with lower PPP is more accurate because the TPR is the same but the false positive rate is necessarily lower. This is conclusion is however only valid if the TPR can be estimated accurately. Furthermore, it does not solve the question which of a set of models is the best when both the TPR and PPP differ.

The PU-performance metrics `puF` related to the F-score [9] and `puAuc` related to the area under the receiver operating curve [7] try to give an answer. Both have been shown proved shown in the cited references to be suitable for ranking models based on PU-data. It is impossible to say which metrics is better in a particular situation. Note that `puF` is based on the TPR and PPP which are derived for a particular threshold, here zero. It is possible that the threshold with which TPR and PPP are estimated is not optimal and thus `puF` can be low even though the model has high discrimnative power. Instead `puAuc` is calculated independent of a particular threshold. In other words, it calculates the performance over the whole range of possible thresholds. Thus it also considers thresholds which are definitively unsuitable which might also lead to misleading results [10].

Based on these thoughts and experience it is not recommended to trust these rankings blindely, particularly in challenging classification problems, e.g. with a small amount of positive training samples or an eventually unsuitable set of unlabeled samples. We can reasonably assume the the PU-performance metrics are positively correlated with PN-performance metrics, such as the overall accuracy of the kappa coefficient. But the relationship can be noisy and in the worst case this could mean that the model with the highest PU-metric has very poor discriminative power. They should rather be used as helpers for selecting a couple of candidate models, which are examined more thoroughly. Furthermore, because the performance metrics do not proved information on the absolute accuracy, such as the overall accuracy, they do not reveal if the

model is poor even though it might be the best one of all evaluated models, e.g. because non of the specified parameter settings are suitable.

Therefore, it can be useful to also investigate the true positive rate (TPR), and the probability of positive prediction (PPP). The quantities are implemented in the function **puSummary**() and calculated by default for all models evaluated during model slection.

2 Installation

The package can be downloaded from GitHub (`https://github.com/benmack/oneClass`). It can be installed from within `R` when the package `devtools` is loaded:

```
require(devtools)
install_github('benmack/oneClass')
```

3 An illustrative synthetic data set

In the following the package is demonstrated by means of the synthetic data set consisting of two banana shaped distributions. The bananas data is stored as raster data (Figure 1), where y is a one-band raster with the class patches, i.e. what we want to find out when performing one-class classification with remotely sensed data. x are the features or predictors based on which the classification has to be build.

```
require(oneClass)
require(raster)
require(RColorBrewer)

data(bananas)
plot(bananas$y)
plot(bananas$x, col=brewer.pal(9, "Greys"))
```

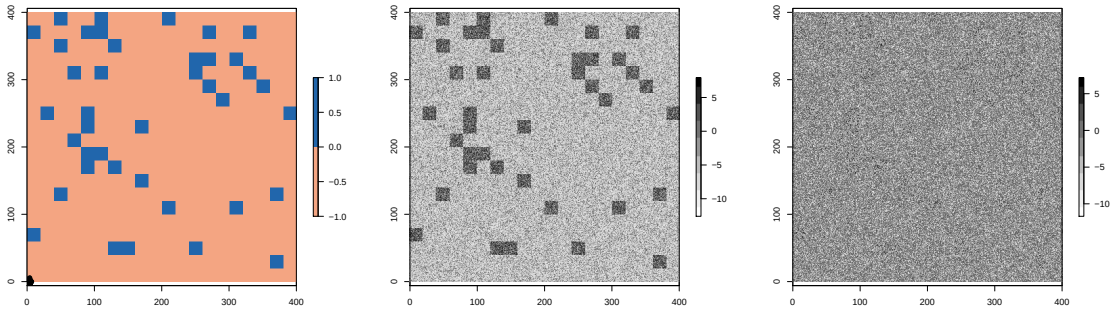

Figure 1: The synthetic data set. The goal of the classification is to estimate the (in reality) unknown class membership y (left) based on observed predictor variables or features, here an image with two bands (middle, right).

In one-class classification a set of positive labeled samples is available for training the classifier. Furthermore, unlabeled samples are used, which are usually a random sample of the whole data. Such a PU-training set is stored in bananas. In order to make the results comparable we create a list with resampling partitions(`tr.index`). Note that if we do not explicitly pass such a list the partitions differ because they are generated randomly every time **trainOcc**() is run again. Additionally, we generate a test data set consisting of 1.000 random samples of the image.

```
seed <- 123456
tr.x <- bananas$tr[, -1]
tr.y <- puFactor(bananas$tr[, 1], positive=1)
set.seed(seed)
tr.index <- createFolds(tr.y, k=10, returnTrain=TRUE)
set.seed(seed)
te.i <- sample(ncell(bananas$y), 1000)
te.x <- extract(bananas$x, te.i)
te.y <- extract(bananas$y, te.i)
```

The two-dimensional synthetic data set can be visualized in the feature space (Figure 2). A one-class classifier is supposed to learn from P- or PU-data in order to find an optimal model for separating PN-data. Recall that even if we use a P-classifier (e.g. the one-class SVM), which is only trained with P-data, it is recommendable to make use of PU-data for model selection, as we will do in the next section.

```
plot(tr.x[tr.y=="pos", ], pch=16 )
plot(tr.x, pch=ifelse(tr.y=="pos", 16, 4) )
plot(te.x, pch=ifelse(te.y==1, 16, 1) )
```

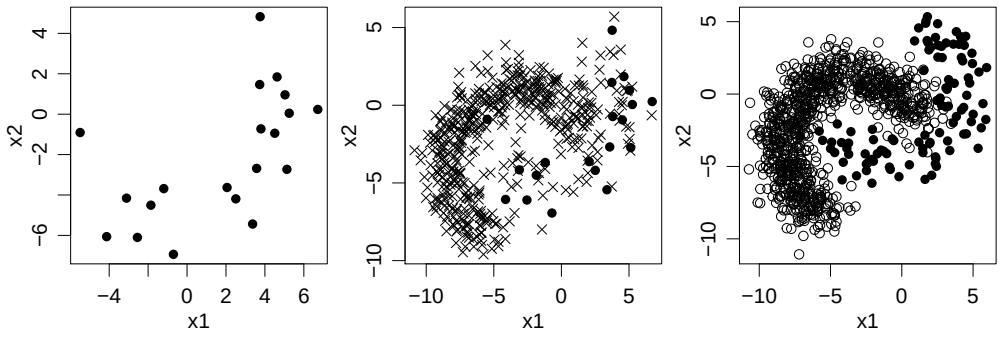

Figure 2: P-training data (LEFT), PU-training data (MIDDLE) and PN-test data (RIGHT). Solid circle: positive, cross: unlabeled, circle: negative.

4 Finding a suitable one-class classification model

4.1 One-class SVM with default settings

Let us first try to solve the classification task using the OCSVM and the default settings. We simply pass the training data to **trainOcc**(). A ten-fold cross validation is performed over a pre-defined grid and the model

with the highest puF value is selected and trained on the whole data. Then we predict the whole image with the fitted model. By default the prediction returns the continuous decision values. In the case of the OCSVM zero can be considered the natural threshold for the derivation of a binary classification result. (Note, that this does not mean that zero is always the optimal threshold!)

```
ocsvm.fit <- trainOcc(x=tr.x, y=tr.y, method="ocsvm", index=tr.index)
ocsvm.pred <- predict(ocsvm.fit, bananas$x)
ocsvm.bin <- ocsvm.pred>0
```

We need to evaluate the classification result without PN-reference data. Several plots can help us to assess the classification results (Figure 3).

```
plot(ocsvm.pred, col=brewer.pal(9, "RdBu")); plot(ocsvm.pred>0, col=brewer.pal(9, "RdBu")[c(2,9)])
hist(ocsvm.fit, ocsvm.pred, th=0); featurespace(ocsvm.fit, th=0)
```

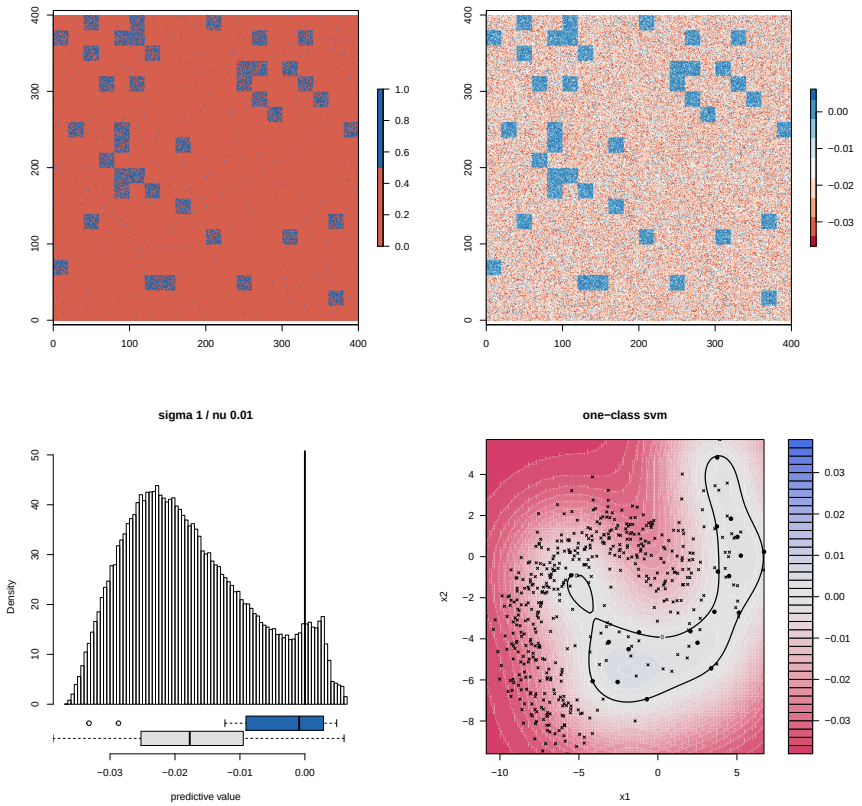

Figure 3: One-class SVM results with default settings. TOPLEFT: Binary predictions given the threshold 0. TOPRIGHT: Continuous predictions, i.e. the distance to to hyperplane. BOTTOMLEFT: Diagnostic densities plot with the continuous predictions (histogram) also shown in the topright image and the distributions of the cross-validated positive (blue boxplot) and unlabeled (grey boxplot) training samples. BOTTOMRIGHT: One-class SVM model in the feature space with the threshold (black line) applied to derive the binary classification in the topleft image.

The diagnostic densities plot (Figure 3 bottomleft) is a very simple yet very informative plot. As long as no complete and representative test data is available, it is recommended to always analyze the plot carefully before accepting any one-class classification result. When interpreting this plot keep Bayes' Theorem in mind.

The diagnostic plot shows evidence that the class of interest can be separated moderately from the negative class. The positive hold-out predictions (blue boxplot, from now on positive predictions) are located at the upper tail of the predictive values. Their median corresponds relatively well with the local maximum of the classifier output histogram. It can therfore be concluded that (most of) the data building the right data cluster belongs to the positive class. However, it is clear that the positive data overlaps significantly with the negative class. Thus, a significant amount of mis-classification has to be accepted, wherever we apply the final binarization threshold. If there was no or just a very small class overlap, a very low density region would exist between two (very well separable) data clusters, one build up by the negative class at lower predictive values and the other build up by the positive class.

Ideally the distributions of the cross-validated positive predictions (blue boxplot) and the whole unlabeled data (histogram) correponds in the non-overlapping region of the output space, i.e. the high predictive valeus. Here, we can observe a bias which is often the case when the positive training data is relatively small. The distiribution of the positive predictions seem to be shifted towards lower predictive values. This is expectable due to the relatively small positive training set. Here we can observe the model in the feature space and, knowing the functioning of support vector machines, can imagine that most of the training samples are important for the support of the decision boundary. During cross-validation the predictive values for the training samples are derived from models trained without the samples to be predicted which, given the number of samples and complexity of the model, obviously leads to biased predictions. It is important to keep this issue in mind when interpreting diagnostic plots of classification results derived from higher dimensional input data, i.e. where the model in the feature space can not be visualized.

Also the salt-and-pepper like image might leads to the conclusion that there is a significant overlap. However, it is usually more easy to get an realistic idea about the degree of separability from the dignostic plot then from the image alone. This is particularly the case with real world data sets.

The feature space plot also reveals the weakness of the OCSVM (and any other classifier which is trained on the positive data only). (Of course, the feature space can not be visualized when working with high dimensinal data and only serves for a better understanding here.) The decision boundary should be tight in those regions where the negative class overlaps with the positive class to avoid a high amount of false positives. Note particularly the single trainign sample located in the inner curvation of the negative class/banana (approximately at (x1/x2)=(-5/-1)). This point is obviously located in a region where the positive class has low density but the negative class has very high density. A P-classifier can not learn anything about the density of the negative class while a PU-classifier can derive such information for the unlabeled trainign samples. Here the decision boundary should be less tight in regions where no negative data lives, e.g. at the right side of the feature space. Then the positive data from low density areas could also be classified correctly without any additional fals positive classifications. But due to the fact that the OCSVM model has only seen positive data during the training it can not be aware of such differences. We will see later that a PU-classifier, such as the BSVM can learn this from the unlabeled data.

4.2 Revising the parameter space

By now let us first try to find a better OCSVM model. Note that the analytic procedure is the same for other methods. First, it is a good idea to proof if the grid of parameter values has been reasonable. We can use methods from the package caret to visualize the dependency of the PU-performance of the parameters (Figure 4 left). The grid shows that the `puF` drops sharply at sigma values smaller and greater than one and highest values at the higher end of the `nu` range. It is thus possible that a finer grid around `sigma==1` contains more powerful models. `nu` values below 0.01 do not make sense but if so, it would also be a good idea to extend the grid in this direction of the parameter space.

Let us re-run **trainOcc**() with a customized grid and visualize it again (Figure 4 right). Note that there are now several models reaching higher `puF` values (max. `puF` in customized/default grid: 5.81/ 3.62)

```
tuneGrid <- expand.grid( sigma = seq(.1, 3, .1), nu = seq(.05, .5, .05) )
ocsvm.fit <- trainOcc(x=tr.x, y=tr.y, method="ocsvm", tuneGrid=tuneGrid,
                      index=tr.index) # same partitions
ocsvm.pred <- predict(ocsvm.fit, bananas$x)
plot(ocsvm.fit, plotType="level")
```

```
trellis.par.set(caretTheme())      # nice colors from caret
plot(ocsvm.fit.def, plotType="level") # see ?plot.train for other plot types
plot(ocsvm.fit, plotType="level")
```

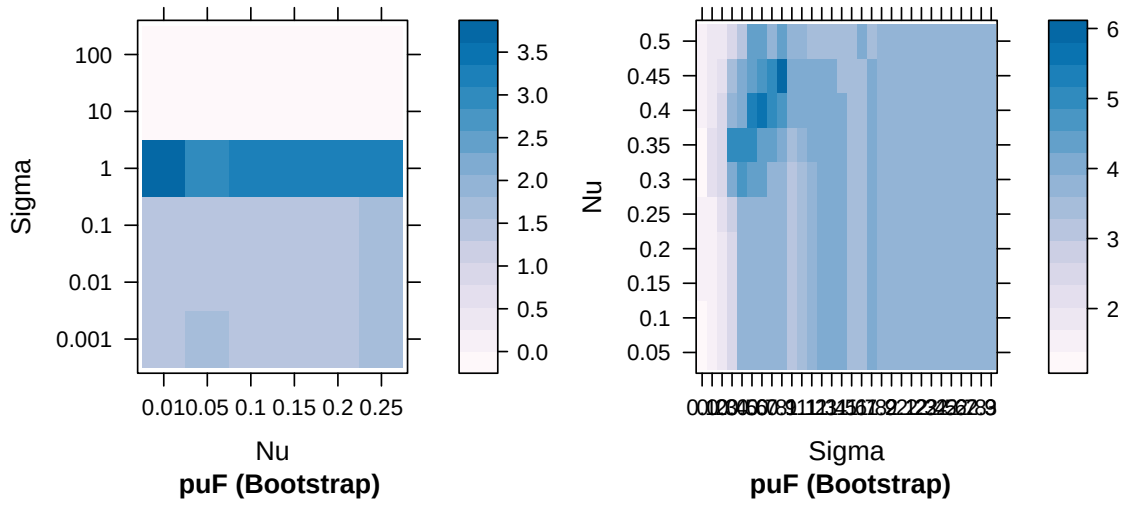

Figure 4: The PU-performance achieved with the one-class SVM models (differing in the Nu and Sigma parameters) for the default parameter grid (LEFT) and the refined parameter grid (RIGHT).

The diagnostic density plot (Figure 5) looks more convincing because the positive data cluster looks more distinctive from the rest of the data. It also reveals that the threshold is unlikely to be located at an optimal location. With Bayes' Theorem in mind and assuming that the positive data mainly builds the right

cluster, we would intuitively set the threshold in the middle or the minimum of the low density area. It is reasonable to believe that the amount of false positives increases less than the amount of true positives when moving the threshold to this point.

Note that it is more difficult to evaluate and compare the two models visulized in Figure 3 and Figure 5) based on the predicted images alone. The diagnostic plot provides additional information which is also easy to understand and interpret.

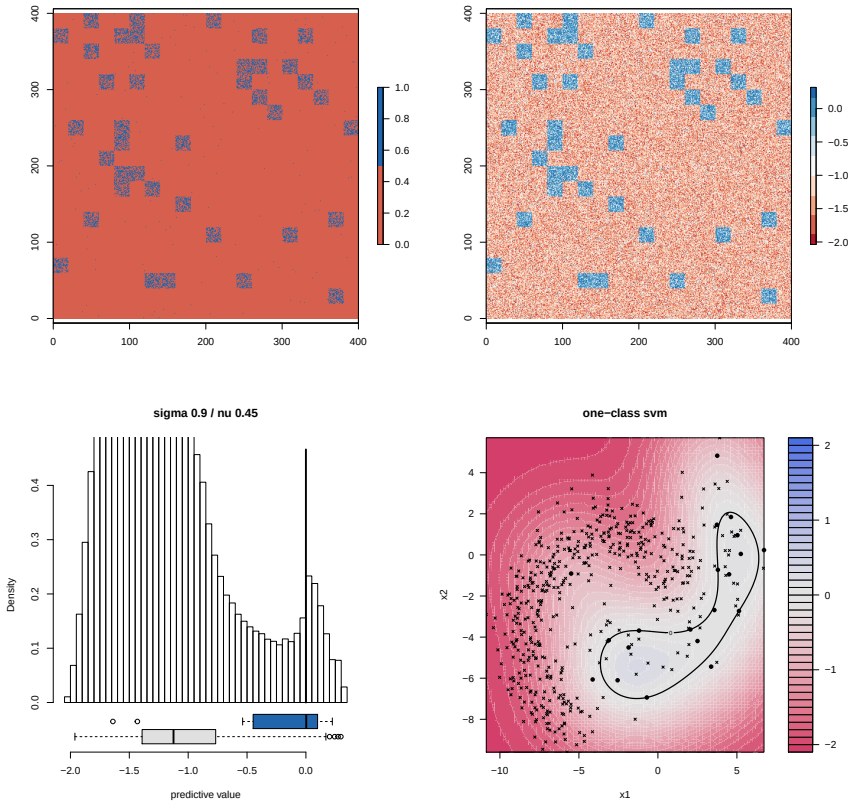

Figure 5: One-class SVM result given the model with maximum puF in the refined parameter space. TOPLEFT: Binary predictions given the threshold 0. TOPRIGHT: Continuous predictions. BOTTOMLEFT: Diagnostic densities plot. BOTTOMRIGHT: One-class SVM model in the feature space.

4.3 Manual model selection (OCSVM)

So far, we trusted the PU-performance metric and selected the model with the highest PU-performance for further evaluation. As has been mentioned in Section 1.2 that the PU-performance metrics are usually in a positive but sometimes noisy relationship with the unidentifiable PN-performance metrics.

Eventually, better models can be determined by comparing the diagnostic plot of other models. Of course, it is too elaborative to compare all models like that because it requires to predict all (or a large fraction) of the unlabeled data to construct the histogram and there might be a hugh amount of models considered for model selection, e.g. 300 in the refined grid defined in Section 4.2. Therefore, a strategy is required which helps to select models to be compared.

First, the PU-performance metric can be used. It is plausible to select the models with highest PU-performance metrics first because we expect a positive relationship between the PU-performance metric and the PN-performance metric.

The model selection table stores the model parameters and performance metrics and can be used for such a ranking. It is stored in `ocsvm.fit$results` and is is printed to the console when printing the `trainOcc` object (which is not done here because the output is too long). A sorted version of this table can also be derived and used to compare the characteristics of the models, e.g. with highest `puF` and `puAuc` values. Below we see the table entries for the ten models with highest `puF` (left) and `puAuc` (right) values. The numbers at the very left are the row *names* and correspond to the rows in the ('ocsvm.fit$results') data frame. The first two columns show the model parameters (`sigma` and `nu`), followed by the estimated true positive rate (TPR), the probability of positive prediction (PPP), and the performance metrics (see also Section 1.2).

```
sort(ocsvm.fit, digits=2, rows = 1:10, cols =1:6)
sort(ocsvm.fit, digits=2, by = 'puAuc', rows = 1:10, cols =1:6)
```

```
##    sigma   nu  tpr   ppp puAuc puF
## 89   0.9 0.45 0.55 0.073  0.86 5.8
## 68   0.7 0.40 0.55 0.079  0.86 5.6
## 58   0.6 0.40 0.55 0.083  0.87 5.2
## 78   0.8 0.40 0.50 0.075  0.86 5.2
## 79   0.8 0.45 0.50 0.073  0.87 5.0
## 47   0.5 0.35 0.55 0.090  0.86 4.9
## 57   0.6 0.35 0.55 0.087  0.86 4.9
## 37   0.4 0.35 0.60 0.100  0.86 4.9
## 88   0.9 0.40 0.45 0.073  0.86 4.7
## 69   0.7 0.45 0.50 0.079  0.87 4.6
```

```
##     sigma   nu  tpr   ppp puAuc puF
## 70    0.7 0.50 0.45 0.063  0.87 4.5
## 50    0.5 0.50 0.40 0.069  0.87 3.1
## 100   1.0 0.50 0.40 0.062  0.87 3.7
## 60    0.6 0.50 0.45 0.065  0.87 4.2
## 79    0.8 0.45 0.50 0.073  0.87 5.0
## 90    0.9 0.50 0.45 0.063  0.87 4.5
## 110   1.1 0.50 0.40 0.063  0.87 3.7
## 120   1.2 0.50 0.35 0.063  0.87 3.4
## 130   1.3 0.50 0.35 0.063  0.87 3.4
## 48    0.5 0.40 0.50 0.085  0.87 4.2
```

Note that the two PU-performance metrics do not return the same ranking. The four models with highest `puF` values (models 89, 68, 58, and 78) are not among the ten models with highest `puAuc` values. We can now investigate the diagnostic plots of manually selected models and models which eventually leads us to a better model. The diagnostic plot of another model can be easily created by 1) updating the `trainOcc` object such that the final models is the manually selected one, 2) predicting the unlabeled samples with the updated model, 3) creating a new diagnostic plot. Here, we create the dignostic and feature space plots for the 3 top ranked models according to `puF` in order to confirm the model selected by `puF` or eventually select a model that looks more suitable.

```
# comparable y-axis range
```

```
ylim=c(0, 0.4)

ocsvm.fit <- update(ocsvm.fit, modRow=89); ocsvm.pred <- predict(ocsvm.fit, bananas$x)
hist(ocsvm.fit, ocsvm.pred, th=0, ylim=ylim); featurespace(ocsvm.fit, th=0)
ocsvm.fit <- update(ocsvm.fit, modRow=68); ocsvm.pred <- predict(ocsvm.fit, bananas$x)
hist(ocsvm.fit, ocsvm.pred, th=0, ylim=ylim); featurespace(ocsvm.fit, th=0)
ocsvm.fit <- update(ocsvm.fit, modRow=58); ocsvm.pred <- predict(ocsvm.fit, bananas$x)
hist(ocsvm.fit, ocsvm.pred, th=0, ylim=ylim); featurespace(ocsvm.fit, th=0)
```

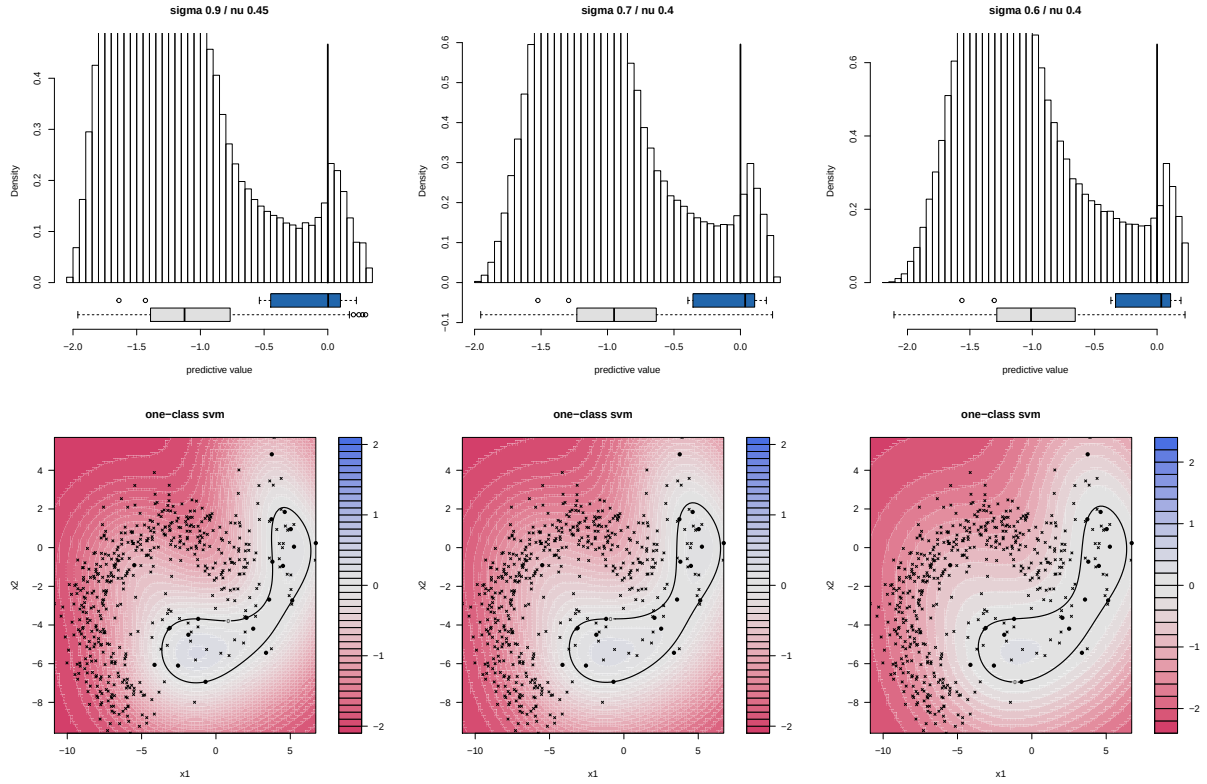

Figure 6: Diagnostic plots (UPPER row) and feature space plots (BOTTOM row) of the three top ranked OCSVM model according to puF (from LEFT to RIGHT in decreasing order).

All diagnostic plots seem relatively similar and there is no reason here to prefer another model against the one seelcted by default. Of course, in practice we might investigate more models, but in this example this would not lead to a more plausible model. Therefore, we re-set the final model to the one selected by maximizing puF.

```
ocsvm.fit <- update(ocsvm.fit, modRow=89)
ocsvm.pred <- predict(ocsvm.fit, bananas$x)
```

It can also be a strategy to investigate the PPP vs. TPR plot (Figure 7) for chosing a subset of models considered for more detailed analysis. As we can see from the ranking in the tabels above the highest

ranked models are very similar in terms of the TPR and PPP. It has been discussed in Section 1.2 that among two models with the same TPR but differnt PPP the one with lower PPP should to be prefered. It might therefore be unnecessary to evaluate model 68. It is the model with second highest `puF` but the models are likely very similar because boteh have a TPR of 0.55 and only differ by 0.006 in the PPP. Therefore, given the PPP vs. TPR plot we would group the models with similar TPRs and select from the different groups the model with the lowest PPP. (The models can be identified by setting the parameter identify=TRUE.)

```
plot_PPPvsTPR(ocsvm.fit, identify=FALSE)
```

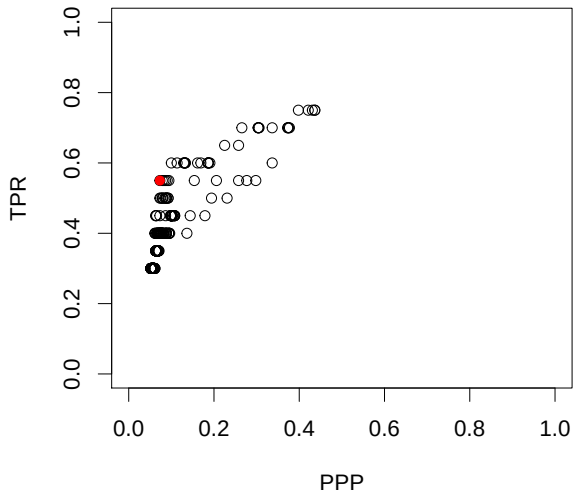

Figure 7: The PPP vs. TPR plot. The red dot highlights the selected model, which is here the one with the highest puAuc.

So far we have shown the most important functions and plots required for model selection and model evaluation. The same procedure can be followed when working with one of the other classifiers. In the next Section we will solve the classification problem with BSVM and MAXENT, i.e. PU-classifiers, and compare the outcome with the OCSVM.

4.4 The PU-classifiers: BSVM and MAXENT

It has been argued in Section 1.1 that PU-classifiers often outperforme P-classifiers. In the following we also inverstigate the dignostic plots and feature space plots of the three highest ranked BSVM (Figure 8) and MAXENT (Figure 9) models.

Note that by default the `puF` is maximized for selecting the BSVM model while the `puAuc` is used to select the MAXENT model. Furthermore, the MAXENT model does not have a "natural" threshold such as the SVM methods. In order to derive the PPP, TPR and `puF`, the threshold maximizing the sum of sensitivity and specificity (maxSSS) is used as suggested in [11]. This and other commonly used thresholds can be derived from a fitted MAXENT model.

First we run **trainOcc**() with the default method, i.e. the BSVM and plot the diagnostic plots and feature space plots for the six highest ranked models (Figure 8). Comparing the six diagnostic plots of Figure 8 we would select the fourth, fifth, or sixth model. In these models there is a wide low density region between the two main clusters of data. Here, we select the fourth model because the right cluster corresponds well with the positive hold-out predictions, i.e. the blue boxplot, and a part of one outlier they are all located right to the default threshold.

```
bsvm.fit <- trainOcc(x=tr.x, y=tr.y, index=tr.index)
order(-bsvm.fit$results$puF)[1:6]

## [1] 30 38 37 79 80 86
```

```
# comparable y-axis range
ylim <- c(0,.15)

hist(bsvm.fit, bsvm.pred, th=0, ylim=ylim)
featurespace(bsvm.fit, th=0)

# update, predict, plot for following 5 models
bsvm.fit <- update(bsvm.fit, modRow=38); bsvm.pred <- predict(bsvm.fit, bananas$x)
hist(bsvm.fit, bsvm.pred, th=0, ylim=ylim)
featurespace(bsvm.fit, th=0)

### ...
```

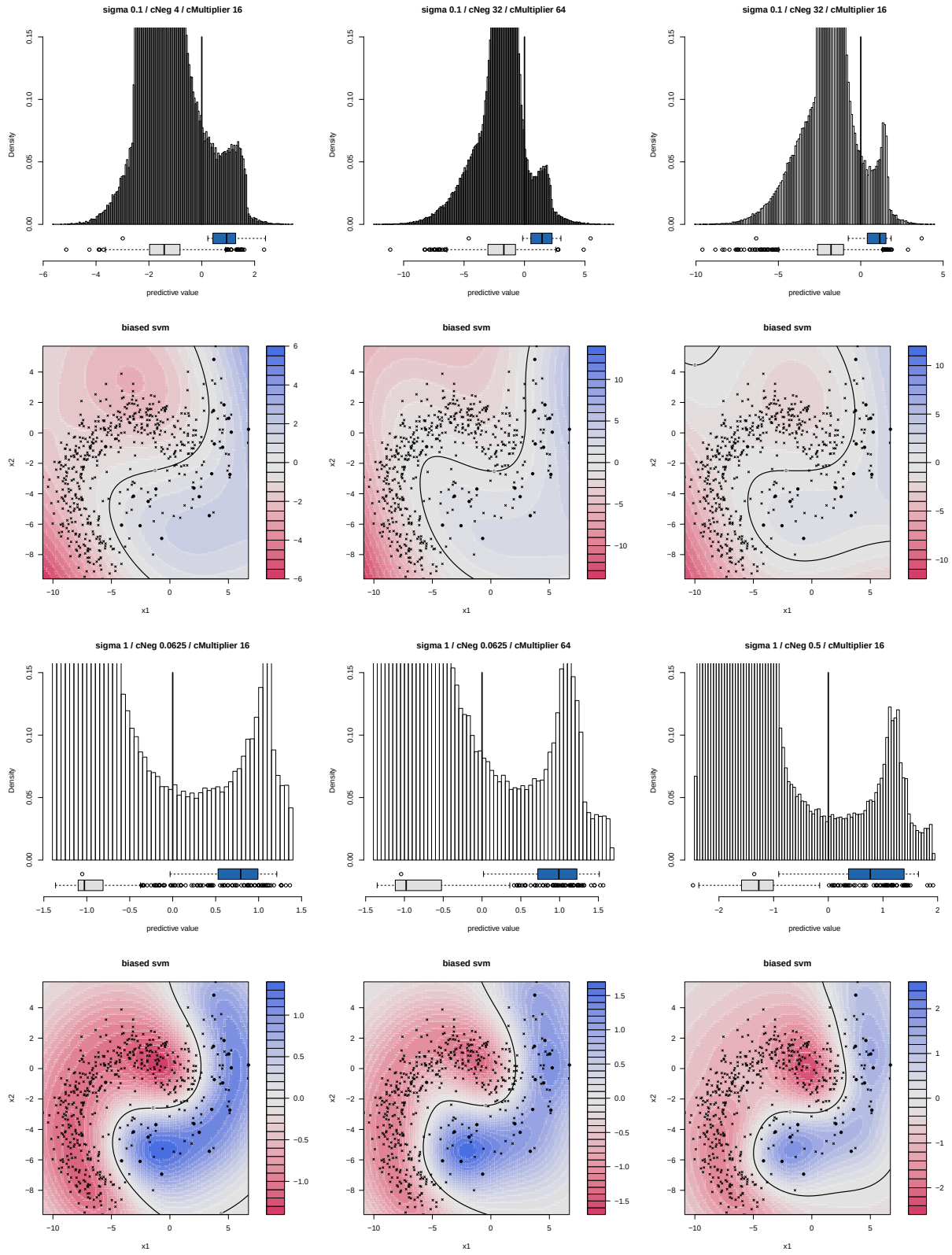

Figure 8: Diagnostic plots (1sr & 3rd row) and feature space plots (2nd & 4th row) of the six top ranked BSVM model according to puF (from LEFT to RIGHT in decreasing order).

```r
# set the final model
bsvm.fit <- update(bsvm.fit, modRow=79)
bsvm.pred <- predict(bsvm.fit, bananas$x)
```

We do the same with the MAXENT method but in consideration of the diagnostic plots we will not change the model selected by maximizing `puAuc`. The first and second diagnostic plots seem similar and the third one already slightly worse because the two clusters which we would identify as positive and negative seem less distinctive.

```r
maxent.fit <- trainOcc(x=tr.x, y=tr.y, method='maxent', index=tr.index,
                       tuneGrid=expand.grid(beta = 2^c(-4:5)))
order(-maxent.fit$results$puAuc)[1:3] # the three top ranked models

## [1] 4 3 5
```

```r
# comparable y-axis range
ylim <- c(0,.6)

# threshold, plot, update, predict, threshold, plot, ...
th <- maxentThresholds(maxent.fit)["Maximum.training.sensitivity.plus.specificity.logistic.threshold"]
hist(maxent.fit, maxent.pred, th=0, ylim=ylim); featurespace(maxent.fit, th=th)
maxent.fit <- update(maxent.fit, modRow=3); maxent.pred <- predict(maxent.fit, bananas$x)
th <- maxentThresholds(maxent.fit)["Maximum.training.sensitivity.plus.specificity.logistic.threshold"]
hist(maxent.fit, maxent.pred, th=0, ylim=ylim); featurespace(maxent.fit, th=th)
maxent.fit <- update(maxent.fit, modRow=5); maxent.pred <- predict(maxent.fit, bananas$x)
th <- maxentThresholds(maxent.fit)["Maximum.training.sensitivity.plus.specificity.logistic.threshold"]
hist(maxent.fit, maxent.pred, th=0, ylim=ylim); featurespace(maxent.fit, th=th)
```

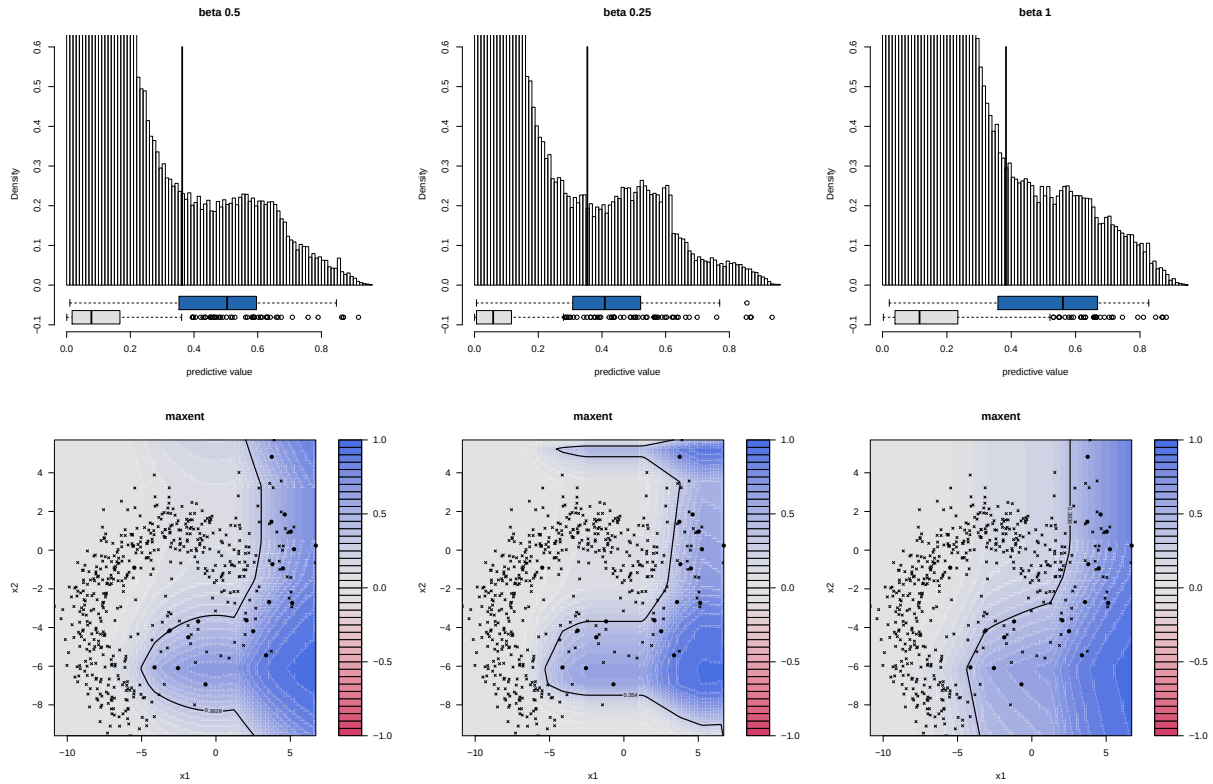

Figure 9: Diagnostic plots (UPPER row) and feature space plots (BOTTOM row) of the three top ranked MAXENT model according to puF (from LEFT to RIGHT in decreasing order).

```
maxent.fit <- update(maxent.fit, modRow=4)
maxent.pred <- predict(maxent.fit, bananas$x)
maxent.th <- maxentThresholds(maxent.fit)["Maximum.training.sensitivity.plus.specificity.logistic.thresh
```

4.5 Comparing OCSVM, BSVM, and MAXENT

Figure 10 shows the dignostic plots and feature space plots of the three final models. The feature space plots of the BSVM and MAXENT clearly show the advantage of the PU-classifiers. Towards the directions of the negative class the decision boundary is fitted tighter while it is more relaxed towards the other directions.

Based on the three diagnostic plots the BSVM model is the one which most likely leads to the highest accuracy. The low density area between the positive and the negative data cluster is the widest and lowest. Furthermore the positive hold-out predictions are for the most part right of the default threshold.

In this example all three methods return plausible results. The OCSVM is certainly a good joice when there is no significant overlap beween the positive and negative class. Particularly when a very large image is to be classified it might be a good starting point because in this case the number of unlabeled samples to be used for training a PU-classifier should be large which results in high training and prediction times. However, if there is a significant class overlap the higher computational cost of a PU-classifier, such as the BSVM of MAXENT should be accepted in favor of a more accurate classification result.

```
hist(ocsvm.fit, ocsvm.pred, th=0, ylim=c(0, 0.5))
featurespace(ocsvm.fit, th=0)

hist(bsvm.fit, bsvm.pred, th=0, ylim=c(0, 0.5))
featurespace(bsvm.fit, th=0)

hist(maxent.fit, maxent.pred, th=maxent.th, ylim=c(0, 0.5))
featurespace(maxent.fit, th=maxent.th)
```

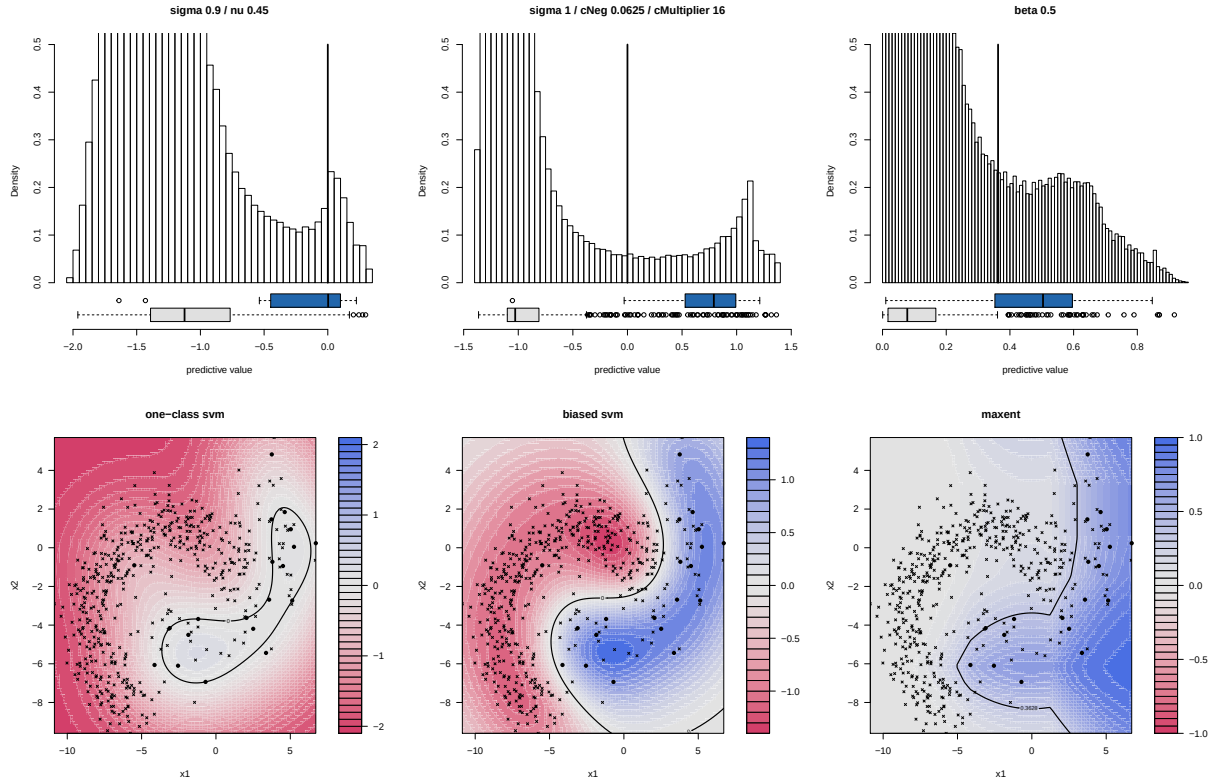

Figure 10: The diagnostic plot (LEFT) and feature space plot (LEFT) for the final model of the OCSVM (TOP), BSVM (MIDDLE), and MAXENT(BOTTOM).

5 Accuracy assessment with PN-data

In many situations PN-data is available for an objective accuracy assessment. Particularly, when optimizing or developing new methods such as one-class classifiers and model selection or threshold selection techniques. Dependent on the objective we might want to compare the discriminative performance of all or a subset of models considered during models selection, or the threshold dependent accuracy for a particular model, or the accuracy for a particular model and threshold. With **evaluateOcc**()[2] an object of class `trainOcc` can be evaluated thoroughly.

[2]The core function of **evaluateOcc**() is **evaluate**() from the package dismo.

We might want to know the accuracy at a particular threshold and compare it with the maximum achievable accuracy, e.g. the accuracy given the threshold maximizing the kappa coefficient. **evaluateOcc**() performs a threshold dependent accuracy assessment for a particular model of a `trainOcc` object. The default model for which the accuracy assessment is performed is the model selected as final model. We can also plot the threshold dependent accuracy, e.g. superimposed over a diagnostic plot as shown in Figure 11 for the BSVM model.

```
bsvm.ev <- evaluateOcc(bsvm.fit, te.u=te.x, te.y=te.y)
th.sel <- 0
th.opt <- slot(bsvm.ev, "t")[which.max(slot(bsvm.ev, "kappa"))]
```

```
hist(bsvm.fit, bsvm.pred, ylim=c(0, 0.5), col="grey", border=NA)
plot(bsvm.ev, add=TRUE, yLimits=c(0, 0.5))
abline(v=c(th.opt, th.sel), lwd=2)
```

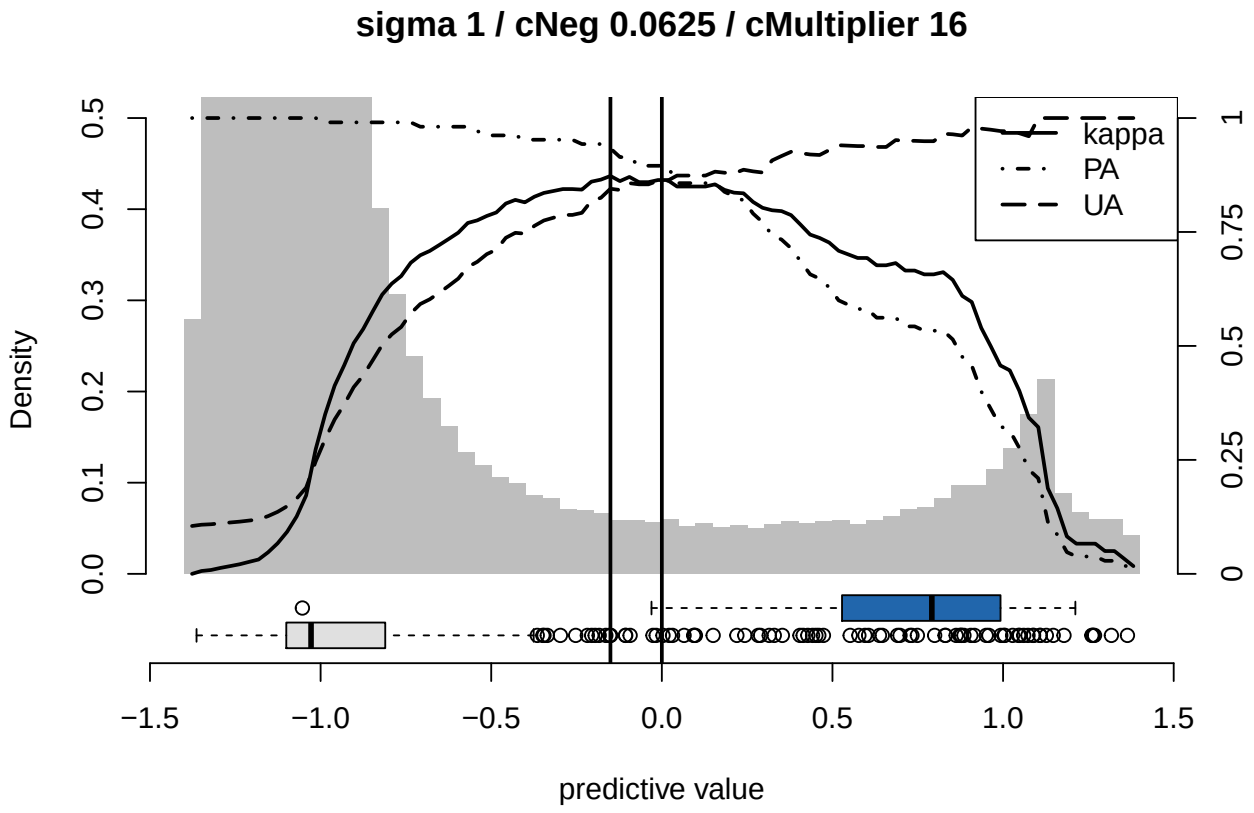

Figure 11: The diagnostic plot with the superimposed threshold dependent accuracy .

```
evaluateAtTh(bsvm.ev, th=th.opt)
evaluateAtTh(bsvm.ev, th=th.sel)
```

```
## Positive/negative (+/-) test samples:
##  105 / 895
##
## Confusion Matrix at threshold -0.1507 :
##
##           + (Test) - (Test)  SUM UA[%]
## + (Pred)        98       18  116    84
## - (Pred)         7      877  884    99
## SUM            105      895 1000
## PA[%]           93       98
## ---
## OA[%]           98
## AUC[*100]       99
## K[*100]         87
```

```
## Positive/negative (+/-) test samples:
##  105 / 895
##
## Confusion Matrix at threshold -0.0113 :
##
##           + (Test) - (Test)  SUM UA[%]
## + (Pred)        94       15  109    86
## - (Pred)        11      880  891    99
## SUM            105      895 1000
## PA[%]           90       98
## ---
## OA[%]           97
## AUC[*100]       99
## K[*100]         86
```

It is also possible to derive the threshold dependent accuracy for all models. This is necessary to compare the discriminative power of differnt one-class classification methods independent of the model selection approach. If we compare the best achievable accuracy of the OCSVM, BSVM, and MAXENT based on the final model we must be aware that we do also evaluate a particular model selection approach, which might not necessarily be desired if comparing the discriminative power of the methods (see Figure **??**). Evaluating all models might also be intersting to evaluate existing or new PU-performance metrics.

Let us first run **evaluateOcc**() on the three `trainOcc` objects with from the previous section. When setting the parameter `allModels=TRUE` an object is returned with the PU-metrics and threshold dependent PN-metrics.

```
ocsvm.ev <- evaluateOcc(ocsvm.fit, te.u=te.x, te.y=te.y, allModels=TRUE)
bsvm.ev <- evaluateOcc(bsvm.fit, te.u=te.x, te.y=te.y, allModels=TRUE)
maxent.ev <- evaluateOcc(maxent.fit, te.u=te.x, te.y=te.y, allModels=TRUE)
```

In order to compare the discriminative power of the three methods and the suitability of the PU-performance metrics we can derive and plot the i) maximum achievable kappas of the best model and of the models selected by ii) manually, iii) maximizing `puF` and iv) maximizing codepuAuc (Figure 12). Figure 12 shows that the highest discriminative power, i.e. the maximum achievable kappa given the best model and threshold, can be achieved by the BSVM, followed by the OCSVM and MAXENT.

In the case of the OCSVM the selection based on `puF` (and thus the final selection) is slightly poorer than the best model but better than the model selected based on `puAuc`. In the case of the BSVM the manually selection model is significantly better than the ones selected based on `puF` or `puAuc`. In the case of the MAXENT all seelected models perform similar to the optimal one.

```
mxK.ocsvm <- sapply(ocsvm.ev$test, function(x) max(x@kappa))
mxK.bsvm <- sapply(bsvm.ev$test, function(x) max(x@kappa))
mxK.maxent <- sapply(maxent.ev$test, function(x) max(x@kappa))

kappas <-
  matrix(c(max(mxK.ocsvm),
           mxK.ocsvm[89],
           mxK.ocsvm[which.max(ocsvm.ev$train$puF)],
           mxK.ocsvm[which.max(ocsvm.ev$train$puAuc)],
           max(mxK.bsvm),
           mxK.bsvm[79],
           mxK.bsvm[which.max(bsvm.ev$train$puF)],
           mxK.bsvm[which.max(bsvm.ev$train$puAuc)],
           max(mxK.maxent),
           mxK.maxent[4],
           mxK.maxent[which.max(maxent.ev$train$puF)],
           mxK.maxent[which.max(maxent.ev$train$puAuc)]),
         4,3)
bp <-barplot(kappas, beside=TRUE, horiz=TRUE, xlab="kappa", col="grey")
text(rep(.01, 9), bp,
     paste( rep(c("ocsvm, ", "bsvm, ", "maxent, "), each=4),
            c("max. kappa", "manual sel.", "max. puF", "max. puAuc")),
     pos=4)
```

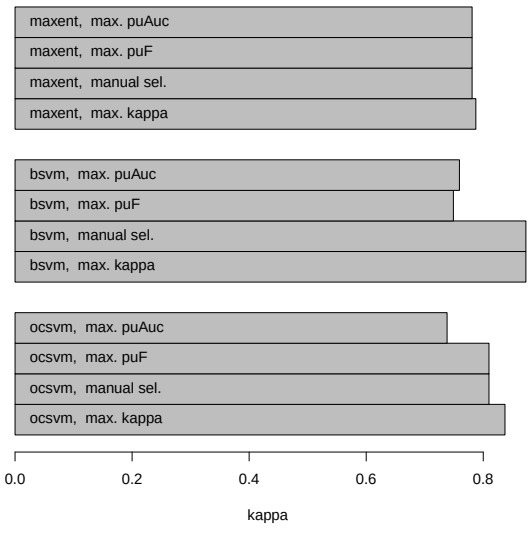

Figure 12: The maximum achievable kappa given the best model and the mdels selected by maximizing puF and puAuc.

6 Summary

The package oneClass serves a double purpose. It should provide an environment for solving real world one-class classification problems in the absence of complete and representative PN-data. Particularly the diagnostic plot can be helpful to examine, approve or improve particular model or threshold selection outcomes.

The package builds upon the powerful infrastructure of the caret packege. The defined one-class classifiers (see **getModelInfoOneClass**()) and PU-performance metrics are custom function defined to be passed to `train`. Thus, the package can be extended easily, e.g. by defining new custom one-class classifiers or PU-performance metrics.

In order to define a custom one-class method a good starting point is to learn about defining custom functions for the function **train**() (see `http://caret.r-forge.r-project.org/custom_models.html`) and use one of the already defined method lists as templates (see **getModelInfoOneClass**()).

Session info

Session info of the system with which this document was compiled:

```
toLatex(sessionInfo())
```

- R version 3.1.0 (2014-04-10), `x86_64-w64-mingw32`
- Locale: `LC_COLLATE=German_Germany.1252`, `LC_CTYPE=German_Germany.1252`, `LC_MONETARY=German_Germany.1252`, `LC_NUMERIC=C`, `LC_TIME=German_Germany.1252`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils
- Other packages: caret 6.0-30, dismo 0.9-3, ggplot2 1.0.0, gridExtra 0.9.1, highr 0.3, kernlab 0.9-19, knitr 1.6, lattice 0.20-29, oneClass 0.1-0, raster 2.2-31, RColorBrewer 1.0-5, rJava 0.9-6, sp 1.0-15
- Loaded via a namespace (and not attached): abind 1.4-0, BradleyTerry2 1.0-5, brglm 0.5-9, car 2.0-20, codetools 0.2-8, colorspace 1.2-4, compiler 3.1.0, digest 0.6.4, doParallel 1.0.8, evaluate 0.5.5, foreach 1.4.2, formatR 0.10, gtable 0.1.2, gtools 3.4.1, iterators 1.0.7, lme4 1.1-6, MASS 7.3-33, Matrix 1.1-4, minqa 1.2.3, mmap 0.6-12, munsell 0.4.2, nlme 3.1-117, nnet 7.3-8, parallel 3.1.0, plyr 1.8.1, pROC 1.7.3, proto 0.3-10, Rcpp 0.11.2, RcppEigen 0.3.2.1.2, reshape2 1.4, rgdal 0.8-16, scales 0.2.4, spatial.tools 1.3.8, splines 3.1.0, stringr 0.6.2, tools 3.1.0

References

[1] Max Kuhn. Contributions from Jed Wing, Steve Weston, Andre Williams, Chris Keefer, Allan Engelhardt, Tony Cooper, Zachary Mayer, and the R Core Team. *caret: Classification and Regression Training*. R package version 6.0-24. 2014. URL: `http://CRAN.R-project.org/package=caret`.

[2] Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. "Estimating the Support of a High-Dimensional Distribution". In: *Neural Computation* 13.7 (2001), pp. 1443–1471. ISSN: 0899-7667. DOI: `10.1162/089976601750264965`.

[3] Bing Liu, Wee Sun Lee, Philip S. Yu, and Xiaoli Li. "Partially Supervised Classification of Text Documents". In: 2002, pp. 387–394.

[4] Bing Liu, Yang Dai, Xiaoli Li, Wee Sun Lee, and Philip S. Yu. "Building text classifiers using positive and unlabeled examples". In: *In: Intl. Conf. on Data Mining*. 2003, pp. 179–188.

[5] Alexandros Karatzoglou, Alex Smola, Kurt Hornik, and Achim Zeileis. "kernlab – An S4 Package for Kernel Methods in R". In: *Journal of Statistical Software* 11.9 (2004), pp. 1–20. URL: `http://www.jstatsoft.org/v11/i09/`.

[6] Jane Elith, Steven J. Phillips, Trevor Hastie, Miroslav Dudík, Yung En Chee, and Colin J. Yates. "A statistical explanation of MaxEnt for ecologists". In: *Diversity and Distributions* 17.1 (2011), pp. 43–57.

[7] Steven J. Phillips and Miroslav Dudík. "Modeling of species distributions with Maxent: new extensions and a comprehensive evaluation". In: *Ecography* 31.2 (2008), pp. 161–175.

[8] Robert J. Hijmans, Steven Phillips, John Leathwick, and Jane Elith. *dismo: Species distribution modeling*. 2013. URL: `http://CRAN.R-project.org/package=dismo`.

[9] Wee Sun Lee and Bing Liu. "Learning with Positive and Unlabeled Examples Using Weighted Logistic Regression". In: *Proceedings of the Twentieth International Conference on Machine Learning (ICML*. 2003, p. 2003.

[10] Jorge M. Lobo, Alberto Jiménez-Valverde, and Raimundo Real. "AUC: a misleading measure of the performance of predictive distribution models". In: *Global Ecology and Biogeography* 17.2 (2008), 145–151. ISSN: 1466-822X. DOI: `10.1111/j.1466-8238.2007.00358.x`.

[11] Canran Liu, Matt White, Graeme Newell, and Richard Pearson. "Selecting thresholds for the prediction of species occurrence with presence-only data". In: *Journal of Biogeography* 40.4 (2013), 778–789. ISSN: 03050270. DOI: `10.1111/jbi.12058`.

[12] Mariya Shcheglovitova and Robert P. Anderson. "Estimating optimal complexity for ecological niche models: A jackknife approach for species with small sample sizes". In: *Ecological Modelling* 269 (2013), 9–17. ISSN: 03043800. DOI: `10.1016/j.ecolmodel.2013.08.011`.