HarvardX: PH125.9x

Data Science: Capstone - CYO

Oral toxicity QSAR modeling

Franz Gruber

21 June, 2020

Introduction

Drug discovery is a lengthy, expensive and risky process[1]. A typical *de novo* drug discovery procedure can take between 10-17 years, with < 10 % chance of finding a successful drug[2]. Early on in the drug discovery process, compounds (i.e. chemical moeities shown to modulate a biological process) need to be triaged out based on their toxicity as fast as possible (*"failing fast and cheap"*[3]). Combining experimentally derived toxicological data of compounds with machine learning algorithms can help to make informed decisions, whether or not to include compounds in the downstream drug discovery pipeline.

In this study, we have developed and tested models, which predict whether a compound is *toxic* (*positive*) or non-toxic (*negative*) using **quantitative structure-activity relationship (QSAR)** modeling. We used a dataset containing 8,992 compounds classified as either toxic or non-toxic based on experiments. For each compound a chemical fingerprint was generated, which encodes structural information in binary form. This leads to 1,024 features for each compound. We trained several machine learning classifiers: *k-NN*, *SVM*, *classification tree*, *random forrest*, *xgboost* and *ensemble* of models. We used `R`[4] as our programming language and the `caret` package[5] for model training and validation. We have reduced the 1,024 bit features (encoding structural information) of our dataset using dimensionality reduction. We have used the reduced features to train a *logistic regression* model, which acted as a baseline to find the model with the best trade-off between *sensitivity*, *specificity* and F_1-score and then use the best two models to predict compound toxicity on a `validation` set, which simulates a dataset our model has not seen before.

Methods

Code availability

This document only contains essential code bits. The full code is available in the accompanying `R script` file.

Dataset

The dataset is provided on the *Machine Learning Repository* website mainted by the University of Irvine[6] as a `zip` file and can be readily downloaded and extracted. The dataset was processed by Ballabio et al.[7] and contains information about whether compounds are very toxic (*positive*) or non-toxic (*negative*). Toxicity classification was based on experimental LD_50 values (i.e. the concentration of a compound to cause 50 % lethality in utilized animals[8]): *positive* ($LD_{50} < 50$ mg/kg), *negative* ($LD_{50} >= 2{,}000$ mg/kg). Furthermore,

[1] https://www.nature.com/articles/nrd1468
[2] https://www.nature.com/articles/nrd2593
[3] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3058157
[4] https://cran.r-project.org/
[5] http://topepo.github.io/caret/
[6] https://archive.ics.uci.edu/ml/index.php
[7] https://onlinelibrary.wiley.com/doi/full/10.1002/minf.201800124
[8] https://onlinelibrary.wiley.com/doi/full/10.1002/minf.201800124

this dataset contains structural information about the compounds used in animal assays to determine LD_{50} values. This information is represented as chemical fingerprints. A chemical fingerprint stores structural information in binary form (i.e. 1 if a structural feature is present, 0 if it is absent). Several algorithms are available to encode structures as fingerprints. A popular version are *extended-connectivity fingerprints (ECFPs)*, which have been used in cheminformatics for similarity searching, compound clustering and virtual screening[9]. The fingerprints this dataset are stored as 1,024 bit fingerprints, which translate to 1,024 features for each compound.

Dataset cleaning, dimensionality reduction and data exploration

The dataset contains 8992 rows *(compounds)* and 1025 columns *(1,024 attributes, 1 datamode/class)* . There are no missing values in our dataset:

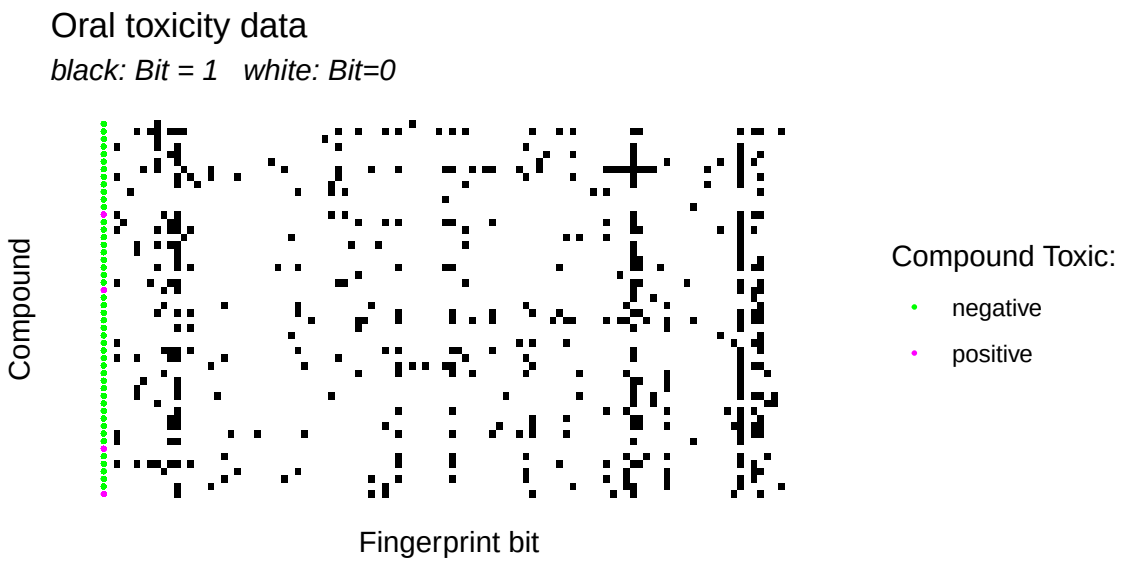
```
input_df %>% is.na() %>% any()
```

```
## [1] FALSE
```

Looking at the first 10 compounds and first 10 bits, we can see how the data looks like:

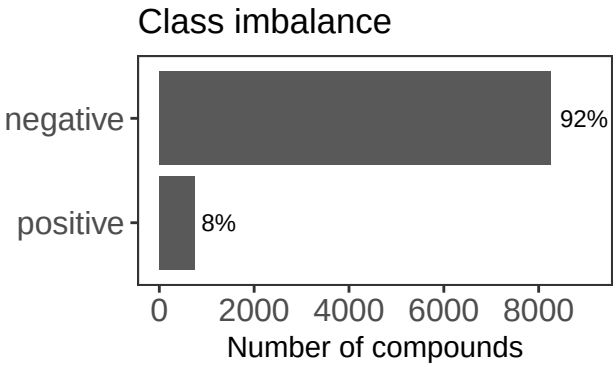
datamode	Bit1	Bit2	Bit3	Bit4	Bit5	Bit6	Bit7	Bit8	Bit9	Bit10
negative	0	0	0	0	0	0	0	0	0	0
negative	0	0	1	0	0	0	0	0	0	0
negative	0	0	0	0	0	0	0	0	0	0
negative	0	0	0	0	0	0	0	1	0	0
negative	0	0	0	0	0	0	0	0	0	0
negative	1	0	0	0	0	0	1	0	0	0
negative	0	0	0	0	0	0	0	0	0	0
positive	0	0	1	0	0	0	0	0	0	0
negative	0	0	0	0	0	0	0	0	0	0
negative	0	0	0	0	0	0	0	0	0	0

We only see one positive compound and we see the occurence or absence of structural information encoded by the binary fingerprint. This can also be visualized:

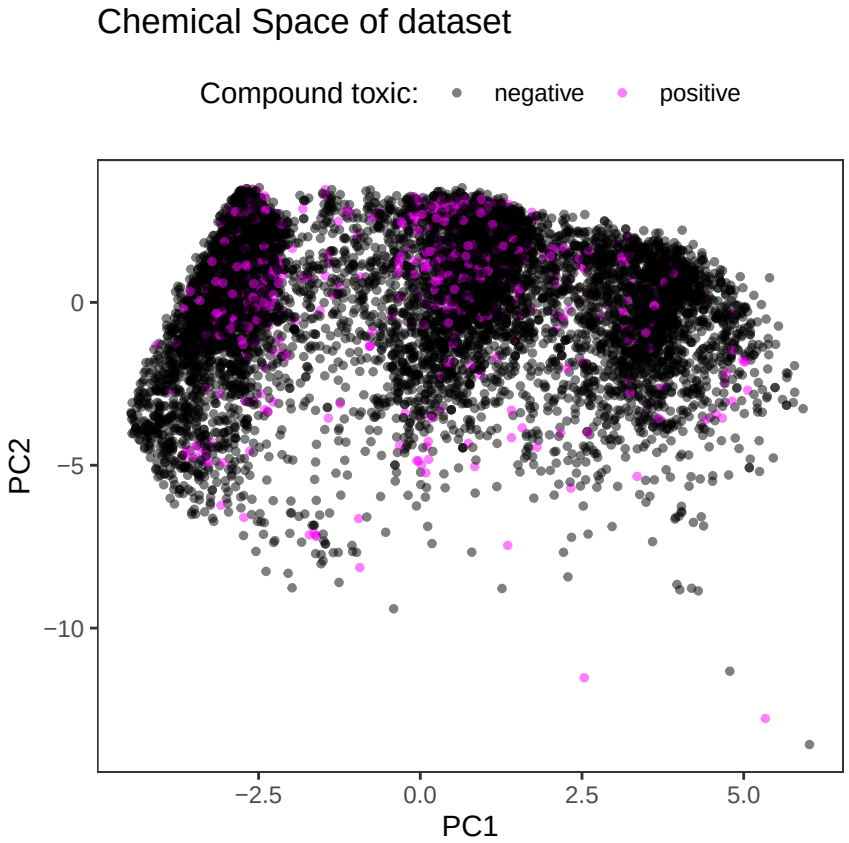


[9]https://pubs.acs.org/doi/pdf/10.1021/ci100050t

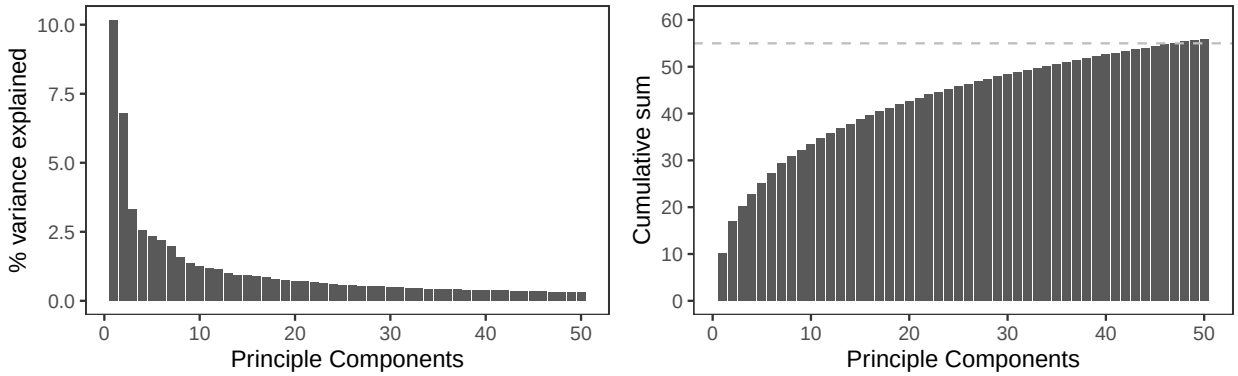
Our dataset is very imbalanced, as only about 8% of compounds have been assigned *positive*.



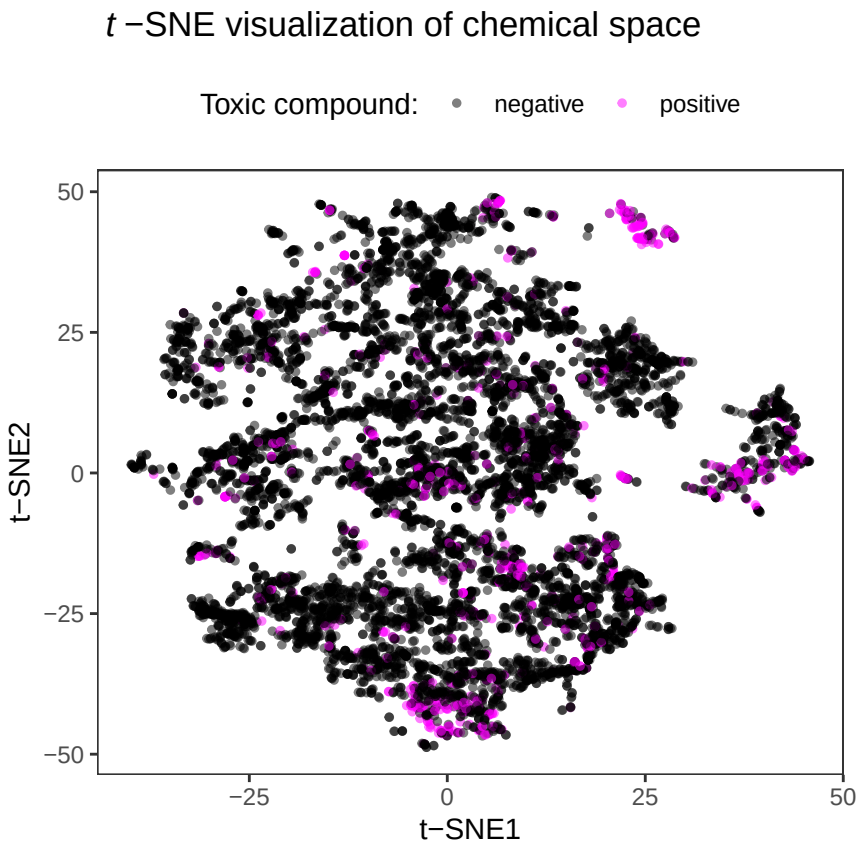
We have seen that our dataest contains 1,024 attributes (or features). It is hard to visualize all of them to infer whether some features are more important than others. However, we can reduce the dimensionality of our dataset using *principle component analysis (PCA)*. This will help us to visualize the chemical space of our compounds by plotting the first two principle components:



We can look at the amount of variance explained by the principle components (left figure panel) and observe that the first 50 principle components add up to about 55% variance explained (dashed line, right figure panel). We will use this information to guide our next approach.



Another dimensionality reduction algorithm called t-stochastic neighbor embedding *(t-SNE)* can improve visualization of high dimensional data, as it might reveal underlying clusters of similar data points[10]. We will use the `Rtsne` function of the `Rtsne` package to perform t-SNE setting the number of dimensions to reduce to `dims` = 3. In addition the `Rtsne` function allows to perform PCA in order to improve computational performance. We will use 50 principle components setting `initial_dims = 50`.



Insights gained

In our data exploration section we have seen that our dataset of 8,992 compounds contains 1,024 attributes (encoded as a chemical fingerprint). The dataset is imbalanced (only few occurances of the *positive* class). We have used dimensionality reduction algorithms (PCA and t-SNE) to visualize chemical space. We saw that both classes *(positive and negative)* overlap in both plots, showing that the dataset represent toxic and

[10] http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf

non-toxic compounds throughout the chemical space within our dataset. The *t*-SNE plot revealed that our dataset contains small clusters of compounds which share similarity.

Modeling approach

For building our QSAR classification model we will use three *t*-SNE dimensions. This will reduce the time it takes to train our models. We will split our dataset into a `model` (80%) and `validation` set (20%) using the `createDataPartition` function of the `caret` package. We will use the validation set only at the very end for model validation.

Furthermore, we will split our `model` dataset into a `train` (80%) and `test` set (20%). We will use those two datasets for model building and optimization.

For model training we will use the default *bootstrap* method of the `caret` package. Furthermore, due to the imbalance of classes, we will use downsampling (i.e. have an equal amount of *positive and negative* classes during model training), which can be specified in the `trainControl(sampling = 'down')` function of the `caret` package.

Our aim is to compare several models, which outperform a logistic regression model, and pick the one with the best performance.

We will briefly explain all modeling algorithms used in the following sections.

Logistic Regression

We will use *Logistic Regression* as a baseline model for classifying compounds. We will use `method = 'glm'` function and `family = 'binomial'` within the `train()` function of `caret` to perform logistic regression.

k-NN model

The **k-Nearest Neighbors** algorithm first calculates distances between observations, using the features of all observations. Typical distance metrics used with the **k-NN** algorithm are *euclidean or cosine* distance. In the second step the alogrithm determines a class label for an observation based on most abundant class in close proximity. This algorithm requires a predefined **k** (i.e. number of neighbors) to be set, which can be determined during model training. In this study, we will use the `knn` function of the `caret` package.

SVM model

A **support vector machine** model tries to find a boundary (i.e. *hyperplane*) between observations in multidimensional space, which can be used for classifcation[11]. We will use the `svmRadial` (radial kernel) function of the `kernlab` package[12].

C5.0 model

The **C5.0** algorithm is an implementation of decision trees[13]. This algortihm uses features to split the data into classes. Descissions on how well a partition based on a feature was can be made on metrices like *Gini Index* or *Entropy*[14]. An advantage of decision trees is that they can be interpreted more readily than other machine learning algorithms and one can visualize them. We will use the `C5.0` function of the `C50` package[15].

random forrest model

The **random forrest** algorithm tries to overcome short comings of a decision tree (changes in training data makes them unstable[16]), by building multiple decision trees (i.e. *forrest*) and take the average of those trees[17]. We will use the `ranger` package[18] to train a **random forrest** model.

[11] Brett Lantz: Machine Learning with R, second edition
[12] https://cran.r-project.org/web/packages/kernlab/index.html
[13] Brett Lantz: Machine Learning with R, second edition
[14] Rafael A. Irizarry: Introduction to Data Science
[15] https://cran.r-project.org/web/packages/C50
[16] Rafael A. Irizarry: Introduction to Data Science
[17] Rafael A. Irizarry: Introduction to Data Science
[18] https://cran.r-project.org/web/packages/ranger

xgboost model

Xgboost stands for e **X** treme **G** radient **B** oosting[19]. It is a very popular machine learning algorithm related to *random forrests* and has won several competitions on *Kaggle*[20]. Xgboost contains additional hyperparameters (e.g. regularization) to prevent overfitting[21].

Model ensemble

Our final model will be an *ensemble* of models. We will use the predictions of all explained models and use *majority vote* to classify our compounds.

Model performance

The aim of this study is to predict, whether a compound is toxic *("positive")* or non-toxic *("negative")*. We will consider a *false negative (FN)* prediction (i.e. compound was classified as non-toxic, but in reality is toxic) to be more harmful then a *false positive (FP)* prediction (i.e. compound was classified as toxic, but in reality is non-toxic). Therefore, we want a model, with a low amount of *FN* classifications.

During our model optimization we will calculate *Sensitivity (true positive rate, TPR or recall)* as:

$$recall = TP/(TP + FN)$$

where TP are true positive predictions and FN are fals negative predictions. The TPR is the proportion of actual positives predicted as positive[22].

We will also calculate *Specificity (PPV or precision)* as:

$$precision = TP/(TP + FP)$$

where TP are true positive predictions and FP are false positive predictions. The PPV is the proportion of actual positives predicted as positives[23].

We will also calculate the F_1-score, defined as:

$$F_1 = 2 \times \frac{precision \cdot recall}{precision + recall}$$

where F_1 is the harmonic average between TPR *(recall)* and PPV *(precision)*.

Furthermore, we will look at model accuracy (also called *success rate*[24]) defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Results

In this section we will look at the performance of several trained models and compare them. We will use all the three t-SNE features received after dimensionality reduction for training our model. Once we have decided on two good performing models, we will use our `validation` set (hold-out data) to see how our model performs on data it has not seen before. We will start our modeling approach with *logistic regression*. To see the effect of an imbalanced dataset on model training, we will train two models: one with the full training set and one where the dataset gets downsampled (i.e. having the same amount of *negative* and *positive* samples) during training. The following table summarizes the performance of the *logistic regression* models:

Model	Accuracy	Sensitivity	Specificity	F1
Logistic Regression all data	0.9179	0.0000	NA	NA
Logistic Regression downsampling	0.5876	0.7458	0.1352	0.2289

[19] https://xgboost.readthedocs.io/en/latest/R-package/xgboostPresentation.html
[20] https://xgboost.readthedocs.io/en/latest/R-package/xgboostPresentation.html
[21] Bradley Boehmke, Brandon Greenwell: Hands-On Machine Learning with R
[22] Rafael A. Irizarry: Introduction to Data Science
[23] Rafael A. Irizarry: Introduction to Data Science
[24] Brett Lantz: Machine Learning with R, second edition

We can see that our *logistic regression* model using all data points achieves an accuracy of 0.9179, which is quite impressive, however its sensitivity is 0. The downsampled has a much smaller accuracy but gains in *Sensitivity* and *Specificity*.

Looking closer at the confusion matrix, we can see what happend with the model that did not use downsampling:

	negative	positive
negative	1320	118
positive	0	0

We have trained a model, which learned to predict mainly *negatives* (non-toxic) but fails to predict *positives* (toxic). This shows that *accuracy* is not a good performance metric if the dataset is imbalanced. Therefore, we will focus on *sensitivity, specificity* and F_1-score. In addition we will use downsampling, which gives a better picture of model performance. One of our main aims is to avoid predicting *false negatives*.

The next model we will test is *k-NN*. We can see that *sensitivity* is slightly higher than the with the *logistic regression* model, and in addition we also increase *specificity* and therfore also the F_1-score.

Model	Accuracy	Sensitivity	Specificity	F1
Logistic Regression all data	0.9179	0.0000	NA	NA
Logistic Regression downsampling	0.5876	0.7458	0.1352	0.2289
k-NN	0.7615	0.7712	0.2236	0.3467

During model training the `caret` package automatically optimized the hyperparameter k (close neighbors):

```
fit_knn$bestTune
```

```
##   k
## 3 9
```

Our next model to try out will be a *support vector machine (SVM)* model. We see that this model, has slighlty lower *sensitivity* but slightly better *specificity*, which leads to an overall higher F_1-score.

Model	Accuracy	Sensitivity	Specificity	F1
Logistic Regression all data	0.9179	0.0000	NA	NA
Logistic Regression downsampling	0.5876	0.7458	0.1352	0.2289
k-NN	0.7615	0.7712	0.2236	0.3467
SVM	0.8401	0.6356	0.2863	0.3947

We can also see that `caret` found the best *SVM* model using the following hyperparameters for *sigma* (inverse kernel width) and C (cost of constraints):

```
fit_svm$bestTune
```

```
##    sigma    C
## 1 0.4332 0.25
```

Next, we will use a *classification tree* model. Like the *SVM* model, the *C5.0 tree* model has slightly lower *sensitivity* but again achieves better *specificity* and therefore F_1-score.

Model	Accuracy	Sensitivity	Specificity	F1
Logistic Regression all data	0.9179	0.0000	NA	NA
Logistic Regression downsampling	0.5876	0.7458	0.1352	0.2289
k-NN	0.7615	0.7712	0.2236	0.3467
SVM	0.8401	0.6356	0.2863	0.3947
C5.0 tree	0.8672	0.5508	0.3202	0.4050

The optimal hyperparameters for this model are chosen by `caret` with *trials* (number of boosting iterations), *model* (tree or rule-based model), *winnow* (feature winnowing/selection):

```
fit_c5$bestTune
```

```
##   trials model winnow
## 5     10  tree   TRUE
```

Can we improve this by using multiple *classification trees* using a *random forrest* model? Our *random forrest* model not only outperformed the *C5.0 tree* but has so far the highest *sensitivity*, although *specificity* is slightly lower than for the *C5.0 tree* model.

Model	Accuracy	Sensitivity	Specificity	F1
Logistic Regression all data	0.9179	0.0000	NA	NA
Logistic Regression downsampling	0.5876	0.7458	0.1352	0.2289
k-NN	0.7615	0.7712	0.2236	0.3467
SVM	0.8401	0.6356	0.2863	0.3947
C5.0 tree	0.8672	0.5508	0.3202	0.4050
random forrest	0.7949	0.8390	0.2640	0.4016

We see that `caret` has optimized `mtry` (number of variables to split), `splitrule` (splitting rule) and `min.node.size` (minimal node size) hyperparameters as follows:

```
fit_rf$bestTune
```

```
##   mtry  splitrule min.node.size
## 2    2 extratrees             1
```

We will now use model type similar to *random forrest* model called `xbgtree`. We see that this model does have similar *sensitivity* and *specificity* performance as the *random forrest* model, however slightly worse.

Model	Accuracy	Sensitivity	Specificity	F1
Logistic Regression all data	0.9179	0.0000	NA	NA
Logistic Regression downsampling	0.5876	0.7458	0.1352	0.2289
k-NN	0.7615	0.7712	0.2236	0.3467
SVM	0.8401	0.6356	0.2863	0.3947
C5.0 tree	0.8672	0.5508	0.3202	0.4050
random forrest	0.7949	0.8390	0.2640	0.4016
xgb Tree	0.7517	0.7966	0.2201	0.3450

The `train` function of the `caret` package has optimized the following hyperparameters: *nrounds* (number of rounds), *max_depth* (depth of tree), *eta* (tree scaling factor), *gamma* (regularization term), *colsample_bytree* (subsample ratio of columns), *min_child_weight* (minimum sum of instance weight) and *subsample* (subsample ratio of training instance).

```
fit_xgbtree$bestTune
```

```
##    nrounds max_depth eta gamma colsample_bytree min_child_weight subsample
## 53     100         3 0.3     0              0.8                1         1
```

Our final model will combine all predictions in an *ensemble*. We will use *majority* vote over all models. We can see that our *ensemble* ranks lower in *sensitivity* but has the best F_1-score of all models.

Model	Accuracy	Sensitivity	Specificity	F1
Logistic Regression all data	0.9179	0.0000	NA	NA
Logistic Regression downsampling	0.5876	0.7458	0.1352	0.2289
k-NN	0.7615	0.7712	0.2236	0.3467
SVM	0.8401	0.6356	0.2863	0.3947
C5.0 tree	0.8672	0.5508	0.3202	0.4050
random forrest	0.7949	0.8390	0.2640	0.4016
xgb Tree	0.7517	0.7966	0.2201	0.3450
ensemble	0.8491	0.7203	0.3160	0.4393

Now that we have tried out several machine learning models, we will validate two of them using our `validation` set (or hold-out set). We will compare the model with the highest *sensitivity (random forrest)* to the model with the highest F_1-score *(ensemble of models)*.

We will train the *random forrest* on the `model` dataset and the validate it with the `validation` set. We see that the *random forrest* model has a similar performance of *sensitivity, specificity* and F_1-score on the `valdiation` set.

Model	Accuracy	Sensitivity	Specificity	F1
Logistic Regression all data	0.9179	0.0000	NA	NA
Logistic Regression downsampling	0.5876	0.7458	0.1352	0.2289
k-NN	0.7615	0.7712	0.2236	0.3467
SVM	0.8401	0.6356	0.2863	0.3947
C5.0 tree	0.8672	0.5508	0.3202	0.4050
random forrest	0.7949	0.8390	0.2640	0.4016
xgb Tree	0.7517	0.7966	0.2201	0.3450
ensemble	0.8491	0.7203	0.3160	0.4393
random forrest validation	0.7920	0.7838	0.2533	0.3828

How will the *ensemble* of models perform? We will train our five models again on the `model` dataset and then predict compound toxicity using the `validation` dataset:

```
#vector with model names
models <- c('knn', 'svmRadial', 'C5.0', 'ranger', 'xgbTree')

#train control for downsampling
train_control <- trainControl(sampling = 'down')

#train models
capture.output(
ensemble <- lapply(models, function(model){

  set.seed(1999, sample.kind = 'Rounding')
  fit_ens <- train(y ~ ., data = model_df,
                   method = model,
                   trControl = train_control)

}), file = 'NUL')
#generate matrix with model predictions
pred_mat_val <- sapply(ensemble, function(x){

  predict(x, validation_df)

})
```

```r
#predict compound toxicity using majority vote
y_hat_ensemble_val <- ifelse(rowMeans(pred_mat_val == 'positive') > 0.5, 'positive', 'negative') %>%
  factor(levels(validation_df$y))
```

We can see that the *ensemble* model also performs similar on the `validation` dataset.

Model	Accuracy	Sensitivity	Specificity	F1
Logistic Regression all data	0.9179	0.0000	NA	NA
Logistic Regression downsampling	0.5876	0.7458	0.1352	0.2289
k-NN	0.7615	0.7712	0.2236	0.3467
SVM	0.8401	0.6356	0.2863	0.3947
C5.0 tree	0.8672	0.5508	0.3202	0.4050
random forrest	0.7949	0.8390	0.2640	0.4016
xgb Tree	0.7517	0.7966	0.2201	0.3450
ensemble	0.8491	0.7203	0.3160	0.4393
random forrest validation	0.7920	0.7838	0.2533	0.3828
ensemble validation	0.8170	0.7027	0.2674	0.3873

Overall, the *random forrest* model has had the best *sensitivity*.

Discussion

Machine learning is indispensable in modern drug discovery. Machine learning models can help making decisions on whether to follow-up on compounds, which have been shown to have an effect on a biological process. One important step early in the drug discovery process is to ensure a compound is specific and does not have any toxic side effects.

QSAR modeling of compound toxicity can help to make follow-up decisions based on predictions using experimental data as modeling inputs. Such models can then be used to predict whether a compound is toxic or not, based on e.g. its chemical structure (encoded as a chemical fingerprints).

We started with our dataset of 8,992 classified compounds by reducing the 1024 features to three t-SNE dimensions. For the modeling process, we have used all three reduced feature columns. We saw that our dataset is highly imbalanced (more *negatives* than *positives*). Therefore, we used *downsampling* during the modeling process. We have tested five models *(k-NN, SVM, classification tree, random forrest, xgboost)* and performed predictions in an *ensemble* of models using majority vote. We saw that the *random forrest* model performed best (highest *sensitivity* and high F_1-score). We consider *sensitivity* more important, as we want to avoid *false negative* predictions (compound is predicted non-toxic, but in reality is toxic) at the expense of decreasing *specificity* (compound is predicted toxic, but is not-toxic). Our *random forrest* model had similar performance on the `validation` (hold-out) set. The model could be improved iteratively by testing predicted compounds in toxicological assays (experiments) and feed this information back to the model to improve it.

The dataset had some limitations. It did not contain any compound identifier, so all information we had were the 1024 fingerprint bits encoding structural information (it is not possible to decode a fingerprint into its original chemical structure). Would the dataset contain a compound identifier such as ChEMBL or Pubchem ID, or structural annotation such as SMILES or InChIKey one could perform web scraping to blend in additional compound data, calculate further properties of a compound (e.g. physico-chemical properties), or generate fingerprints with more information (e.g. 2048 bit). Any additional compound properties could be used as additional input features for the modeling process.

What would be the next steps? We could try reducing our dataset to more dimension and compare how a *random forrest* model would perform. In addtion other dimensionality reduction techniques such as UMAP could be tried out. During the modeling process we only used *downsampling* to ensure equal classes. There are other algorithms not explored here e.g. *upsampling* or *SMOTE*, which could be explored in more detail. Furthermore, we could perform a more in-depth hyperparameter tuning, when training a model. Our *random forrest* model was optimized to use only 2 feature columns (`mtry = 2`). It would be interesting to

see how we could improve the model using a broader set of hyperparameters in a tuning grid. There is much room to improve our models. E.g. one could introduce cost-sensitive learning, which would penalize certain outcomes more than others. The `C50` package offers options to do so, which have not been explored here. To our surprise the `xgboost` model, which is a very popular model did not perform as good as the *random forrest* model. Again, a more in-depth hyperparameter tuning might lead to better performance. Finally, we could enrich our *ensemble* of models by adding predictions of additional models.