OpenRiskNet

RISK ASSESSMENT E-INFRASTRUCTURE

Case Study

Modelling for Prediction or Read Across [**ModelRX**]

SUMMARY

The ModelRX case study was designed to cover the important area of generating and applying predictive models, and more specifically QSAR models in hazard assessment endorsed by different regulations, as completely *in silico* alternatives to animal testing and useful also in early research when no data is available for a compound. The QSAR development process schematically presented in Figure 1 begins by obtaining a training data set from an OpenRiskNet data source. A model can then be trained with OpenRiskNet modelling tools and the resulting models are packaged into a container, documented and ontologically annotated. To assure the quality of the models, they are validated using OECD guidelines (Jennings et al. 2018). Prediction for new compounds can be obtained using a specific model or a consensus of predictions of all models. This case study will present this workflow with the example of blood-brain-barrier (BBB) penetration, for which multiple models were generated using tools from OpenRiskNet consortium and associated partners used individually as well as in a consensus approach using Dempster-Shafer theory (Park et al. 2014; Rathman et al. 2018).

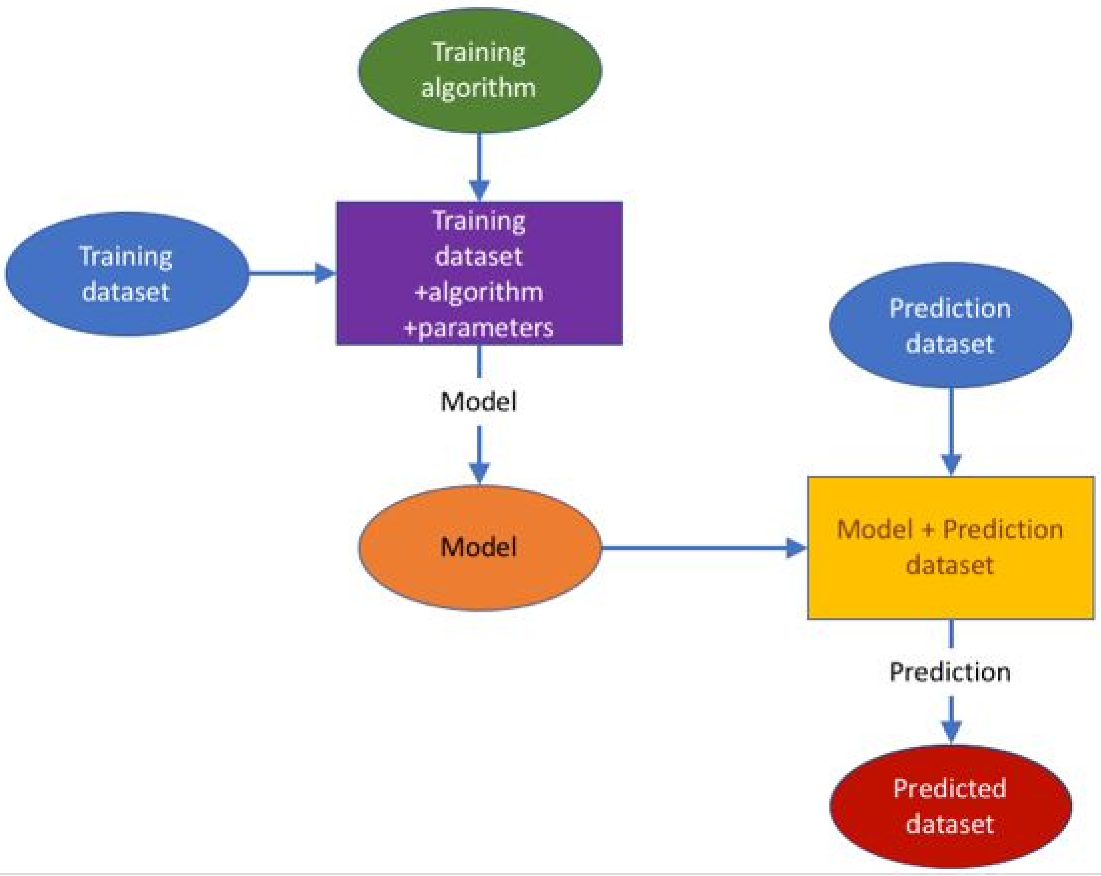


Figure 1. Building and using a prediction model workflow

DESCRIPTION

Implementation team

CS leader	Team
Harry Sarimveis (NTUA)	NTUA, JGU, UU, EwC

Case Study objective

The objectives of this case study are: use *in-silico* predictive modelling approaches (QSAR) to support final risk assessment by supporting similarity identification related to the DataCure case study (by providing tools for calculating theoretical descriptors of substances) and fill data gaps for specific compounds or to augment incomplete datasets.

Risk assessment framework

The ModelRX case study contributes in two tiers (as tiers are defined in Berggren et al. 2017):

- On the one hand, it provides computational methods to support suitability assessment of existing data and identification of analogues (Tier 0);
- On the other hand, it provides predictive modelling functionalities, which are essential in the field of final risk assessment (Tier 2).

DEVELOPMENT

Databases and tools

Jaqpot Quattro (NTUA), CPSign (UU), JGU WEKA Rest service (JGU), Lazar (JGU/IST).

Technical implementation

From the developer's perspective, this case study demonstrates the improved interoperability and compatibility/complementarity of the models based on services deployed following the general steps that have been agreed for developing the OpenRiskNet infrastructure and services. Each application is delivered in the form of a container image and deployed. Docker is used as the basic engine for the containerisation of the applications. Above that, OpenShift, which is a container orchestration tool, is used for the management of the containers and services. OpenShift provides many different options for deploying applications. Some recipes and examples have been documented in the OpenRiskNet GitHub page[1].

When an application is deployed, a service discovery mechanism is responsible for discovering the most suitable services for each application. Based upon the OpenAPI specification, each API should be deployed with the swagger definition. This swagger file should then be integrated with the Json-LD annotations as dictated by the Json-LD specification. The discovery service mechanism parses the resulting Json-LD and resolves the annotations into RDF triplets. These triplets can then be queried with SPARQL. The result of the SPARQL query lets the user know which services are responsible for making models or predictions. The documentation can be found via swagger definition of each application. This way, the services are integrated into the OpenRiskNet virtual environments and can be used and incorporated into end user applications and other services as demonstrated here with a workflow performing consensus modelling.

QSAR modelling was already the main topic of the OpenTox project, which is a clear predecessor of OpenRiskNet. OpenTox had a much more focused aim and the clear goal to have very interlinked services, where it is even possible to combine parts of the workflow from different partners, e.g. descriptor calculation is performed by a service from one partner, the model is trained using algorithms from another partner, and finally the prediction is performed by integrating the trained model into a user interface of a third partner. To allow this, the technical implementation had to be based on rigorously defined modelling APIs the OpenTox standards[2]. Additionally, following these specifications and standards opened the full flexibility of model development and performing predictions to the user, which, on the one side, allows optimization of the workflow to a specific problem at hand but, on the other hand, also requires more experienced users. Therefore, the case study and the integration of QSAR services into OpenRiskNet in general used the OpenTox specifications as the starting point for developing a more flexible QSAR workflow, which, like the OpenTox workflow, has all components represented in Figure 1 but allows services to combine and simplify different steps and, in this way, e.g. reduces the necessity for rigorous usage of Uniform Resource

[1] https://github.com/OpenRiskNet/home/tree/master/openshift
[2] http://old.opentox.org/dev/apis/api-1.2

Identifiers (URIs) for simple substances, which can be referenced by chemical identifiers like SMILES or InChIs, and descriptors, e.g. when they are calculated on the fly by the service. This allows the easier integration of external tools as e.g. provided by the associated partners, which don't support all the features required for an OpenTox service. This approach allows us to include both more straightforward tools that can work together with OpenRiskNet with minimal setup, but at the same time can accommodate more feature-rich tools that require more feature-rich APIs to provide their full offering.

However, OpenRiskNet enforces additional requirements due to the broadening of the application area. As described in detail in deliverable reports D2.2 and D2.4, standardization of the APIs was not possible and even not desired to allow very different tools from many areas to be run on the OpenRiskNet virtual environments. Instead, harmonization and interoperability was obtained by semantic annotation of the APIs and the semantic interoperability later, which provide the user with the information needed to link to services using workflow tools. Modelling APIs need a high level of integration into the OpenRiskNet ecosystem. Integration with the DataCure CS is vital. On the semantic interoperability layer, training datasets should be compatible with an algorithm and, in turn, prediction datasets should be compatible with a prediction model. Additionally, in a best practice scenario, the generated models and datasets need to be accompanied with semantic metadata on their life cycle, thus enforcing semantic enrichment of the dynamically-created entities. Algorithms, models and predicted datasets are built as services, discoverable by the OpenRiskNet discovery service. This is a step that should occur whenever an entity (algorithm, model, predicted dataset) is created.

To enable the user to train models and use them in predictions, , the guidelines agreed on by OpenRiskNet for functionality, which should be provided by QSAR and read-across services to allow highest flexibility include the following steps. As already stated before, some of the requirements might become irrelevant for a specific service if it combines different steps to provide an easier way to make predictions, especially for less experienced users. More details on the features implemented in each service are annexed below.

1. Selecting a training data set

The user chooses among OpenRiskNet compliant data sets already accessible through the discovery service. Following the OpenTox specification, a **Dataset** includes, at a minimum:

- a dataset URI
- substances: substance URIs (each substance URI will be associated with a term from the ontology)
- features:
    - feature URIs (each feature URI will be associated with a term from the ontology)
    - values in numerical format
    - category (experimental/computed)
    - if computed, the URI of the model used to generate the values
    - Units

An alternative path to provide data in OpenRiskNet e.g. followed in the DataCure case study is by dedicated services offering the following elements:

- a dataset URI
- a semantically annotated data API providing information on how to access the data

and what specific data schema is used

With this information, it will be possible for alternative implementations to integrate into OpenRiskNet and interact with modelling services following the full OpenTox specifications, provided they also implement intermediate processing steps, i.e. within the environment of a Jupyter notebook, to structure the data so that it fulfills the minimum set of requirements for the following steps of the QSAR workflow.

2. Selecting a (suitable) modelling algorithm

The user chooses a suitable algorithm. **Algorithms** include at a minimum:

- algorithm URI
- title
- description
- algorithm type (regression/classification)
- default values for its parameters (where applicable)

3. Specifying parameters and Generating Predictive model

Once an algorithm has been selected, the user defines the endpoint, selects the tuning parameters, (only if different values from the default ones are desired) and runs the algorithm. The generated **Model** contains, at a minimum:

- model URI
- title
- description
- the URI of the dataset that was used to create it
- the URIs of the input features
- the URI of the predicted feature
- values of tuning parameters

The following were identified as possible extensions:

- *Include services/APIs for validation of the generated model*

  This has been implemented by Jaqpot (NTUA) as well as Weka (JGU) and Lazar (JGU).

- *Provide mechanisms to pick out the best algorithm for a specific dataset: (e.g. RRegrs)*

  As this is a highly resource-intensive process and requires significant exploration of possible choices and parameters, it would translate into additional workload on the infrastructure, which would be challenging to sustain. That, together with the fact that users have a selection of tools in the search for the best model: tools like RRegrs[3] (written in the R language) and TPOT (written in Python) can be used in common R-Python notebooks so that the user settles down on the most appropriate model and then resorts to OpenRisket tools.

[3] https://github.com/muntisa/RRegrs

- *Include algorithms to calculate domain of applicability*

  Domain of applicability calculations have been implemented by Jaqpot (NTUA). An alternative to applicability domains using Conformal Prediction methodologies (Norinder et al, 2014) is provided by the CPSign/ModelingWeb tool (UU).

4. Selecting a prediction data set

After the creation of a model, the user selects a prediction dataset meeting all the requirements specified in (Chomenidis et al. 2017). This dataset is tested for compatibility against the required features of the model in terms of feature URIs, i.e. the dataset should contain all the subset of features used to produce the model. Additional features are allowed, however they will be ignored.

5. Running predictive model on the prediction data set

The predictive model is applied on the prediction dataset to generate the predicted dataset, which must be compatible with the requirements specified in (Chomenidis et al. 2017). The predicted dataset augments the prediction dataset with all necessary information about the predicted feature:

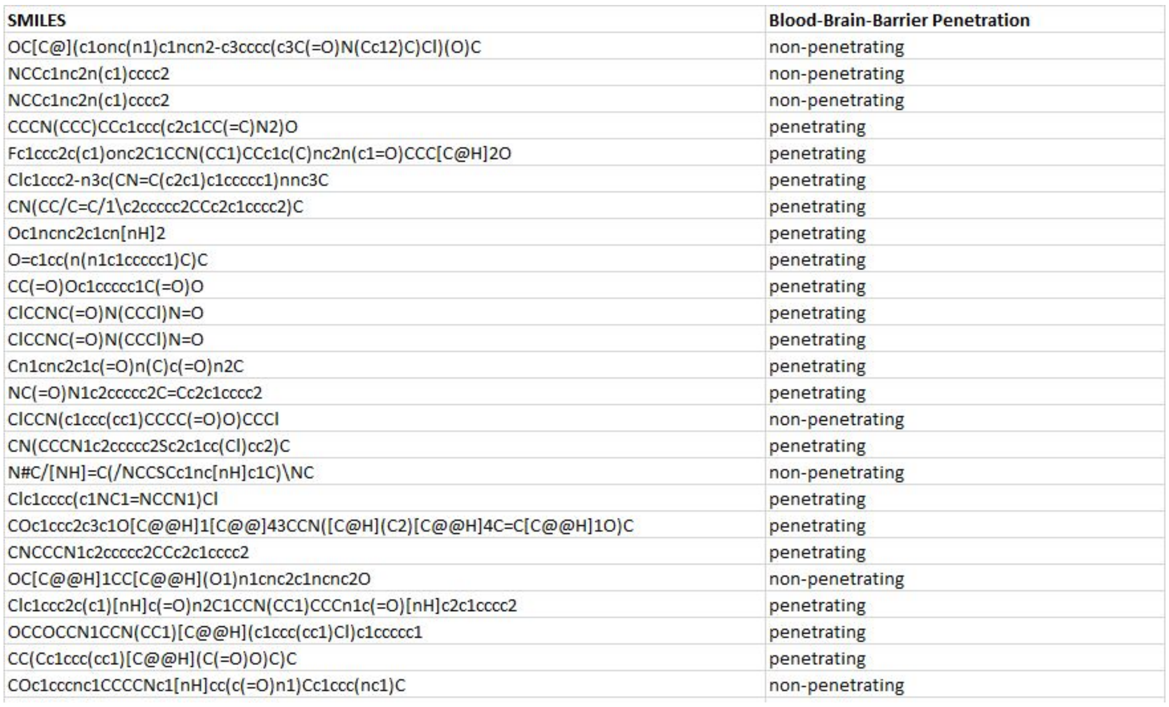
- prediction feature URIs (each feature URI will be associated with a term from the ontology)
- values in numerical format
- category (computed)
- the URI of the model used to generate the values
- units

OUTCOMES

The work on the case study was designed to showcase how the workflow defined above for producing semantically annotated predictive models can be shared, tested, validated and eventually be applied for predicting adverse effects of substances in a safe by design and/or risk assessment regulatory framework. OpenRiskNet provides the necessary functionalities that allow not only service developers but also researchers and practitioners to easily produce and expose their models as ready-to-use web applications. The OpenRiskNet e-infrastructure serves as a central model repository in the area of predictive toxicology. For example, when a research group publishes a predictive model in a scientific journal, they can additionally provide the implementation of the model as a web service using the OpenRiskNet implementation. The produced models contain all the necessary metadata and ontological information to make them easily searchable by the users and systematically and rigorously define their domain of applicability. Most importantly, the produced resources are not just static representations of the models, but actual web applications where the users can supply the necessary information for query substances and receive the predictions for their adverse effects. However, because of the harmonization and interoperability, these are not just stand-alone tools but can be easily combined to improve the overall performance or can be used to replace older tools with newer ones without changing the overall procedure. This was demonstrated with the workflows for developing a consensus model for blood-brain-barrier (BBB) penetration (available online[4]). The test set of 414 compounds was obtained from the Lazar service[5]. Part of this dataset is shown below:

[4] https://github.com/OpenRiskNet/notebooks/tree/master/ModelRX/Blood-brain%20barrier%20-%20Consensus

[5] https://lazar.prod.openrisknet.org/predict/dataset/blood-brain-barrier



SMILES	Blood-Brain-Barrier Penetration
OC[C@](c1onc(n1)c1ncn2-c3cccc(c3C(=O)N(Cc12)C)Cl)(O)C	non-penetrating
NCCc1nc2n(c1)cccc2	non-penetrating
NCCc1nc2n(c1)cccc2	non-penetrating
CCCN(CCC)CCc1ccc(c2c1CC(=C)N2)O	penetrating
Fc1ccc2c(c1)onc2C1CCN(CC1)CCc1c(C)nc2n(c1=O)CCC[C@H]2O	penetrating
Clc1ccc2-n3c(CN=C(c2c1)c1ccccc1)nnc3C	penetrating
CN(CC/C=C/1\c2ccccc2CCc2c1cccc2)C	penetrating
Oc1ncnc2c1cn[nH]2	penetrating
O=c1cc(n(n1c1ccccc1)C)C	penetrating
CC(=O)Oc1ccccc1C(=O)O	penetrating
ClCCNC(=O)N(CCCl)N=O	penetrating
ClCCNC(=O)N(CCCl)N=O	penetrating
Cn1cnc2c1c(=O)n(C)c(=O)n2C	penetrating
NC(=O)N1c2ccccc2C=Cc2c1cccc2	penetrating
ClCCN(c1ccc(cc1)CCCC(=O)O)CCCl	non-penetrating
CN(CCCN1c2ccccc2Sc2c1cc(Cl)cc2)C	penetrating
N#C/[NH]=C(/NCCSCc1nc[nH]c1C)\NC	non-penetrating
Clc1cccc(c1NC1=NCCN1)Cl	penetrating
COc1ccc2c3c1O[C@@H]1[C@@]43CCN([C@H](C2)[C@@H]4C=C[C@@H]1O)C	penetrating
CNCCCN1c2ccccc2CCc2c1cccc2	penetrating
OC[C@@H]1CC[C@@H](O1)n1cnc2c1ncnc2O	non-penetrating
Clc1ccc2c(c1)[nH]c(=O)n2C1CCN(CC1)CCCn1c(=O)[nH]c2c1cccc2	penetrating
OCCOCCN1CCN(CC1)[C@@H](c1ccc(cc1)Cl)c1ccccc1	penetrating
CC(Cc1ccc(cc1)[C@@H](C(=O)O)C)C	penetrating
COc1cccnc1CCCCNc1[nH]cc(c(=O)n1)Cc1ccc(nc1)C	non-penetrating

In the consensus modelling, we combine independent sources of evidence in a well defined manner to generate the final prediction and estimate its uncertainty. For that purpose we employed the Dempster-Shafer theory (DST), which provides a solid mathematical framework for combining multiple evidences, where each of them is characterized by evidence-specific certainty (Park et al. 2014, Rathman et al. 2018).

Here we combine multiple independent *in silico* predictive models, each of which classifies compounds as BBB penetrating or non-penetrating. The models, which were available before the start of the project or were specifically generated for this case study, are:

LAZAR (JGU/IST): The model is based on the MP2D fingerprints and uses nearest-neighbour algorithm with a weighted majority vote and distance based on the Tanimoto similarity with a threshold of 0.1. The model was validated using 3-fold cross-validation.

Jaqpot (NTUA): The model is based on the Mordred descriptors obtained from the SMILES using RDKit and recursive feature elimination for the selection of the 20 most important features. The model uses logistic regression based on these 20 features. The model was validated using 10-fold cross-validation.

CPSign (UU): The Cross Venn-ABERS predictor (CVAP), as implemented in the CPSign software. The signatures descriptor of atom neighbours height 1 to 3 was used together with an SVM with RBF kernel as underlying learning model. The gamma and cost values were optimized with 10-fold cross-validation and set to 0.0039 (gamma) and 4 (cost).

JGU WEKA Rest service (JGU): The model is based on features/fingerprints extracted from the Blood-Brain Barrier dataset using a graph mining based algorithm called LAST-PM or Latent Structure Pattern Mining. The fingerprints used in the majority of chemical toolkits are handcrafted by chemical experts, however, identifying frequent or correlated subgraphs has the potential to reveal latent information not present in any individual ground features. This also provides a potentially different perspective to the problem at hand. The model is created using Support Vector Machines (specifically the implementation provided by LibSVM in Weka). The model parameters (cost and gamma) have been tuned using grid search.

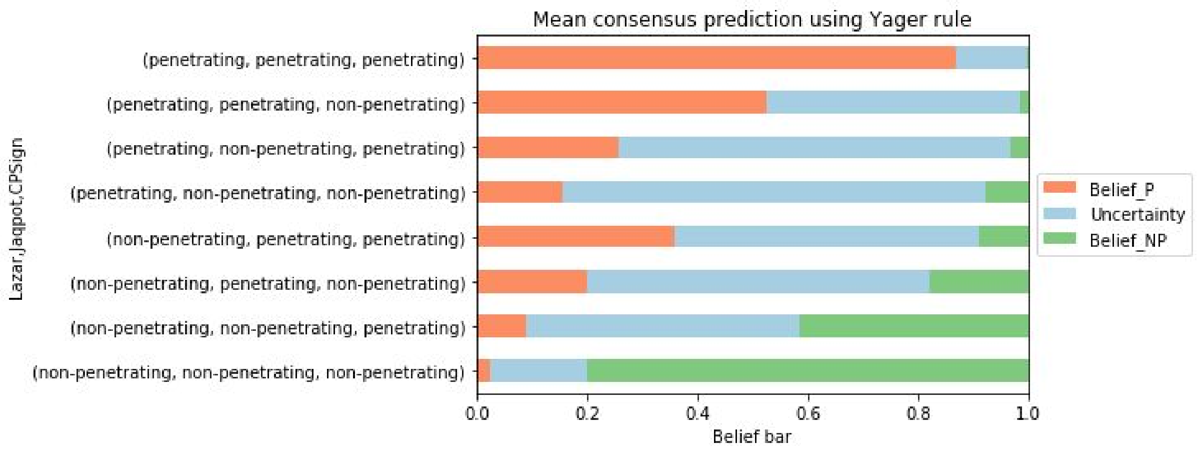
OCHEM (BIGCHEM): On-line Chemical Database and Modeling environment (OCHEM) platform was used to develop OCHEM model, which was based on Associative Neural Network (ASNN, Tetko 2008) and alvaDesc (https://www.alvascience.com/alvadesc/) descriptors. The default hyper-parameters of the ASNN were used. Namely, neural networks with one hidden layer, which included three neurons each, were used. Each network was trained for 1000 iterations using early-stopping. ASNN used an ensemble of 64 individual networks, predictions of which were averaged, to provide final model predictions. The 2D to 3D conversion was done using Corina (https://www.mn-am.com/) program. The developed model is available at http://ochem.eu/model/12147752. The accuracy of predictions are estimated based on the uncertainty of ensemble predictions as described in (Sushko et al 2010).

In DST, each model used for the consensus is not weighted equally but based on the confidence in the individual predictions. The certainty of a positive or negative prediction of every model is characterized by its positive or negative predictive value (PPV or NPV), where PPV is defined as the fraction of true positives of all the positive predictions (PPV = TP / (TP + FP)), and NPV is defined as the fraction of true negatives of all the negative predictions by a given model (NPV = TN / (TN + FN)). In our case PPV and NPV were obtained from the k-fold cross-validation of the respective models. Since the models from the associated partners were not completely integrated into the OpenRiskNet infrastructure at the time of this writing, only the LAZAR, Jaqpot, CPSign, WEKA and OCHEM models are considered in the following analysis.

	LAZAR	**Jaqpot**	**CPSign**	**WEKA**	**OCHEM**
PPV	0.765	0.864	0.806	0.909	0.86
NPV	0.690	0.788	0.698	0.924	0.71

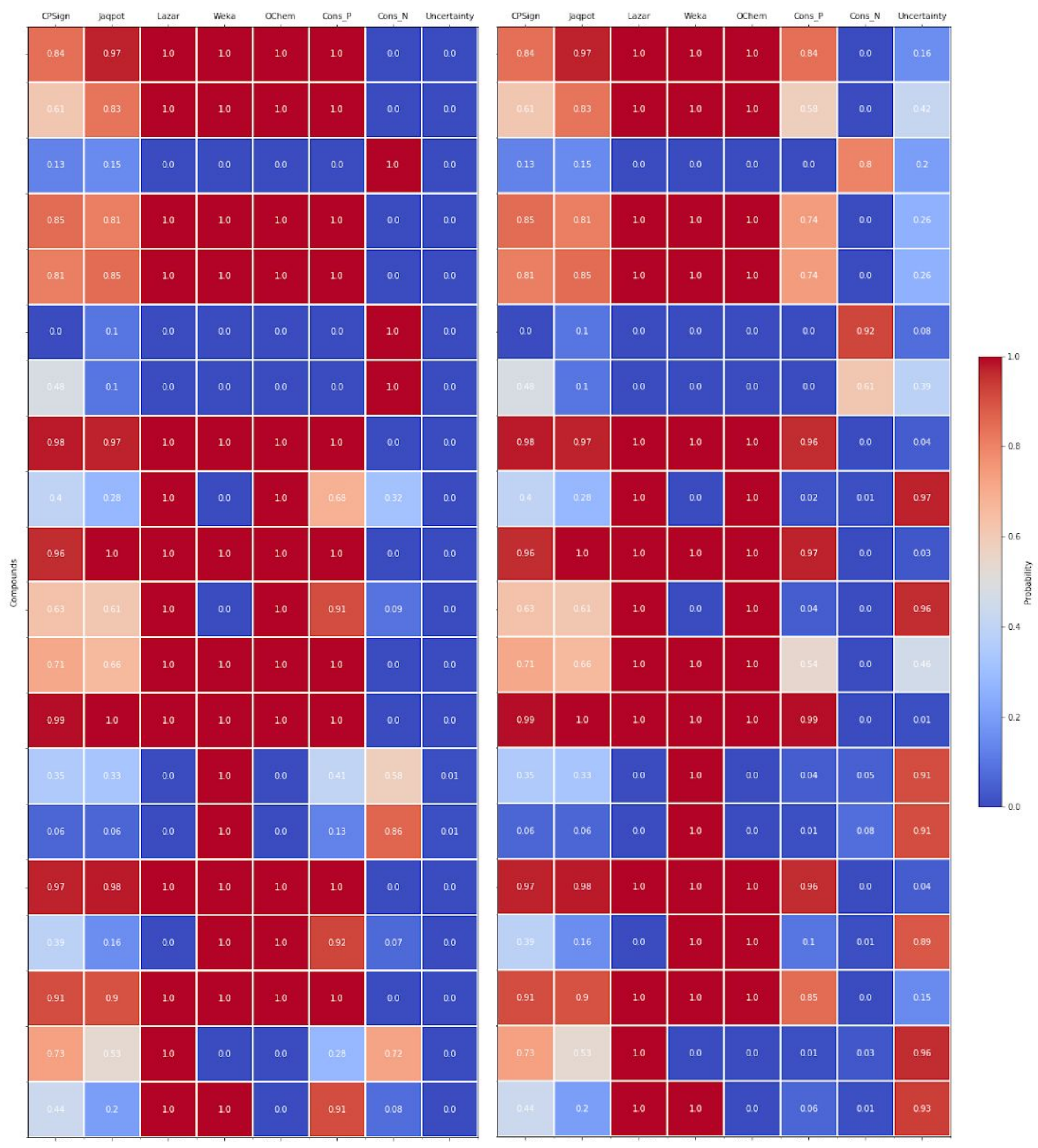
DST additionally allows the user to put more or less weight on the concordance between the sources of evidence by the choice of the so-called combination rule. In this way the user is allowed to choose how conservative the consensus prediction should be. The two rules examined in this case study are Dempster and Yager combination rule. The first one neglects the disagreements among the various sources of evidence and provides lower uncertainties, whereas the conflicts among the sources of evidence result to a greater uncertainty when using the latter one. In other words, Dempster combination rule provides results similar to the majority voting rule, whereas Yager rule is more likely to produce equivocal prediction in case of disagreements between the sources of evidence.

For every possible outcome, the DST provides **belief** and **plausibility**, which can be viewed as the lower and upper bound of the probability of that outcome, respectively, and their difference is the **uncertainty** of that outcome.
To illustrate these concepts, the figure below shows the computed average belief, plausibility and uncertainty for different combinations of predictions from three sources of evidence (where each source of evidence is a single predictive model, which predicts a compound to be penetrating or non-penetrating). The three models used for this demonstration are Lazar (JGU/IST), Jaqpot (NTUA) and CPSign (UU). The red bars represent the belief that a compound is penetrating, green bars show that it is non-penetrating and the blue bar is the corresponding uncertainty.
Belief of penetrating and uncertainty sum up to the plausibility that compound is penetrating. Analogously, the belief of non-penetrating and uncertainty sum up to plausibility that a compound is non-penetrating. Clearly, both beliefs and uncertainty should always sum up to 1.



The heatmaps below show the difference of the consensus predictions using the Dempster rule (left) and the Yager rule (right). The color designates the probability and ranges from blue (0) through white (0.5) to red (1). Every row represents the result for a single compound. The first 4 columns in each figure correspond to the probability that a given compound is BBB penetrating according to the individual predictive models. The next two columns (termed *Cons_P* and *Cons_N*) represent the consensus belief that a given compound is penetrating (*Cons_P*) or non-penetrating (*Cons_N*), while the last column depicts the uncertainty of the consensus prediction. Note that *Cons_P*, *Cons_N* and *Uncertainty* always sum up to 1.

When all 4 models agree in their prediction (all 4 are blue or red), the consensus prediction is very clear. However, the difference between the two combination rules becomes evident when we look at the cases where the models disagree. Clearly, Dempster rule tends to diminish uncertainty and puts more weight to the prevailing prediction, while the Yager rule tends to increase uncertainty, being a more conservative prediction since more molecules will show high uncertainty levels preventing a clear categorization.



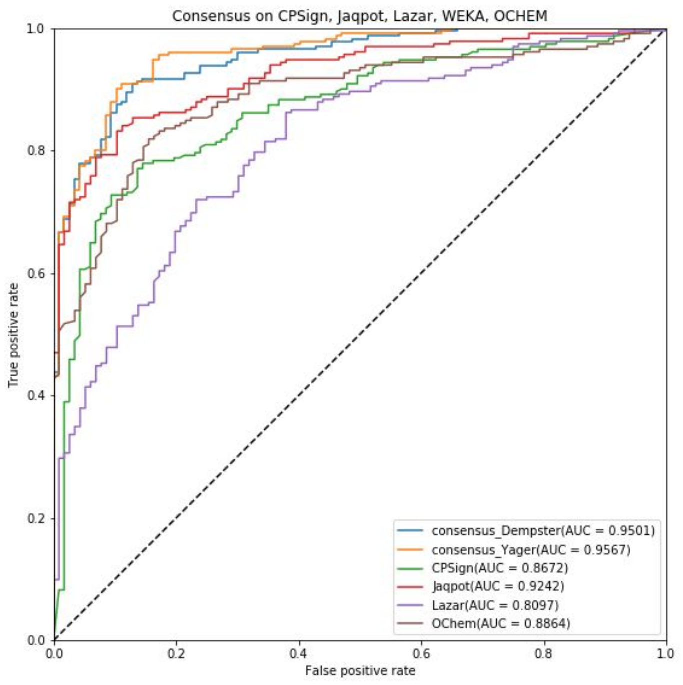
Heatmap representation of BBB predictions of individual and consensus models. Left side: consensus model using the Dempster combination rule. Right side: consensus model using the Yager combination rule. Each row presents the results for a single compound. Only the results for the first 20 compounds are displayed. Columns 1 to 5 represent the probability that a given compound is BBB penetrating as predicted by individual models (from left to right: CPSign, Jaqpot, Lazar, WEKA, OCHEM). Columns 6 and 7 represent the consensus probability that a given compound is BBB penetrating and non-penetrating, while column 8 depicts the uncertainty of the consensus approach. The figure was generated within the workflow of the *batch-compounds-offline.ipynb* Jupyter notebook available over GitHub[6].

[6] https://github.com/OpenRiskNet/notebooks/blob/master/ModelRX/Blood-brain%20barrier%20-%20Consensus/batch-compounds-offline.ipynb

The performance of individual predictive models and the consensus predictions can be compared on the ROC plot (below). Clearly, the consensus predictions based on five models offer better performance, which is also reflected in greater AUC values. Note that for the WEKA predictive model the results could not be quantified, so the ROC curve could not be made.

Also it should be noted that not every predictive model was able to make a prediction for all the compounds of the test set. For example:
- The Lazar predictive model uses a nearest-neighbour algorithm with a cutoff chemical similarity of 0.1. If there was no compound found in the training set that would be sufficiently similar, then prediction could not be made.
- The Jaqpot predictive model uses Mordred descriptors, which are generated from the Mol representation of the compound. During preprocessing the Mol representations were generated from the SMILES strings using RDKit, but not all conversions were successful. Hence in the analysis for Jaqpot compounds for which Mol representation and Mordred descriptors could not be calculated were removed.

These restrictions resulted in the final number of 348 compounds, for which the consensus predictions could be made.



This figure was generated within the workflow of the *model-comparison.ipynb* Jupyter notebook available over GitHub[7]. More complete summary and comparison of consensus predictions of different combinations of models is available in the *model-comparison-summary.ipynb* also available over GitHub[8].

7 https://github.com/OpenRiskNet/notebooks/blob/master/ModelRX/Blood-brain%20barrier%20-%20Consensus/model-comparison.ipynb.

8 https://github.com/OpenRiskNet/notebooks/blob/master/ModelRX/Blood-brain%20barrier%20-%20Consensus/model-comparison-summary.ipynb

REFERENCES

Berggren, Elisabet, Andrew White, Gladys Ouedraogo, Alicia Paini, Andrea-Nicole Richarz, Frederic Y. Bois, Thomas Exner, et al. 2017. “Ab Initio Chemical Safety Assessment: A Workflow Based on Exposure Considerations and Non-Animal Methods.” *Computational Toxicology (Amsterdam, Netherlands)* 4 (November): 31–44.

Chomenidis, Charalampos, Georgios Drakakis, Georgia Tsiliki, Evangelia Anagnostopoulou, Angelos Valsamis, Philip Doganis, Pantelis Sopasakis, and Haralambos Sarimveis. 2017. “Jaqpot Quattro: A Novel Computational Web Platform for Modeling and Analysis in Nanoinformatics.” *Journal of Chemical Information and Modeling* 57 (9): 2161–72.

Gajewicz, Agnieszka, Nicole Schaeublin, Bakhtiyor Rasulev, Saber Hussain, Danuta Leszczynska, Tomasz Puzyn, and Jerzy Leszczynski. 2015. “Towards Understanding Mechanisms Governing Cytotoxicity of Metal Oxides Nanoparticles: Hints from Nano-QSAR Studies.” *Nanotoxicology* 9 (3): 313–25.

Jennings, Paul, Thomas Exner, Lucian Farcal, Noffisat Oki, Harry Sarimveis, Philip Doganis, Danyel Jennen, et al. 2018. “Final Definition of Case Studies (Deliverable 1.3),” November. https://doi.org/10.5281/zenodo.1479127.

Norinder, Ulf, Carlsson, Lars, Boyer, Scott, Eklund, Martin. 2014. “Introducing conformal prediction in predictive modeling. A transparent and flexible alternative to applicability domain determination”. *Journal of Chemical Information and Modeling* 23;54(6):1596-603. https://doi.org/10.1021/ci5001168.

Park, Sung Jin, Ogunseitan, Oladele A, Lejano, Raul P. 2014. “Dempster-Shafer theory applied to regulatory decision process for selecting safer alternatives to toxic chemicals in consumer products”. *Integrated Environmental Assessment and Management* 10 (1): 12-21. https://doi.org/10.1002/ieam.1460

Rathman, James F, Yang, Chihae, Zhou, Haojin. 2018. “Dempster-Shafer theory for combining in silico evidence and estimating uncertainty in chemical risk assessment”. *Computational Toxicology* 6: 16-31. http://dx.doi.org/10.1016/j.comtox.2018.03.001

Tetko, Igor V. 2008. “Associative neural network”. *Methods in Molecular Biology* 458: 185-202. doi:10.1007/978-1-60327-101-1_10

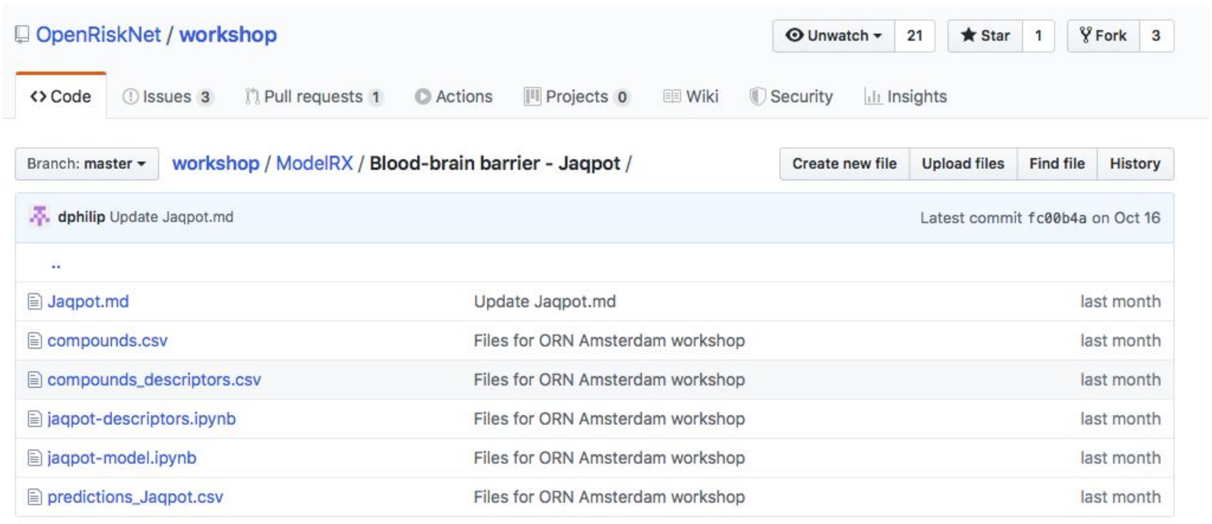
Sushko, Iurii, Novotarskyi, Sergii, Körner, Robert, Pandey, Anil Kumar, Kovalishyn, Vasily V, Prokopenko Volodymyr V, Tetko Igor V. 2010. “Applicability domain for in silico models to achieve accuracy of experimental measurements”. *Journal of Chemometrics* 24 (3-4): 202-208. https://doi.org/10.1002/cem.1296

APPENDIX

Jaqpot

Jaqpot (developed at NTUA) allows users to transform models they create in Python into web services with 1 line of code using the Jaqpotpy package. For more extended reference to the Jaqpotpy package, please refer to the package reference at https://jaqpotpy.readthedocs.io and for Jaqpot in PBPK modelling, please refer to the REVK Case Study.

In order to briefly demonstrate the functionality of Jaqpot, we will present an example of the user creating a new model in Python, making the model as a web service in Jaqpot and finally, using the new model to make predictions. The example was presented at the Modelling Session of the Final OpenRiskNet workshop in Amsterdam and is available here: https://github.com/OpenRiskNet/workshop/tree/master/ModelRX (please note that all services run on the cloud and the CSV files are only provided to users for comparison purposes).

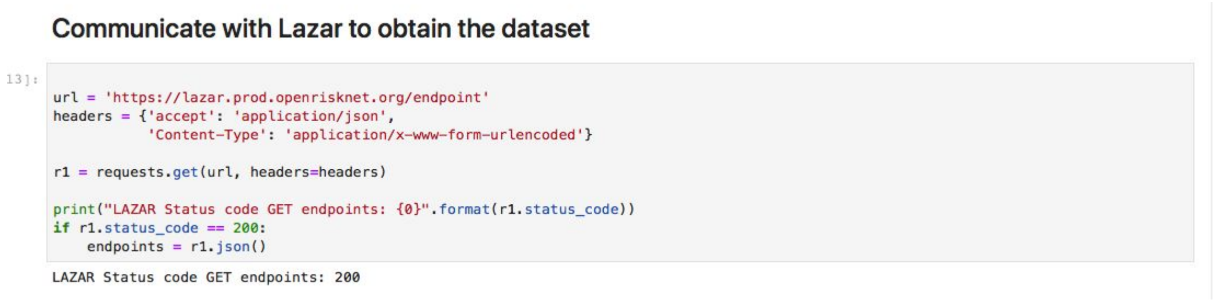


workshop / ModelRX / Blood-brain barrier - Jaqpot /

dphilip Update Jaqpot.md   Latest commit fc00b4a on Oct 16

..		
Jaqpot.md	Update Jaqpot.md	last month
compounds.csv	Files for ORN Amsterdam workshop	last month
compounds_descriptors.csv	Files for ORN Amsterdam workshop	last month
jaqpot-descriptors.ipynb	Files for ORN Amsterdam workshop	last month
jaqpot-model.ipynb	Files for ORN Amsterdam workshop	last month
predictions_Jaqpot.csv	Files for ORN Amsterdam workshop	last month

The example is split in two phases, with respective notebooks. In the first phase (in the https://github.com/OpenRiskNet/workshop/blob/master/ModelRX/Blood-brain%20barrier%20-%20Jaqpot/jaqpot-descriptors.ipynb notebook), users first get the dataset from Lazar:



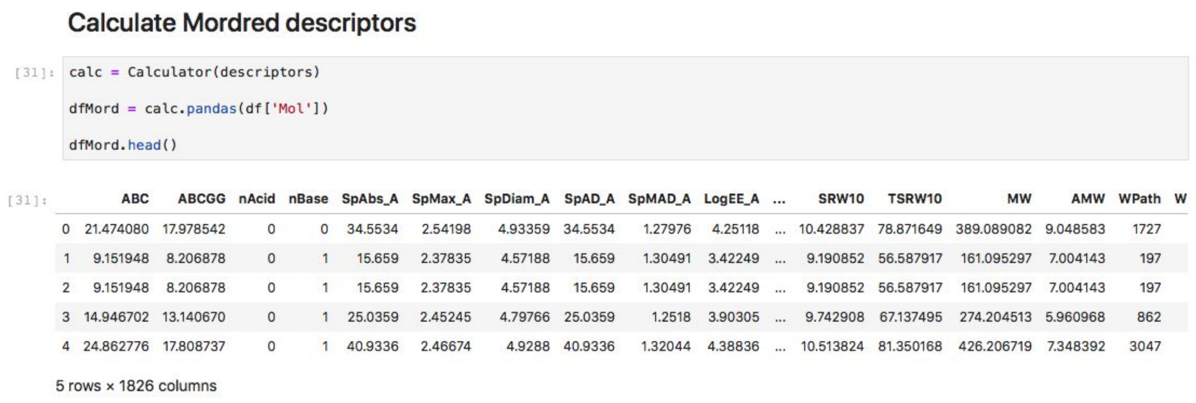
Communicate with Lazar to obtain the dataset

```
13]:
url = 'https://lazar.prod.openrisknet.org/endpoint'
headers = {'accept': 'application/json',
           'Content-Type': 'application/x-www-form-urlencoded'}

r1 = requests.get(url, headers=headers)

print("LAZAR Status code GET endpoints: {0}".format(r1.status_code))
if r1.status_code == 200:
    endpoints = r1.json()
LAZAR Status code GET endpoints: 200
```

After necessary preprocessing steps on the dataset are performed, the user produces Mordred descriptors on the dataset:



Calculate Mordred descriptors

```
[31]: calc = Calculator(descriptors)

      dfMord = calc.pandas(df['Mol'])

      dfMord.head()
```

[31]:

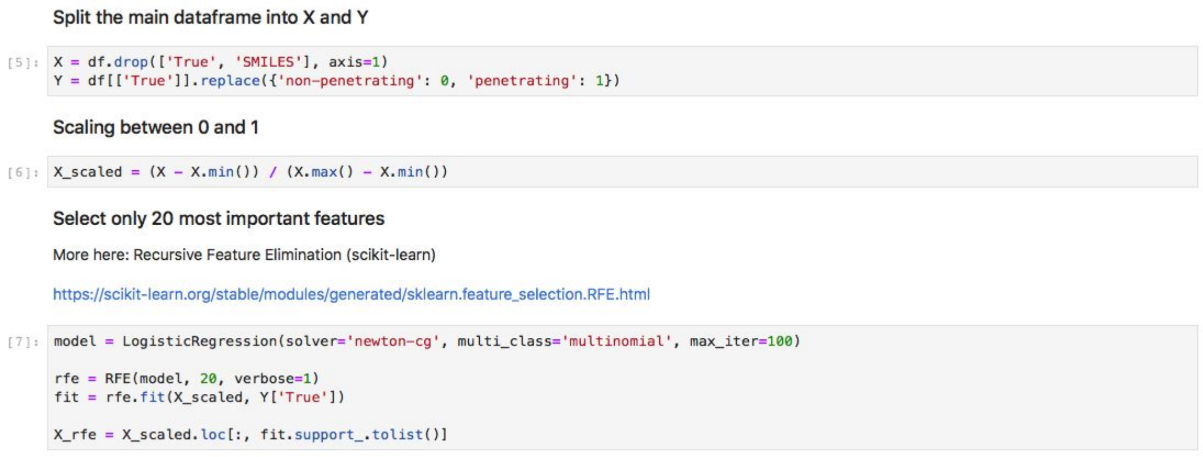
	ABC	ABCGG	nAcid	nBase	SpAbs_A	SpMax_A	SpDiam_A	SpAD_A	SpMAD_A	LogEE_A	...	SRW10	TSRW10	MW	AMW	WPath	W
0	21.474080	17.978542	0	0	34.5534	2.54198	4.93359	34.5534	1.27976	4.25118	...	10.428837	78.871649	389.089082	9.048583	1727	
1	9.151948	8.206878	0	1	15.659	2.37835	4.57188	15.659	1.30491	3.42249	...	9.190852	56.587917	161.095297	7.004143	197	
2	9.151948	8.206878	0	1	15.659	2.37835	4.57188	15.659	1.30491	3.42249	...	9.190852	56.587917	161.095297	7.004143	197	
3	14.946702	13.140670	0	1	25.0359	2.45245	4.79766	25.0359	1.2518	3.90305	...	9.742908	67.137495	274.204513	5.960968	862	
4	24.862776	17.808737	0	1	40.9336	2.46674	4.9288	40.9336	1.32044	4.38836	...	10.513824	81.350168	426.206719	7.348392	3047	

5 rows × 1826 columns

The results are in turn processed to make sure only meaningful results will be used for modelling.

In the second phase, users build their model (available as a Python notebook at https://github.com/OpenRiskNet/workshop/blob/master/ModelRX/Blood-brain%20barrier%20-%20Jaqpot/jaqpot-model.ipynb).

Users develop a predictive model for blood-brain-barrier penetration using Logistic Regression (from scikit-learn[9]), after reducing the number of descriptors used to 20, based on Recursive Feature Elimination (scikit-learn).



Split the main dataframe into X and Y

```
[5]: X = df.drop(['True', 'SMILES'], axis=1)
     Y = df[['True']].replace({'non-penetrating': 0, 'penetrating': 1})
```

Scaling between 0 and 1

```
[6]: X_scaled = (X - X.min()) / (X.max() - X.min())
```

Select only 20 most important features

More here: Recursive Feature Elimination (scikit-learn)

https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html

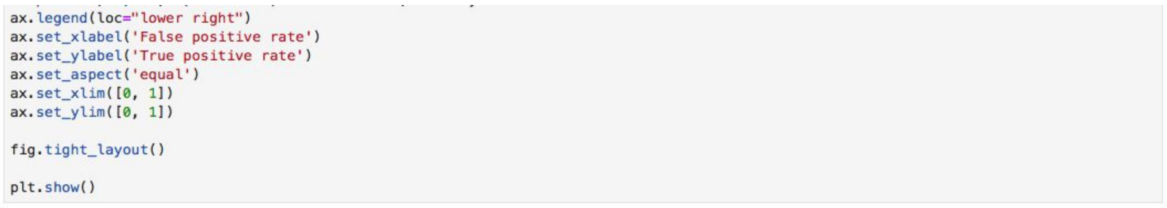
```
[7]: model = LogisticRegression(solver='newton-cg', multi_class='multinomial', max_iter=100)

     rfe = RFE(model, 20, verbose=1)
     fit = rfe.fit(X_scaled, Y['True'])

     X_rfe = X_scaled.loc[:, fit.support_.tolist()]
```

After evaluating model performance through the Confusion matrix, the Positive predictive value, the Negative predictive value and the ROC plot (shown below) the model is accepted.

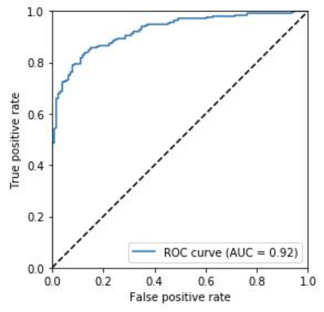
[9] https://scikit-learn.org



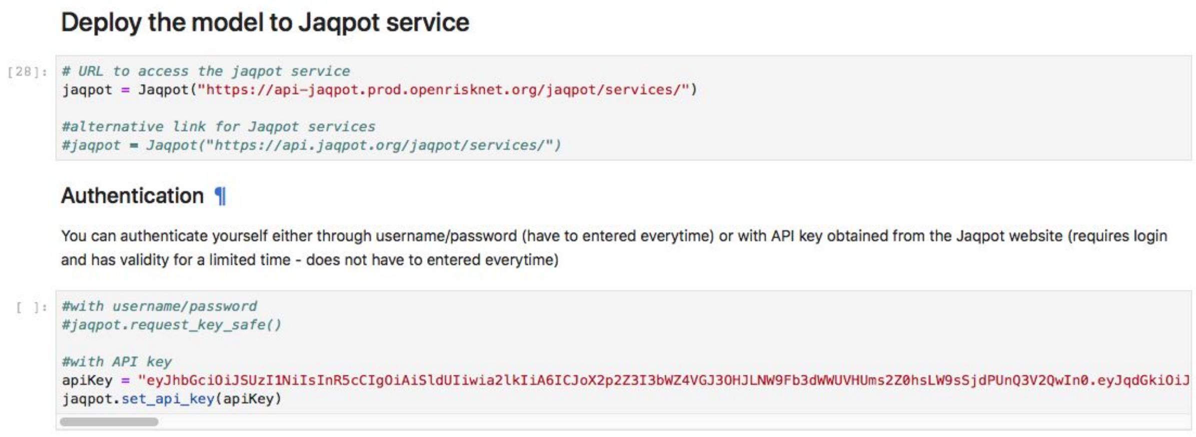
```
ax.legend(loc="lower right")
ax.set_xlabel('False positive rate')
ax.set_ylabel('True positive rate')
ax.set_aspect('equal')
ax.set_xlim([0, 1])
ax.set_ylim([0, 1])

fig.tight_layout()

plt.show()
```



The trained model can be added to Jaqpot using the Jaqpotpy library that was specially written by the NTUA team for the purpose of uploading models to Jaqpot. First the user needs to define which Jaqpot instance actions will refer to and authenticate access to it.



Deploy the model to Jaqpot service

```
[28]: # URL to access the jaqpot service
      jaqpot = Jaqpot("https://api-jaqpot.prod.openrisknet.org/jaqpot/services/")

      #alternative link for Jaqpot services
      #jaqpot = Jaqpot("https://api.jaqpot.org/jaqpot/services/")
```

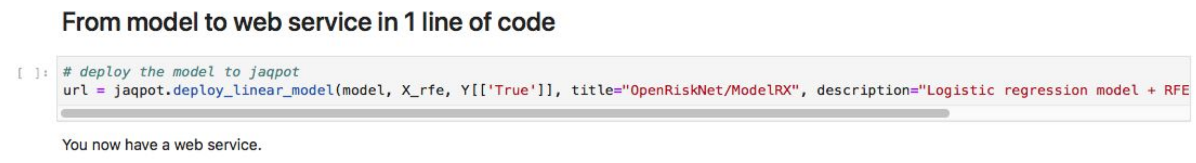
Authentication ¶

You can authenticate yourself either through username/password (have to entered everytime) or with API key obtained from the Jaqpot website (requires login and has validity for a limited time - does not have to entered everytime)

```
[ ]: #with username/password
     #jaqpot.request_key_safe()

     #with API key
     apiKey = "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJoX2p2Z3I3bWZ4VGJ3OHJLNW9Fb3dWWUVHUms2Z0hsLW9sSjdPUnQ3V2QwIn0.eyJqdGkiOiJ
     jaqpot.set_api_key(apiKey)
```

After that, uploading a ready model is a matter of 1 line of code, where the user defines the details for the model and necessary references:

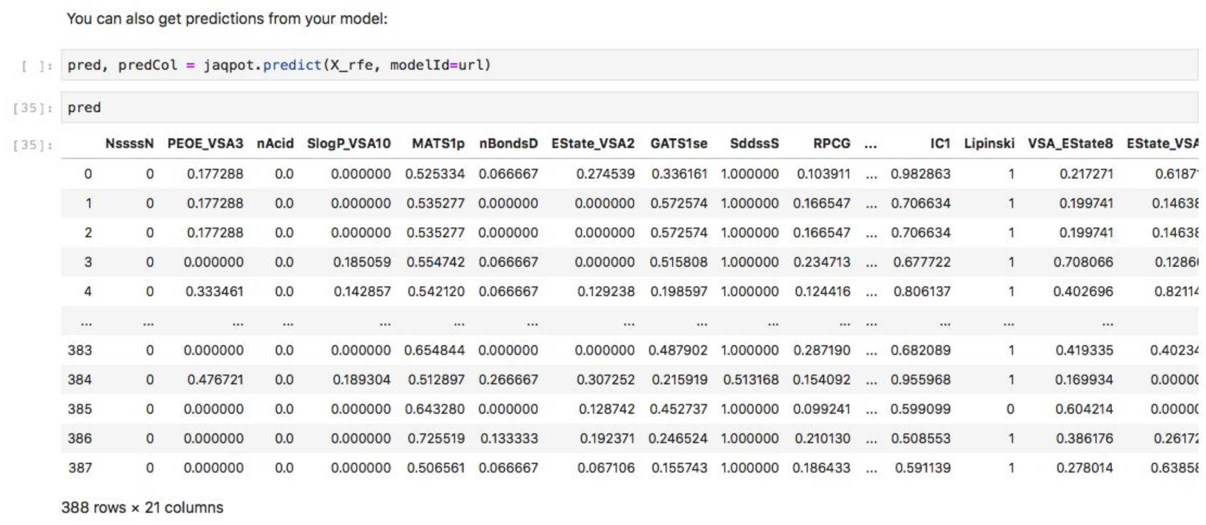


From model to web service in 1 line of code

```
[ ]: # deploy the model to jaqpot
     url = jaqpot.deploy_linear_model(model, X_rfe, Y[['True']], title="OpenRiskNet/ModelRX", description="Logistic regression model + RFE
```

You now have a web service.

Now that the model has become a web service, getting predictions from it is simple:


You can also get predictions from your model:

```
[ ]: pred, predCol = jaqpot.predict(X_rfe, modelId=url)
```

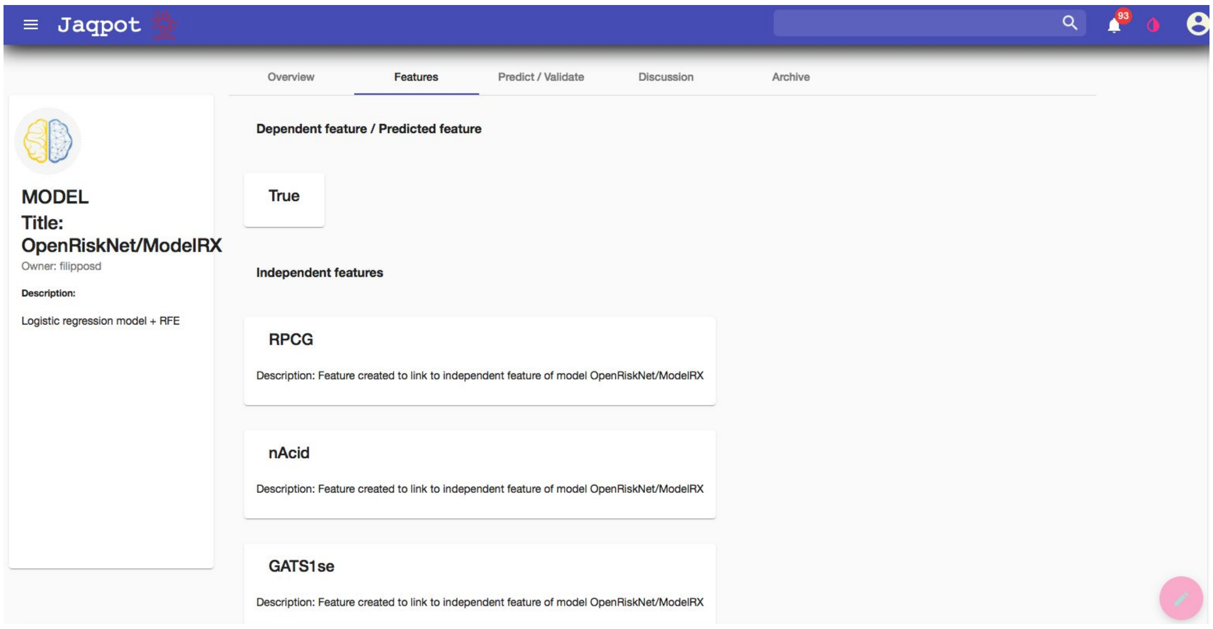
```
[35]: pred
```

[35]:

	NssssN	PEOE_VSA3	nAcid	SlogP_VSA10	MATS1p	nBondsD	EState_VSA2	GATS1se	SddssS	RPCG	...	IC1	Lipinski	VSA_EState8	EState_VSA
0	0	0.177288	0.0	0.000000	0.525334	0.066667	0.274539	0.336161	1.000000	0.103911	...	0.982863	1	0.217271	0.6187
1	0	0.177288	0.0	0.000000	0.535277	0.000000	0.000000	0.572574	1.000000	0.166547	...	0.706634	1	0.199741	0.1463
2	0	0.177288	0.0	0.000000	0.535277	0.000000	0.000000	0.572574	1.000000	0.166547	...	0.706634	1	0.199741	0.1463
3	0	0.000000	0.0	0.185059	0.554742	0.066667	0.000000	0.515808	1.000000	0.234713	...	0.677722	1	0.708066	0.1286
4	0	0.333461	0.0	0.142857	0.542120	0.066667	0.129238	0.198597	1.000000	0.124416	...	0.806137	1	0.402696	0.8211
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
383	0	0.000000	0.0	0.000000	0.654844	0.000000	0.000000	0.487902	1.000000	0.287190	...	0.682089	1	0.419335	0.4023
384	0	0.476721	0.0	0.189304	0.512897	0.266667	0.307252	0.215919	0.513168	0.154092	...	0.955968	1	0.169934	0.0000
385	0	0.000000	0.0	0.000000	0.643280	0.000000	0.128742	0.452737	1.000000	0.099241	...	0.599099	0	0.604214	0.0000
386	0	0.000000	0.0	0.000000	0.725519	0.133333	0.192371	0.246524	1.000000	0.210130	...	0.508553	1	0.386176	0.2617
387	0	0.000000	0.0	0.000000	0.506561	0.066667	0.067106	0.155743	1.000000	0.186433	...	0.591139	1	0.278014	0.6385

388 rows × 21 columns

Please note that the model is now also available over the Jaqpot user interface at https://ui-jaqpot.prod.openrisknet.org for users to inspect:



MODEL
Title:
OpenRiskNet/ModelRX
Owner: filipposd
Description:
Logistic regression model + RFE

Dependent feature / Predicted feature

True

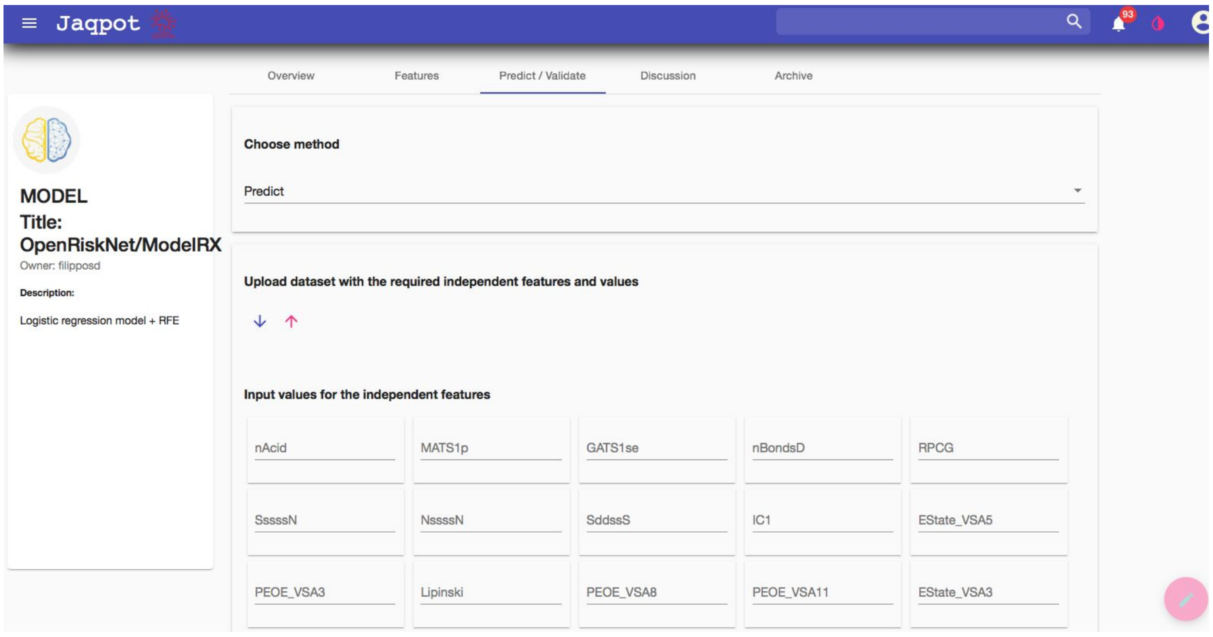
Independent features

RPCG
Description: Feature created to link to independent feature of model OpenRiskNet/ModelRX

nAcid
Description: Feature created to link to independent feature of model OpenRiskNet/ModelRX

GATS1se
Description: Feature created to link to independent feature of model OpenRiskNet/ModelRX

And also make predictions on their own data, either typed in or provided over a CSV file, using the auto-generated template provided by Jaqpot:



MODEL
Title:
OpenRiskNet/ModelRX
Owner: filipposd

Description:

Logistic regression model + RFE

Choose method

Predict

Upload dataset with the required independent features and values

Input values for the independent features

nAcid	MATS1p	GATS1se	nBondsD	RPCG
SssssN	NssssN	SddssS	IC1	EState_VSA5
PEOE_VSA3	Lipinski	PEOE_VSA8	PEOE_VSA11	EState_VSA3

A more detailed presentation of Jaqpot’s offering was presented during the webinar “Demonstration on OpenRiskNet approach on modelling for prediction or read across”, available at https://openrisknet.org/events/67/, where both the recording and the slides are available.

The documentation for the Jaqpot 5 API has been made available over Swagger at https://api-Jaqpot.prod.openrisknet.org/Jaqpot/swagger/.

Lazar

lazar (Lazy Structure-Activity Relationships developed at JGU/IST) is a framework based on Ruby libraries. It also depends on a couple of external programs and libraries. All required libraries will be installed with the `gem install lazar` command. To build this prediction model in lazar following steps where required. Executing the following commands either from an interactive Ruby shell or a Ruby script.

1. Create the training dataset

Create a CSV file with two columns. The first line should contain either SMILES or InChI (first column) and the endpoint (second column). The first column should contain either the SMILES or InChI of the training compounds, the second column the training compounds toxic activities (qualitative or quantitative). Add metadata to a JSON file with the same basename containing the fields "species", "endpoint", "source".

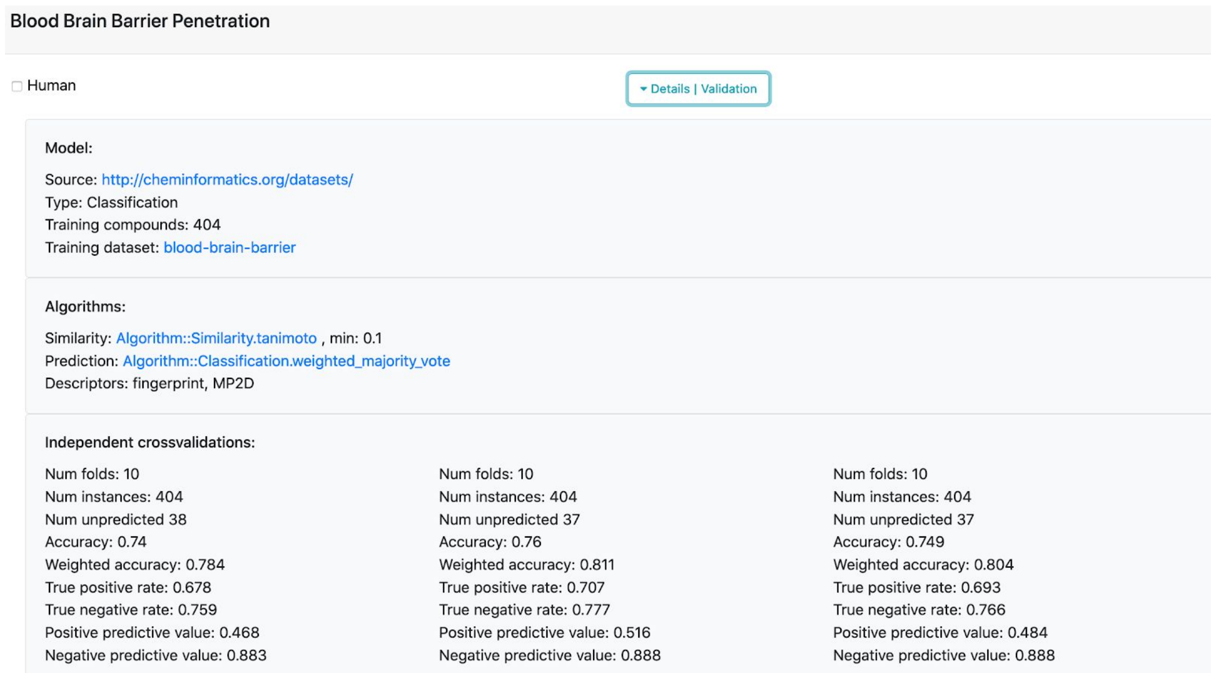
```
training_dataset = Dataset.from_csv_file "Blood_Brain_Barrier_Penetration-Human.csv"
```

2. Create and validate the lazar model with default algorithms and parameters

```
validated_model = Model::Validation.create_from_csv_file Blood_Brain_Barrier_Penetration-Human.csv
```

This command will create a *lazar* model and validate it with three independent 10-fold cross validations.

Following screenshots represents the model details and validation results as shown in the *lazar* GUI (https://lazar.prod.openrisknet.org/predict).



Blood Brain Barrier Penetration

Human

Model:

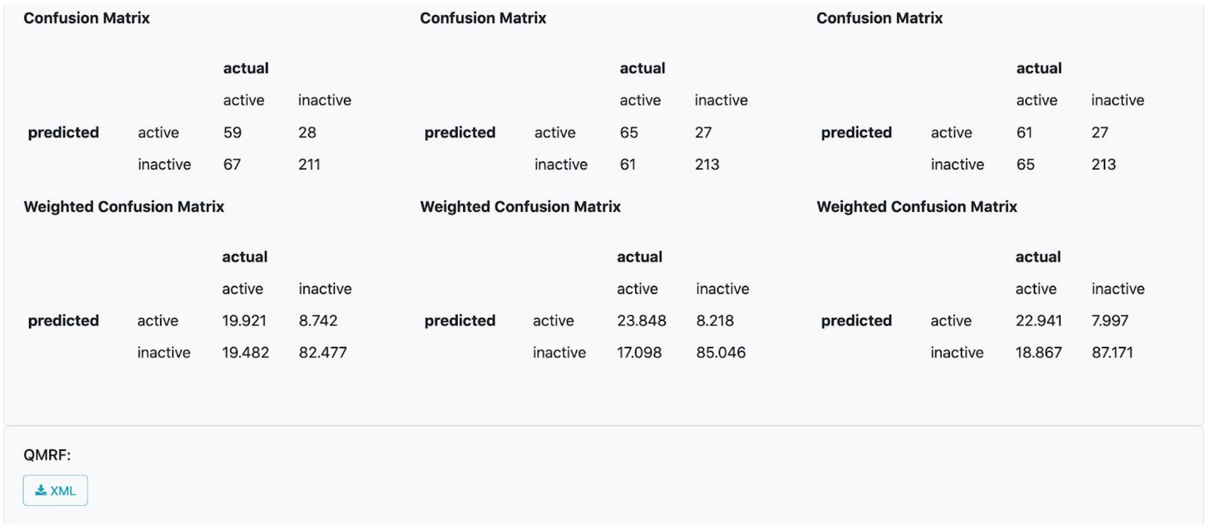
Source: http://cheminformatics.org/datasets/
Type: Classification
Training compounds: 404
Training dataset: blood-brain-barrier

Algorithms:

Similarity: Algorithm::Similarity.tanimoto , min: 0.1
Prediction: Algorithm::Classification.weighted_majority_vote
Descriptors: fingerprint, MP2D

Independent crossvalidations:

Num folds: 10	Num folds: 10	Num folds: 10
Num instances: 404	Num instances: 404	Num instances: 404
Num unpredicted 38	Num unpredicted 37	Num unpredicted 37
Accuracy: 0.74	Accuracy: 0.76	Accuracy: 0.749
Weighted accuracy: 0.784	Weighted accuracy: 0.811	Weighted accuracy: 0.804
True positive rate: 0.678	True positive rate: 0.707	True positive rate: 0.693
True negative rate: 0.759	True negative rate: 0.777	True negative rate: 0.766
Positive predictive value: 0.468	Positive predictive value: 0.516	Positive predictive value: 0.484
Negative predictive value: 0.883	Negative predictive value: 0.888	Negative predictive value: 0.888



Confusion Matrix

		actual	
		active	inactive
predicted	active	59	28
	inactive	67	211

Confusion Matrix

		actual	
		active	inactive
predicted	active	65	27
	inactive	61	213

Confusion Matrix

		actual	
		active	inactive
predicted	active	61	27
	inactive	65	213

Weighted Confusion Matrix

		actual	
		active	inactive
predicted	active	19.921	8.742
	inactive	19.482	82.477

Weighted Confusion Matrix

		actual	
		active	inactive
predicted	active	23.848	8.218
	inactive	17.098	85.046

Weighted Confusion Matrix

		actual	
		active	inactive
predicted	active	22.941	7.997
	inactive	18.867	87.171

QMRF:

Experiment with other algorithms

You can pass algorithm specifications as parameters to the Model::Validation.create_from_csv_file command. Algorithms for descriptors, similarity calculations, feature_selection and local models are specified in the algorithm parameter. Unspecified algorithms and parameters are substituted by default values.

The example below selects

MP2D fingerprint descriptors

Tanimoto similarity with a threshold of 0.1

no feature selection

weighted majority vote predictions

```
algorithms = {
:descriptors => { # descriptor algorithm
  :method => "fingerprint", # fingerprint descriptors
  :type => "MP2D" # fingerprint type, e.g. FP4, MACCS
},
:similarity => { # similarity algorithm
  :method => "Algorithm::Similarity.tanimoto",
  :min => 0.1 # similarity threshold for neighbors
},
:feature_selection => nil, # no feature selection
:prediction => { # local modelling algorithm
  :method => "Algorithm::Classification.weighted_majority_vote",
},
}
training_dataset = Dataset.from_csv_file "Blood_Brain_Barrier_Penetration-Human.csv"
model = Model::Validation.create  training_dataset: training_dataset, algorithms: algorithms
```

lazar is implemented as a RESTful service as well as a graphical user interface.

REST API: https://lazar.prod.openrisknet.org

GUI: https://lazar.prod.openrisknet.org/predict

WEKA

In this section, we present a sample use case for employing the JGU Weka REST API for creating a predictive model based on a user-provided dataset. The model can be evaluated based on the already provided dataset or the user can use the generated model for evaluating a different dataset split kept as a test set. The web version of the JGU Weka REST API can be explored at the URL https://jguweka.prod.openrisknet.org/.

The BBB penetration dataset contains SMILES representations of chemical compounds and whether the particular compound can cross the blood-brain-barrier. In order to train a model for predicting the blood-brain-barrier penetrating or non-penetrating chemicals, we need some features for the chemical compounds, based on which we can train a model. The user can create the feature based dataset compatible with Weka ARFF format in any desired way. It is possible to use existing chemoinformatics libraries, e.g. CDK, Mordred, RDKit, etc. for extracting features from the chemical compounds, however, since the other services in this case study already use one or the other mentioned libraries, we will use a graph mining based feature extraction algorithm for chemical compounds in order to use features which provide a different perspective to the problem.

The existing libraries use handcrafted features which have been identified by chemical experts due to certain properties of these chemical structures/features. It has been demonstrated that elaborate patterns can also be used to summarize ground features. Using patterns which summarize several ground features also has the potential of revealing latent information not present in any ground feature. The **La**tent **St**ructure **P**attern **M**ining (LAST-PM) algorithm by Maunz et al.[10] aims to extract ground features from chemical compounds based on embedding relationships between individual patterns while taking into consideration the frequency and/or correlation of the patterns.

The LAST-PM implementation depends on a number of libraries, therefore, the algorithm has been containerised and a fully functional Docker image has been made available at https://hub.docker.com/r/jguweka/chem_descriptor_miner. The feature extraction application takes as input two files, (i) an “smi” file containing SMILES formatted chemical compounds, and (ii) a “class” file with the target/class variable corresponding to each chemical compound in the SMILES file. All the processing steps are then handled by the containerised application and feature extraction, conversion of chemical features from graph data format into SMARTS notation, and finally, the creation of a Weka ARFF file with the extracted features is carried out by the containerised application.

Assuming that we have separated the SMILES formatted chemical compounds and the target variable of the BBB dataset into an *smi* file and a *class* file, respectively, we can initiate the LAST-PM based feature extraction process using the following command

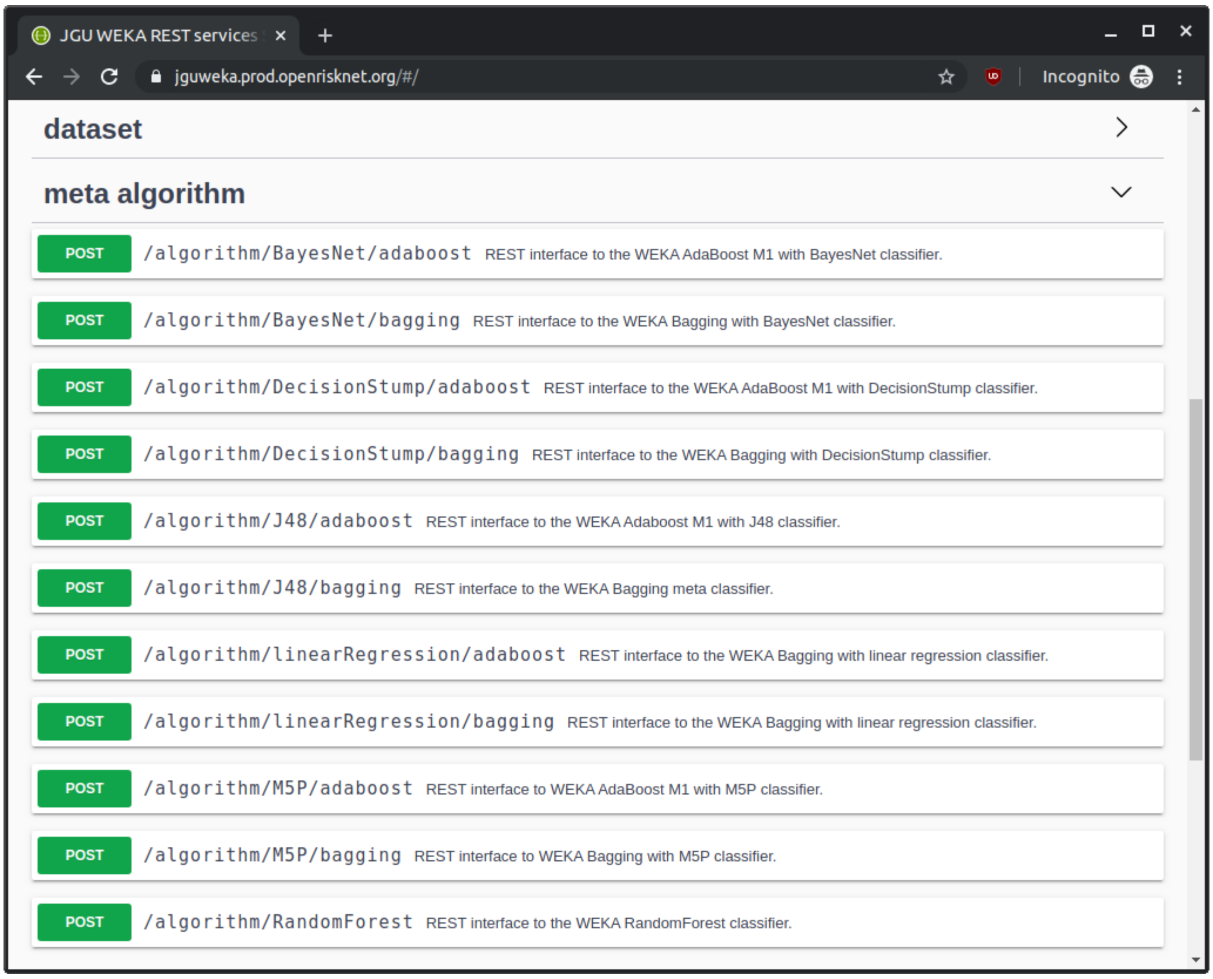
[10] Maunz A., Helma C., Cramer T., Kramer S. (2010) Latent Structure Pattern Mining. In: Balcázar J.L., Bonchi F., Gionis A., Sebag M. (eds) Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2010. Lecture Notes in Computer Science, vol 6322. Springer, Berlin, Heidelberg

to start the chem_descriptor_miner Docker container.

```
docker run -ti -v /path/to/data_dir:/work jguweka/chem_descriptor_miner /bin/bash
```

Here, the `/path/to/data_dir` is the host computer's directory containing the smi and class files. The resulting ARFF files are also saved to the same directory on the host machine. Running the above command starts the Docker container, which contains the main script in the `chem-descriptors` directory and the smi and class files are linked from the `/path/to/data_dir` directory in the host machine to the `work` directory in the Docker container. The different program options, e.g. dataset name, minimum descriptor frequency, number of ground features, SMARTS wildcarding and aromatic annotations, etc. can be adjusted by editing the `script.sh` BASH script in the `chem-descriptors` directory before running the task of feature extraction. Once the process completes, the ARFF file is created in the host computer's data directory `/path/to/data_dir`.

Here we assume that the BBB dataset has been successfully processed by the `jguweka/chem_descriptor_miner` containerised application. The Weka REST API exposes a number of machine learning algorithms. We will use the Random Forest algorithm for this demonstration using the default parameters. Apart from appearing under the *algorithm* category, the ensemble methods are also grouped under the *meta algorithm* category.



dataset

meta algorithm

POST /algorithm/BayesNet/adaboost REST interface to the WEKA AdaBoost M1 with BayesNet classifier.

POST /algorithm/BayesNet/bagging REST interface to the WEKA Bagging with BayesNet classifier.

POST /algorithm/DecisionStump/adaboost REST interface to the WEKA AdaBoost M1 with DecisionStump classifier.

POST /algorithm/DecisionStump/bagging REST interface to the WEKA Bagging with DecisionStump classifier.

POST /algorithm/J48/adaboost REST interface to the WEKA Adaboost M1 with J48 classifier.

POST /algorithm/J48/bagging REST interface to the WEKA Bagging meta classifier.

POST /algorithm/linearRegression/adaboost REST interface to the WEKA Bagging with linear regression classifier.

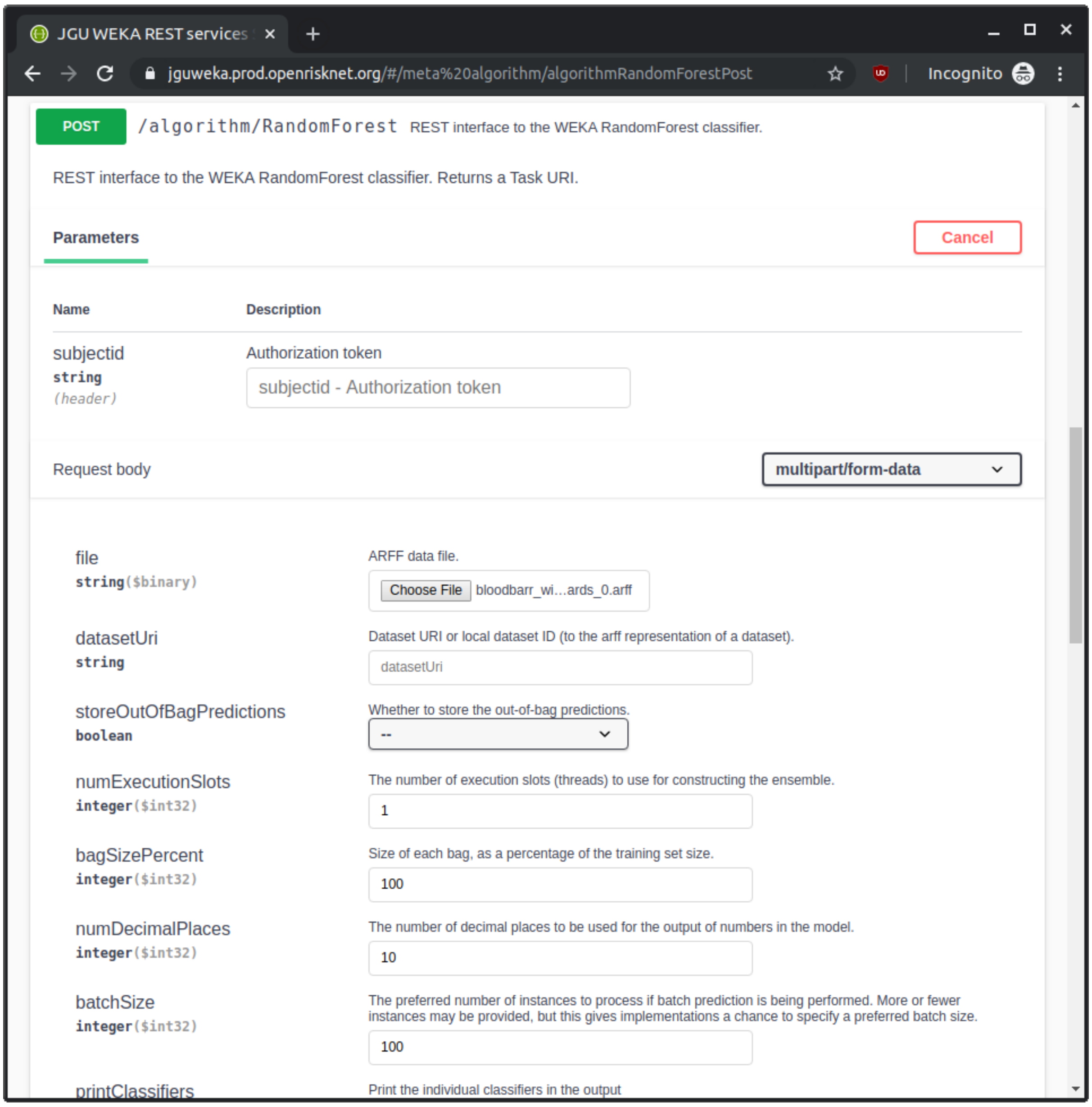
POST /algorithm/linearRegression/bagging REST interface to the WEKA Bagging with linear regression classifier.

POST /algorithm/M5P/adaboost REST interface to WEKA AdaBoost M1 with M5P classifier.

POST /algorithm/M5P/bagging REST interface to WEKA Bagging with M5P classifier.

POST /algorithm/RandomForest REST interface to the WEKA RandomForest classifier.

Selecting an algorithm entry opens the details for the particular algorithm. Clicking the “Try it out” button allows the user to upload a dataset or provide a URI and run the algorithm for the given data. The interface provides default values for the different parameters required for the selected algorithm. Here, we are using the default values.


JGU WEKA REST services
jguweka.prod.openrisknet.org/#/meta%20algorithm/algorithmRandomForestPost
Incognito
POST
Cancel

/algorithm/RandomForest REST interface to the WEKA RandomForest classifier.

REST interface to the WEKA RandomForest classifier. Returns a Task URI.

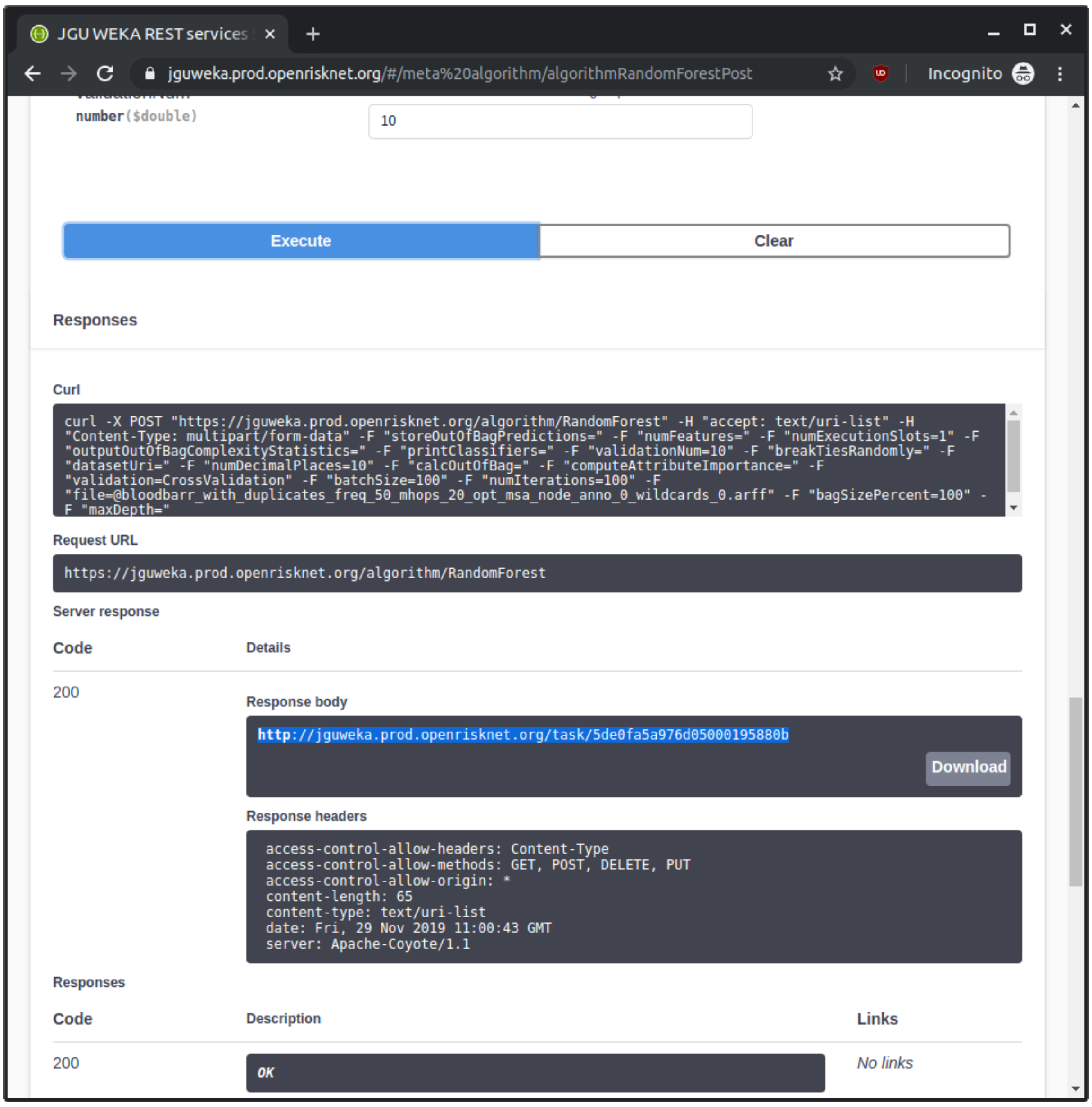
Parameters

Name	Description
subjectid string (header)	Authorization token subjectid - Authorization token

Request body: multipart/form-data

file string($binary)	ARFF data file. Choose File bloodbarr_wi…ards_0.arff
datasetUri string	Dataset URI or local dataset ID (to the arff representation of a dataset). datasetUri
storeOutOfBagPredictions boolean	Whether to store the out-of-bag predictions. --
numExecutionSlots integer($int32)	The number of execution slots (threads) to use for constructing the ensemble. 1
bagSizePercent integer($int32)	Size of each bag, as a percentage of the training set size. 100
numDecimalPlaces integer($int32)	The number of decimal places to be used for the output of numbers in the model. 10
batchSize integer($int32)	The preferred number of instances to process if batch prediction is being performed. More or fewer instances may be provided, but this gives implementations a chance to specify a preferred batch size. 100
printClassifiers	Print the individual classifiers in the output

Clicking the "Execute" button creates a task for model creation. The task link is returned as a response to the execute command.


JGU WEKA REST services
jguweka.prod.openrisknet.org/#/meta%20algorithm/algorithmRandomForestPost
Incognito
Execute
Clear
Download

number($double) 10

Responses

Curl

```
curl -X POST "https://jguweka.prod.openrisknet.org/algorithm/RandomForest" -H "accept: text/uri-list" -H
"Content-Type: multipart/form-data" -F "storeOutOfBagPredictions=" -F "numFeatures=" -F "numExecutionSlots=1" -F
"outputOutOfBagComplexityStatistics=" -F "printClassifiers=" -F "validationNum=10" -F "breakTiesRandomly=" -F
"datasetUri=" -F "numDecimalPlaces=10" -F "calcOutOfBag=" -F "computeAttributeImportance=" -F
"validation=CrossValidation" -F "batchSize=100" -F "numIterations=100" -F
"file=@bloodbarr_with_duplicates_freq_50_mhops_20_opt_msa_node_anno_0_wildcards_0.arff" -F "bagSizePercent=100" -
F "maxDepth="
```

Request URL

```
https://jguweka.prod.openrisknet.org/algorithm/RandomForest
```

Server response

Code	Details
200	Response body

```
http://jguweka.prod.openrisknet.org/task/5de0fa5a976d050001958800b
```

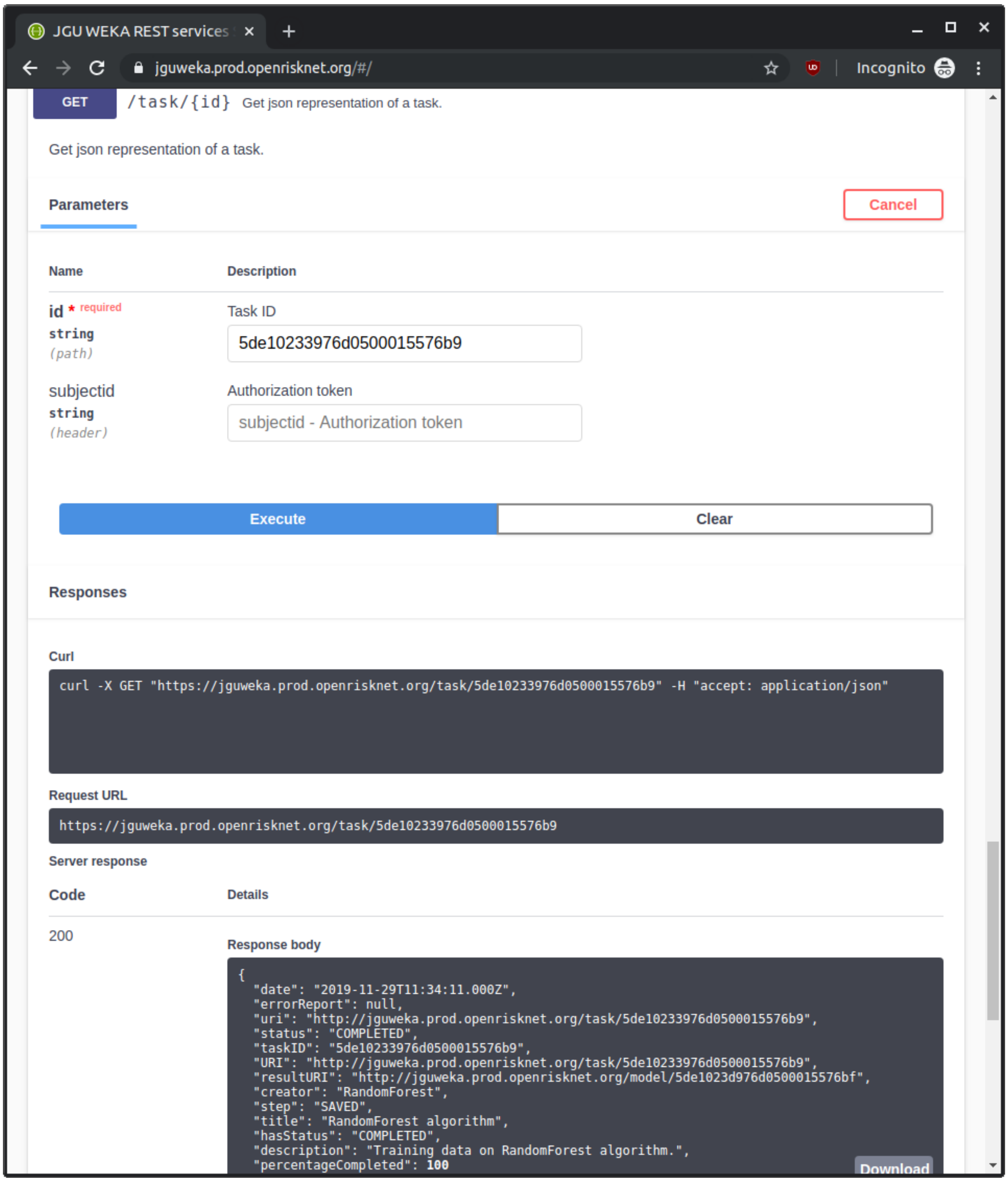
Response headers

```
access-control-allow-headers: Content-Type
access-control-allow-methods: GET, POST, DELETE, PUT
access-control-allow-origin: *
content-length: 65
content-type: text/uri-list
date: Fri, 29 Nov 2019 11:00:43 GMT
server: Apache-Coyote/1.1
```

Responses

Code	Description	Links
200	*OK*	*No links*

The task ID can be used to query the server about the state of the modelling task.



/task/{id} Get json representation of a task.

Get json representation of a task.

Parameters

Name	Description
id * required string (path)	Task ID 5de10233976d0500015576b9
subjectid string (header)	Authorization token subjectid - Authorization token

Responses

Curl

```
curl -X GET "https://jguweka.prod.openrisknet.org/task/5de10233976d0500015576b9" -H "accept: application/json"
```

Request URL

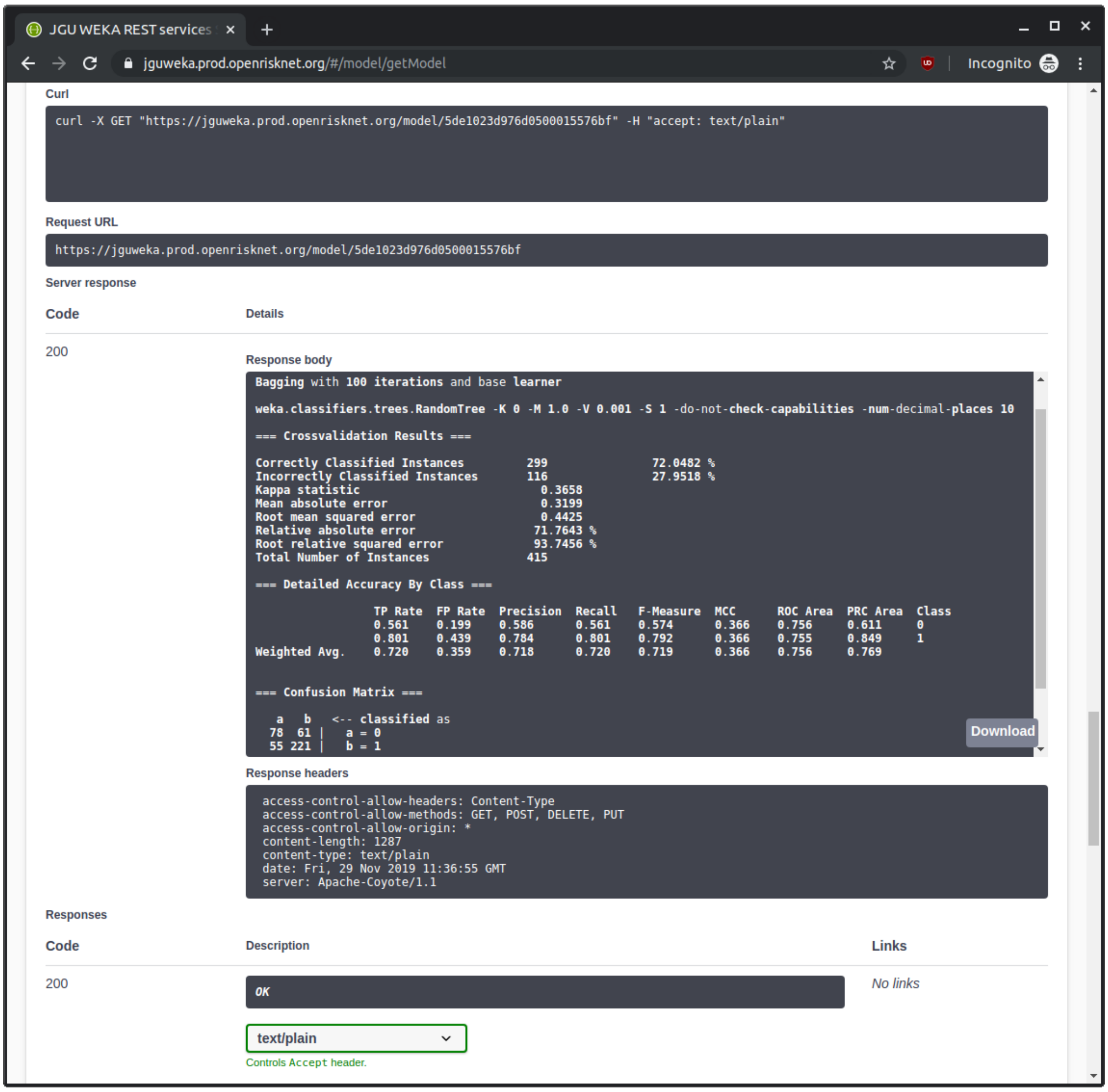
```
https://jguweka.prod.openrisknet.org/task/5de10233976d0500015576b9
```

Server response

Code	Details
200	Response body

```
{
  "date": "2019-11-29T11:34:11.000Z",
  "errorReport": null,
  "uri": "http://jguweka.prod.openrisknet.org/task/5de10233976d0500015576b9",
  "status": "COMPLETED",
  "taskID": "5de10233976d0500015576b9",
  "URI": "http://jguweka.prod.openrisknet.org/task/5de10233976d0500015576b9",
  "resultURI": "http://jguweka.prod.openrisknet.org/model/5de1023d976d0500015576bf",
  "creator": "RandomForest",
  "step": "SAVED",
  "title": "RandomForest algorithm",
  "hasStatus": "COMPLETED",
  "description": "Training data on RandomForest algorithm.",
  "percentageCompleted": 100
```

Once the server returns the COMPLETED status for the task, the model can be retrieved based on the model ID given under resultURI. The generated model is presented in human readable form but it can also be downloaded as a JSON object.



Curl

```
curl -X GET "https://jguweka.prod.openrisknet.org/model/5de1023d976d0500015576bf" -H "accept: text/plain"
```

Request URL

```
https://jguweka.prod.openrisknet.org/model/5de1023d976d0500015576bf
```

Server response

Code | Details

200

Response body

```
Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities -num-decimal-places 10

=== Crossvalidation Results ===

Correctly Classified Instances         299               72.0482 %
Incorrectly Classified Instances       116               27.9518 %
Kappa statistic                          0.3658
Mean absolute error                      0.3199
Root mean squared error                  0.4425
Relative absolute error                 71.7643 %
Root relative squared error             93.7456 %
Total Number of Instances              415

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.561    0.199    0.586      0.561    0.574      0.366    0.756     0.611     0
                 0.801    0.439    0.784      0.801    0.792      0.366    0.755     0.849     1
Weighted Avg.    0.720    0.359    0.718      0.720    0.719      0.366    0.756     0.769


=== Confusion Matrix ===

   a   b   <-- classified as
  78  61 |   a = 0
  55 221 |   b = 1
```

Response headers

```
access-control-allow-headers: Content-Type
access-control-allow-methods: GET, POST, DELETE, PUT
access-control-allow-origin: *
content-length: 1287
content-type: text/plain
date: Fri, 29 Nov 2019 11:36:55 GMT
server: Apache-Coyote/1.1
```

Responses

Code	Description	Links
200	*OK*	*No links*

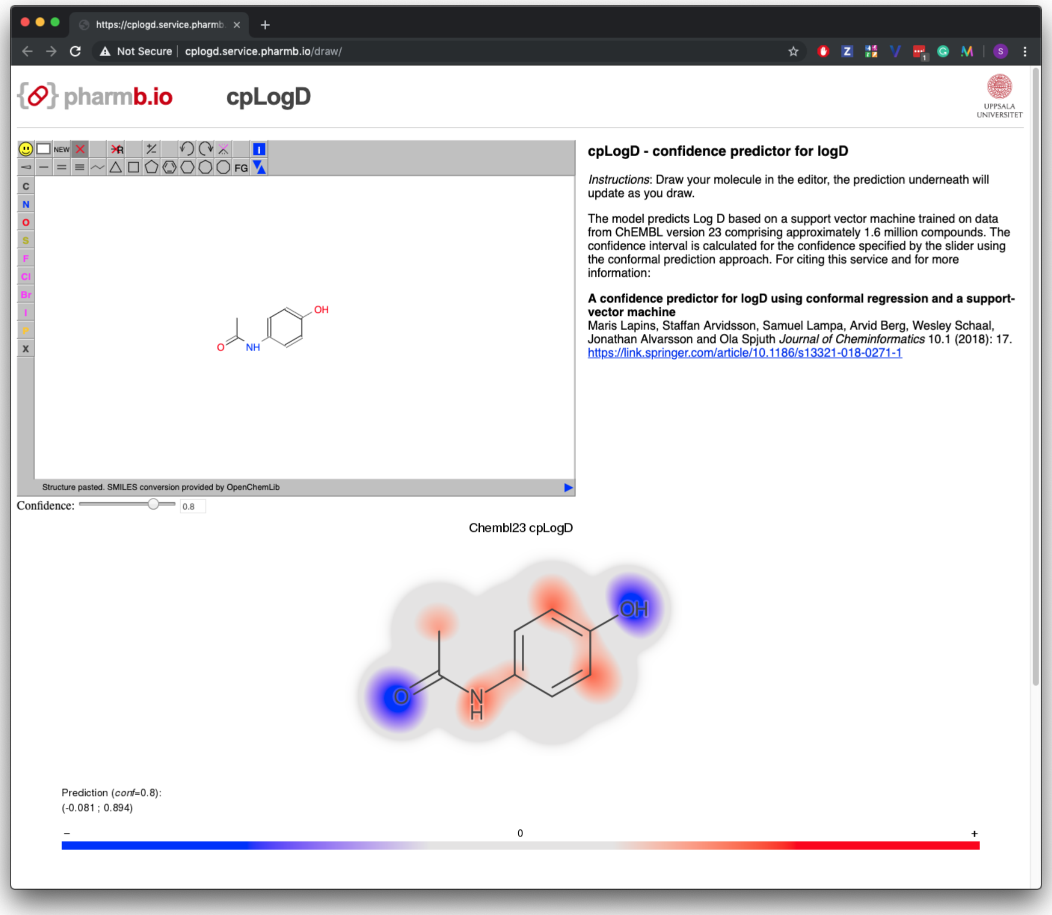
text/plain

Controls Accept header.

In this case, the generated model was able to achieve a 72% accuracy. The outlined steps can also be carried out using REST calls through a Jupyter Notebook or in any other scripts the user is writing for their predictive task. The REST calls for each step are also shown in the Swagger UI based front end.

CPSign

CPSign is a licensed software co-developed by the UU team. This software brings together Conformal Prediction and cheminformatics, accessible through a Java API, command line interface and a Web UI. Currently it supports loading chemistry from various formats, basic filtration of data, descriptor generation using the Signatures descriptor and predictive modeling using Transductive (TCP) and Inductive conformal prediction (ACP and CCP) as well as Cross Venn-ABERS prediction (CVAP). Models can be trained from a web UI as well as a OpenAPI documented REST API. Currently the REST API supports uploading of license files, datasets and training of Venn-ABERS based classification models. Since CPSign is a licensed software the functionality requires authentication using Keycloak. The web UI is located at the URL: http://modelingweb.prod.openrisknet.org/ and the Swagger UI for the OpenAPI definition is available at the URL: http://modelingweb.prod.openrisknet.org/swagger-ui/. Trained models can, aside from previous modes of access, be deployed as microservices in OpenShift and expose a REST API described using OpenAPI. Each microservice also include a GUI where users can load molecules or draw them on their own, continuously making predictions as the molecule is edited in the GUI. An example of the drawing GUI is shown below, for a model predicting the LogD value. Colouring of the atoms show how individual atoms contribute to the prediction (blue contribute towards a lower LogD and red towards an increased LogD).


https://cplogd.service.pharmb
Not Secure | cplogd.service.pharmb.io/draw/
pharmb.io
cpLogD
UPPSALA UNIVERSITET
NEW
FG
C
N
O
S
F
Cl
Br
I
X
OH
O
NH
N
H
Structure pasted. SMILES conversion provided by OpenChemLib
Confidence:
0.8
Chembl23 cpLogD
–
0
+

cpLogD - confidence predictor for logD

Instructions: Draw your molecule in the editor, the prediction underneath will update as you draw.

The model predicts Log D based on a support vector machine trained on data from ChEMBL version 23 comprising approximately 1.6 million compounds. The confidence interval is calculated for the confidence specified by the slider using the conformal prediction approach. For citing this service and for more information:

A confidence predictor for logD using conformal regression and a support-vector machine
Maris Lapins, Staffan Arvidsson, Samuel Lampa, Arvid Berg, Wesley Schaal, Jonathan Alvarsson and Ola Spjuth *Journal of Cheminformatics* 10.1 (2018): 17.
https://link.springer.com/article/10.1186/s13321-018-0271-1

Prediction (*conf*=0.8):
(-0.081 ; 0.894)