650.025 – Machine Learning and Deep Learning Winter Term, 2021/22

Bio-activity estimation

Magnus Offermanns 01360639

Outline and Motivation

Drug research is often supported by computational analysis. One good indicator if a drug is effective at the right place is to monitor its bio activity towards the targeted human protein. This bio activity can be determined experimentally in laboratories, which is costly. To reduce the experimental workload, a filtering in which we eliminate molecules that are unlikely to match to the proteins is therefore very valuable. Therefore we want to implement a machine-learning algorithms to predict the bio-activity of molecules to wards a protein that is called *beta-lactamase*.

As features we use descriptors calculated from the *Mol* representation of the molecules which is a standard class in the Rdkit.Chem. Therefore we use the Mordred a molecular descriptor calculator which calculates about 1600 descriptors calculated from the 2D and 3D representation of the molecules.
We implemented three estimation algorithms: A random forest, a multi-layer-perceptron both using scikit-learn and a deep neural-net using Keras. Lastly we compare the approaches and point out difficulties in training and their consequences in the estimation performance.

In section 1 we describe the Dataset, it's structure and its complexities. In section 1.1 we describe the descriptors calculated with Mordred and their meaning and how low quality features were eliminated. Then we give an overview over the code in section 2. In the Evaluation section 4 we compare the performances of the different approaches and discuss how the estimation accuracy could be improved upon. Lastly in section 5 we discuss the learnings of this project and steps that could have been implemented with additional time at hand.

1 Dataset

The dataset was compiled by a Data-science Youtuber and was compiled from data of the bio-informatics database ChEMBL. In table 1 one can see a representative part of the dataset consisting of 8 Molecules. The first column named *molecule_chembl_id* is a unique identifier of each molecule. The second column called *canonical_smiles* is a string representation of the structural makeup of the molecule and common in many bio-informatics application. In table 1 we only show the first two characters of the representation. The third column *standard_relation* and *standard_value* are describing a

CHEMBL606647	CO	=	25000.0	nM	IC50	4.60	Beta-lactamase-1	False
CHEMBL277857	CC	=	96200.0	nM	IC50	4.02	Beta-lactamase-1	False
CHEMBL84953	CC	=	100000.0	nM	IC50	4.00	Beta-lactamase-2	False
CHEMBL177772	C[	=	1600.0	nM	IC50	5.80	Beta-lactamase-1	False
CHEMBL175189	C[	=	120.0	nM	IC50	6.92	Beta-lactamase-2	True
CHEMBL1239	CC	>	90000.0	nM	IC50	4.05	Beta-lactamase-1	False
CHEMBL91	CO	=	40000.0	nM	IC50	4.40	Beta-lactamase-3	False
CHEMBL808	C[	=	25000.0	nM	IC50	4.60	Beta-lactamase-1	False

Table 1: Snapshot of the Dataset

the bio-activity value. While the column *standard_value* describes the numerical value the column *standard_relation* refers to a feature of the machine. If the measured value is bigger than the measurable range of the machine, the maximum value possible is inserted and the value of the column is set to '>' to indicate that a overflow over the measuring range occurred. This can be seen in row Nr. six. This means that the value displayed can be far higher than the displayed value. The two following columns *standard_units* and *standard_type* are describing the unit and the measuring procedure and are not relevant for the evaluation. The following column *pchembl_value* is a derived value and a logarithmic representation of the column *standard_value* calculated by formula 1. It is used to generate the binary labels.

$$\text{pchembl\_value} = -log(\text{standard\_value}) \tag{1}$$

If the value is bigger than 6 we consider the element as bio-active (True) otherwise as bio-inactive (False). The column *target_pref_name* is determining the target protein the molecule wants to bind to. As there are different kinds of beta-lactamase (here visualized as beta-lactamase-1 till beta-lactamase-4) we will only use the molecules that bind to a specific form of beta lactamase. The last column *binary* is the output term and determines if we consider the molecule as active or inactive.
In the following chapter we will discuss how we filtered and pre-processed the data to create viable datasets.

1.1 Descriptors

The descriptors used for classification were calculated using the Mordred [**?**] library. It can calculate upto 1600 2D and 3D descriptors. These are describing certain properties of the molecule for example the number of acids, the number of carbon atoms or the size of the entire Molecule. As many 3D descriptors return NaN values, we decided to focus on 2D descriptors. There are 198 descriptors with int datatype, 467 with float datatype and lastly there are 2 descriptors who are boolean. In our model we only work with numerical features, as it is not clear which integer features are categorical and which are not, due to a lack of information what the features actually mean. Therefore we eliminate the boolean features and the integer features in order to preventing bad data from lowering the estimation quality. With further background knowledge on descriptors they could be included into a future model by transforming them into a single feature space.

But also the remaining data is not optimal, we encounter big value differences in certain descriptors. Further we are not aware which descriptors actually contain information if a molecule is biologically active. We can assume that some of the descriptors do not contain any information and are only adding noise.

Summing up we can say that the data is non optimal, due to the dis balance of the dataset and the lack of information on the descriptors. Additional filtering is necessary and will described in the next chapter.

1.2 Pre-processing and creation of the dataset.

We want to split the pre-processing into three parts. First the filtering of molecules necessary before the calculation of the descriptors and the calculation after the calculation of the descriptors. Lastly the we have to choose the used descriptors.
First we eliminate molecules that do not target the beta-lactamase AmpC but another form of beta-lactamase.
In the next step we eliminate molecules where the measured value is above the measuring range of the laboratory equipment. This is the case when there is a '>' in the entry of the column *standard_relation*.
Lastly we combine measurements of the same molecule. We calculate the mean and the standard deviation of each molecule group eliminating molecules that have a *pchembl_value* more than two standard deviations away from the mean. We re-calculate the mean after discarding irregular values and keep one molecule with the *pchembl_value* equal to the calculated value.

In the next step we calculate descriptors and choose which descriptors are used for the fitting of the model. First we eliminate all descriptors that produce NaNs as a result. This are mainly 3D-descriptors. Then we filter descriptors that have more NaNs than a certain threshold. About 3000 NaNs in dataset of 65000 molecules was chosen as most descriptors fail to calculate values on the same 2500 molecules.

Afterwards we eliminate all molecules that have NaNs as descriptors. With this filtering about 61000 molecules with 900 descriptors remain.

I actively decided against interpolation of NaN values as we have not enough information about the individual descriptors and the diversity of the molecules. Further we face in some descriptors strongly varying magnitudes(i.e. a big range of values for instance between 10^-3 to 10^6). Therefore a logarithmic conversion could be of value but as this only concerns individual descriptors, we omitted this step.

Then, due to the class imbalance (20% active, 80 % inactive) of the original dataset, the dataset is oversampled. This occurs by splitting the original dataset into two datasets one containing the active molecules and one containing all inactive molecules. Then we assemble the new datasets out of n times all active molecules (which is about 20 % of the whole dataset) and a random sample of inactive molecules equal to the number of active molecules. I choose n to be 2. Resulting we have a dataset with 50%/50% active/inactive molecule distribution. The size of the dataset is about 45000 molecules.

In the next chapter we will describe the structure of the code and it's execution.

2 Code overview

In this section we give an overview over the individual options when executing this program. The program can be executed with the following command while `option` stands for one of the later explained options.

```
python main.py option
```

The first four options are concerning dataset creation from the raw data and filtering and need to be executed after each other. Those are : `prepare_dataset`,`calculate_fingerprints`, `filter_descriptors` and `create_train_validate_test`.
The three options `random-forest`,`multi-layer-perceptron` and `keras-deep-model` are starting the training of the three models. We will describe them more detailed in section 3.2 and 3.3.
The last option we implemented was the option `descriptor_stats` which calculate and output certain statistics as datatype, max value,min value, mean value and standard deviation.

2.1 download

When executing `python main.py download` the zib-file containing the dataset is downloaded from Github and is unziped into 137 zib files.

2.2 prepare_dataset

When executing `python main.py prepare_dataset` we first read in the previously downloaded CSV files and convert them to a pandas DataFrame. Then we remove molecules without a smiles representation, pchembl value and, molecules that have a different target than *'Beta-lactamase AmpC'*. Furthermore we eliminate molecules that returned pchmbl

values greater than the measuring range of the machine. Lastly we combine molecules that occur double in the dataset following the description of section 1.2.
Then we create an output column for binary and class descriptors based on the *'pchembl_value'* column.
Lastly we calculate the mol notation needed for the descriptor calculation from the already existent *smiles* notation.

The resulting dataset is saved as a file in `./Data/dataset/cleaned_dataset.pkl`

2.3 calculate_fingerprints

When executing `python main.py calculate_fingerprints` we use the mordred library [1, 2] to calculate 2-D descriptors from the previously calculated mol representation. The result are about 1600 descriptors that are filtered and refined when executing `filter_descriptors`.
The calculation takes on a laptop about 3 hours and takes great amounts of RAM. Therefore, it might comes to stops if not enough memory is present. If this happens the calculation has to be restarted. To not recalculate already calculated descriptors one can adapt the running variable of the for loop in `Data_functions.calculate_descriptors` to start not a 0 but at the wanted number.

2.4 filter_descriptors

When executing `python main.py filter_descriptors` one can filter descriptors based on the number of Nan entries and the datatype. Then all molecules that have a NaN entry in one of their descriptors are filtered and one complete dataset is created.
Which descriptors are filtered can easily be adapted by adding functions to the `FilterClass`.

2.5 create_train_validate_test

When executing `python main.py create_train_validate_test` three steps are done. First, due to the class imbalance of the original dataset, the dataset is oversampled. This occurs by splitting the original dataset into two datasets one containing the active molecules and one containing all inactive molecules. Then we assemble the new datasets out of n times all active molecules (which is about 20 % of the whole dataset) and a random sample of inactive molecules equal to the number of active molecules. I choose n to be 2. Resulting we have a dataset with 50%/50% active/inactive molecule distribution. The size of the dataset is about 45000 molecules.

Secondly the dataset is split up in three parts,train with 65% of datasets , validate with 15% of the datasets and test with the remaining 20 %. This is inline with rule of thumbs and was adapted experimentally during development.

Lastly the dataset is normalized using scikit-learns' StandardScaler.

2.6 random-forest

When executing `python main.py random-forest`, the model that is trained is the scikit-learns Random forest classifier [3]. Instead of running it on the standard parameters I doubled the number of trees in the estimated random forest.

2.7 multi-layer-perceptron

When executing `python main.py multi-layer-perceptron` the trained model is scikit-learns class `MLPClassifier` that implements a multi-layer-perceptron using backpropagation [4].The algorithm used is 'adam'. The chosen architecture will be described further in chapter 3.

2.8 keras-deep-model

When executing `python main.py keras-deep-model` we train a defined keras Sequential-model running the adam algorithm. The architecture will be described further in chapter 3.

2.9 descriptor_stats

When executing the command `python main.py descriptor_stats` analysis of the individual descriptors are printed. This includes the max and min value, mean and standard deviation, the number of unique values the number of NaNs, the most common value and the frequency of the most common value. In listing 1 one can see an example output of a descriptor. I decided against `Dataframe.Describe` as mordred introduces mordred objects into the dataframe and hindered the use of standard functions.

Listing 1: Statistics about the ETA_espilon_5 descriptor

```
Name: ETA_epsilon_5
max : 1.229762       min: 0.652381
mean: 0.832055 std_dev: 0.045008
uniq: 0015356 num_nan: 0002164
com : 0.800000 freq: 0000254
type: <class 'float'> | nr: 61740
```

3 Architectures

In this chapter we describe the architectures of the individual models. In section 3.1 we describe the architecture of the random forest, in section 3.2 we describe the architecture of the scikit-learn Multi-layer-perceptron and lastly in section 3.3 we describe the architecture of the defined model in keras.

3.1 Random Forest

We used the function `RandomForestClassifier` provided by scikit learn. As our Dataset is complex we increased the number of trees in the forest to 200. With bigger sizes we had problems with the memory. Therefore the parameter of `max_depth` could be used, but after experiments we were not able to see an improvement of the performance when reducing the depth of the trees for having more trees.

3.2 Multi-layer-perceptron

The multi-layer-perceptron we implemented had additionally to the input and output layer two hidden layers with the size of 400 each. We followed the advice of Sandhya Krishnans [5] article in which she cites 3 rules from literature [6]:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

As we have about 600 descriptors we fulfill all three rules.

3.3 Keras deep neural network

As described the deep neural network built with keras did not have good performance. The models are seem to be mainly guessing and the best accuracy of all architectures was 57% achieved by the *baseline simmilar to MLPC* model(See table 2 for reference). In every architecture the activation function used was either Relu, in the last layer we used sigmoid as activation function in order to receive a probability output between 0 and 1 which can be converted to the two classes easily.
I implemented three main architectures, first a baseline to compare to the scikit-learn MLPC-classifier that has 400 neurons in two layers. Second I implemented a small network with two layers with 60 and 3 neurons to see how well a network could perform without over-fitting. Lastly I implemented a network with 5 layers starting with 400 neurons and from there halfing the number of neurons. I added also a dropout-layer to prevent overfitting. For reference the architectures are listed in table 2.

Baseline simmilar to MLPC	Small	Deep
dense(400,act=relu)	dense(60,act=relu)	dense(400,act=relu)
dense(400,act=relu)	dense(30,act=relu)	dropout(0.5,act=relu)
dense(1,act=sigm)	dense(1,act=sigm)	dense(200,act=relu)
		dense(100,act=relu)
		dense(50,act=relu)
		dense(1,act=sigm)

Table 2: Snapshot of the Dataset

4 Evaluation

Summarizing it can be said that the cleaning and creation of the dataset was definitely the most time intensive and challenging work, due to a lack of experience of both biological data and to the handling of big datasets. The biggest influence on performance had the countering of the class-imbalance of the dataset. Before the oversampling described in section 1.2 all classifiers learned to give all molecules the inactive flag resulting in a 80 % accuracy rating (as seen during the presentation).

Both scikit-learn functions performed very well but due to the oversampling, it can occur that some of the molecules re-appear in training and test set which means that the performance of the classifiers on entirely unbiased data might be slightly worse. The designed keras-deep model did not perform well and I guess that I still have somewhere an error in the design.

The accuracy scores of the models are: Random forest classifier: 89.04%, MLP-classifier: 83.58 % and for the keras-deep model: 56.35% and can be seen in the figures 1,]2 and 3.

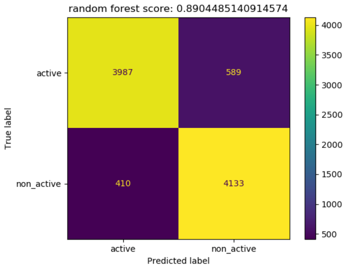

Figure 1: Random Forest

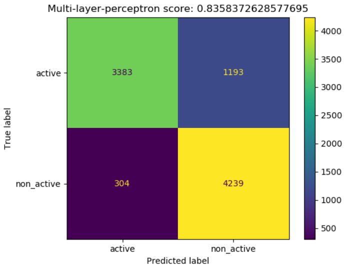

Figure 2: MLP-classifier

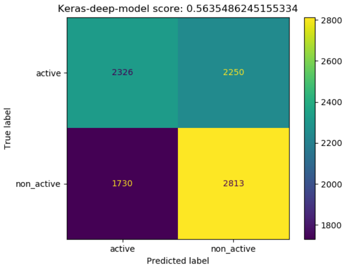

Figure 3: Keras-deep-model

The most surprising observation in line with the paper [7] that was also cited in the project proposal is that also less complex classifiers like the Random-forest-classifier can perform very well. My personal guess is that the neural-networks could outperform the RF-classifier, but it needs considerably more fine-tunig. This performance needs to be taken with a grain of salt, as we are reusing the active molecules. Therefore we can imagine that the performance on a totally unrelated dataset would be smaller.

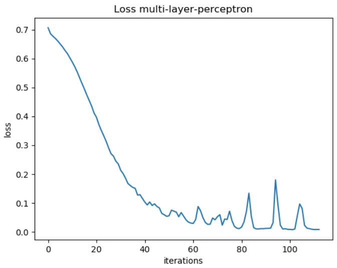

Figure 4: Loss MLP

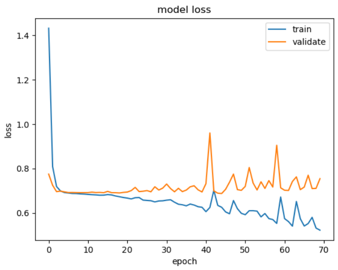

Figure 5: Loss keras-deep-model

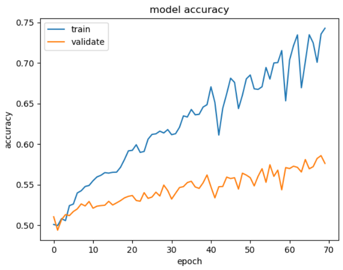

Figure 6: Accuracy keras-deep-model

The loss curve of the Multilayer-perceptron in figure 4 looks very similar to a optimal loss curve. It starts at about 0.7 and finishes at 0.06. This is also reflected in the performance. In comparison, the loss curve of the keras-deep-model 5 is starting very high and is stopping at about 0.6 which is not low enough for a good estimation. Additionally the curve decreases at the start strongly. To counter this I lowered the learning rate but, the form of the curve changed little and the performance worsened.
In figure 6 one can see the accuracy of the estimation between training and validation set. From about epoch 5 we slowly over fit which is in line with the loss curve. The accuracy on the training set in the end of the computation is bigger than 75 % while it is well below 60% in the validation set. This is also in-line with the bad performance of the classifier on the test-set.

5 Conclusion and Future-work

Summarizing it can be said that the random-forest algorithm outperforms both neural nets. This is in line with the literature [7], but still surprising. In the future the dataset needs to be cleaned even more, and more information about the individual features need to be collected to use the data efficiently. Further, the setup of the keras-deep-model needs to be restructured and possible errors should be eliminated.

References

[1] "Git repository: Mordred: a molecular descriptor calculator," https://github.com/mordred-descriptor/mordred, 2018, [Online; accessed 10-March-2022].

[2] H. Moriwaki, Y.-S. Tian, N. Kawashita, and T. Takagi, "Mordred: a molecular descriptor calculator," *Journal of cheminformatics*, vol. 10, no. 1, pp. 1–14, 2018.

[3] "Scikit-learn random forest classifier," https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html, 2021, [Online; accessed 6-March-2022].

[4] "Neural network models (supervised)," https://scikit-learn.org/stable/modules/neural_networks_supervised.html, 2021, [Online; accessed 6-March-2022].

[5] S. Krishnan, "How to determine the number of layers and neurons in the hidden layer?" https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3#, 2021, [Online; accessed 6-March-2022].

[6] J. Heaton, *Introduction to Neural Networks for Java, 2nd Edition*, 2nd ed. Heaton Research, Inc., 2008.

[7] M. C. Robinson, R. C. Glen *et al.*, "Validating the validation: reanalyzing a large-scale comparison of deep learning and machine learning models for bioactivity prediction," *Journal of computer-aided molecular design*, vol. 34, no. 7, pp. 717–730, 2020.