Programming for Data Science, ID2214

Project Report

Activity prediction for chemical compounds

Santiago Carlino, Ivan Klabucar and Nicola Toscan

December 4, 2021

1 Main objectives and scope of the assignment

The goals in the assignment were

- Identify relevant features for compound classification
- Explore performance of different models
- Choose the best model, and estimate its performance on unseen data

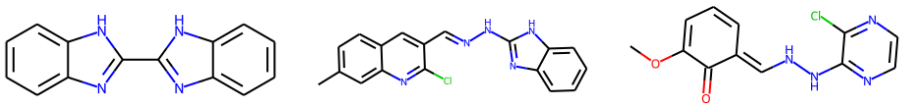
2 Methods

The dataset consists of molecules represented as "Simplified molecular input line entry specification" (SMILES) strings. To generate numerical properties of these molecules we used the library `RDKit`. This library enabled the extraction of different molecular descriptors, boolean features that represented the existence of specific functional groups in a molecule, Lipinski parameters and Morgan Fingerprints. For implementations of different machine learning algorithms and cross-validation we used mostly `scikit-learn`, and for one model `imbalanced-learn`. The metric used to compare the models was the AUC score, which we obtained by applying the `sklearn.metrics.roc_auc_score` method on the probabilities provided by the models.

Also, it is important to note that at every step in our experiments where we employed functions that introduce randomness, we called them with parameter `random_state=42` to ensure reproducibility of our results.

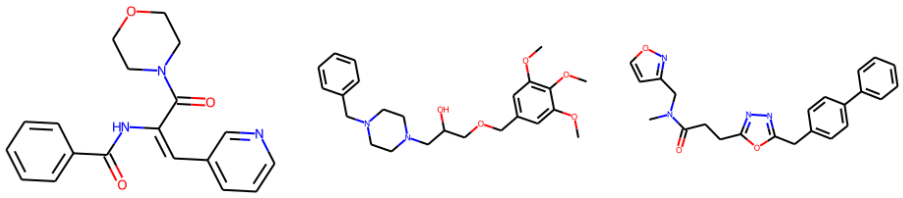
3 Data preparation and feature selection

On Figures 1 and 2 we can see the structures of some of the compounds we are working with.



(a) (b) (c)

Figure 1: Structures of three active molecules



(a) (b) (c)

Figure 2: Structures of three inactive molecules

Since our task is to develop a classifier which is able to recognize molecules which are active according to some unknown criteria, we focused on gathering as much descriptors about the structure and presence of specific functional groups in examined molecules, as these characteristics determine the behavior of a certain compound.

We have managed to gather 227 features which were obtained from 3 different sub-packages, distributed as follows (the names of the libraries contain links to their respective documentation):

- Fragments `rdkit.Chem.Fragments`: 85
- Lipinski parameters `rdkit.Chem.Lipinski`: 18
- Morgan Fingerprints `AllChem.GetMorganFingerprintAsBitVect`: 124

Because the Morgan fingerprint is strongly dependant on the structure of the molecule we suspect there exists a significant informational overlap between the fingerprint and the features provided by `rdkit.Chem.Fragments`. If this were the case, then using the aforementioned set of features should work comparably well even when the Morgan fingerprint is removed. That is why we will conduct the cross-validation of the models with and without the Morgan fingerprints to decide whether to use them or not.

With all these features imported, some preliminary studies where done in order to determine the datatype of the different features. As all of them are numerical they were well suited to be used by most machine learning algorithms.

Available samples were then randomly split into training and test sets with 70% of samples going to training. The training set will later be used in a cross-validation scheme to determine the best model and will then be used to retrain the chosen architecture on its entirety. The hold-out test set will be used to simply estimate the AUC score of the final best model and nothing else. Since the dataset is heavily skewed in favour of the negative samples, it is crucial to perform the split into training and test sets in a stratified manner. That is to ensure the same ratio of positive and negative samples in both sets. Code for this can be seen on Figure 3.

```
raw_Xtr, raw_Xtest, Ytr, Ytest = train_test_split(dfAllFeatures,
                                                  labels,
                                                  test_size=0.3,
                                                  stratify=labels,
                                                  random_state=42)
```

Figure 3: The code used for the stratified split.

3.1 Feature pre-processing

As the feature set obtained turns out to be very high-dimensional it is reasonable to assume the data are actually confined to a lower dimensional manifold. Therefore, before employing the machine learning algorithms, we will test out the option of performing dimensionality reduction by means of PCA.

Of course, before performing PCA it is instrumental to normalize every feature. Otherwise, a relatively unimportant feature with a relatively large spread could disproportionately influence the algorithm. This was avoided by z-normalization, i.e., by subtracting the mean and dividing by the standard deviation.

Afterwards, PCA was employed on the normalized dataset, with the information loss threshold set to **0.9**, meaning we would allow for only 10% loss of variance.

If the Morgan fingerprint was included in the dataset the number of remaining features drops to 145. A reasonable optimization considering the original dataset was comprised of 227 features.

In the case the Morgan fingerprint was excluded from the dataset the number of remaining features drops to **56** from 103.

It is important to note that all parameters of z-normalization (means and standard deviations of features) and PCA were learned on the training set ONLY. The test was only used for AUC estimation of the final model.

It is crucial to remark that the given dataset is heavily imbalanced. The "output" values represent the activity labels (a binary classification, with 0 corresponding to inactive and 1 corresponding to active) with respect to some unknown biological activity. From a total of 148565 different instances in the given dataset (The one that was split into train and test sets), just 442 belong to the active group. This should be taken into account when choosing the models to try out.

4 Model training

In this section we will give an overview of all the models we considered for this classification task. Each algorithm was run with the specified combinations of hyperparmeters. The default values of all unspecified hyperparameters can be looked up in their linked documentations.

- `sklearn.linear_model.LogisticRegression`, documentation link
  - `class_weight='balanced', C=0.30`
  - `class_weight='balanced', C=1.00`
  - `class_weight='balanced', C=2.00`
  - `class_weight='balanced', C=3.00`
- `sklearn.naive_bayes.GaussianNB`, documentation link
  - `var_smoothing=1e-4`
- `sklearn.neural_network.MLPClassifier`, documentation link
  - `hidden_layer_sizes=(5, 2)`
  - `hidden_layer_sizes=(10, 5)`
  - `hidden_layer_sizes=(20)`
- `sklearn.ensemble.RandomForestClassifier`, documentation link
  - `max_samples=5000, n_estimators=500, class_weight=None`
  - `max_samples=5000, n_estimators=500, class_weight='balanced'`
  - `max_samples=5000, n_estimators=500, class_weight='balanced_subsample'`
- `imblearn.ensemble.BalancedRandomForestClassifier`, documentation link

- `n_estimators=500`

As the main output related with the data set was binary (active or non-active), we considered that binary logistic regression could be a good start for the model training phase. It is known that this method tends to overfit the data when working in a high dimensional space, and due to the fact that we had a large amount of features, we used ridge regularisation (L2), which helped by penalizing high regression weights, keeping the total squared values of the regression low. It is also important to remark that this algorithm does not support imbalanced classification in a direct way, so the class-weight parameter should be assigned to "*balanced*" for taking this into consideration. We also considered 4 different values for the parameter C which represents the inverse of the regularization strength.

We were interested to see whether the activity of the molecules could be satisfactorily modeled by a simple probabilistic model such as Naive Bayes, so we also included it among the tested models. In the case that this model achieved good performance that would imply that certain features strongly influence the resulting label independently of one another. Looking into the weights of such a model would then yield significant insight into the chemistry of the problem. In the end this was not possible as the model did not perform well.

Multi layer perceptrons were also included in our experiment. We decided on a pretty standard architecture with the relu actvation function, log-loss function, and stochastic gradient descent backprop optimization algorithm. We tried out three different configurations of the hidden layer: (5,2), (10,5), and (20). We realize that increasing the number of nodes in the network while using a regularization technique might have been beneficial, but we opted not to due to computational constraints. The `sklearn` library does not provide any way to make use of a GPU while training an MLP, so doing intense back prop optimization would have been very slow.

We knew some type of ensemble method should definitely be studied due to the characteristics of the underlying problem. Specifically, we had hoped that we could combine a number of weak learners with the bagging technique to reduce the model's overall variance. This would be particularly suited to this case as the imbalance of the classes could be remedied by heavily under sampling the negative class which would in turn lead to high variance of each individual learner. The chosen method for this was Random Forests. In contrast with other methods, Random Forests do not tend to suffer from the curse of dimentionality if each tree does not consider all the features. In our case we left this up to the default behavior of the class which is to only allow each tree to look at a random subset of features of size $sqrt(\mathrm{n})$. The main problem faced for these models was the highly unbalanced dataset we had. We knew that, if used with default settings, the recall from these models would not be optimal, since the minority class would mostly not be taken into account while doing predictions. However, if in the case of low recall the model also happened to have a good AUC score that would imply that the problem could simply be remedied by changing the classification threshold. As a starting point we

decided to implement a basic random forest classifier, with the class-weight hyper-parameter set to *"none"* and with 500 individual trees. After this, we considered a model done with the same algorithm but changing the class weight to *"Balanced"*, which therefore made the model consider the inverse weighting form the training dataset, giving therefore more focus to the minority class. The third alteration of this model was considering the class-weight as *"balanced-subsample"*, which made the class weighting according to the class distribution of each of the bootstrap samples instead of doing it according to the entire training dataset. It is important to note that due to the high computational cost of training these models, we decided to set the hyper-parameter *max-samples* with a value of 5000, which represents the number of samples to draw from `Xtr` to train each base estimator. Finally, we considered a different algorithm, the *"BalancedRandomForestClassifier"*, which performs random under-sampling of the majority class in each bootstrap sample.

The models will be trained only on the train set, first in a cross-validation scheme, and then on the whole training dataset after the final model is identified.

5 Model selection

The different models will be compared by their average AUC score in a cross-validation scheme. The number of folds used was 5. To decrease the variance of the score estimates the scheme was conducted on 3 different cross-validation splits for every model. Also, because of the heavy skewness of the data toward negative samples, it is of great importance to ensure all folds contain approximately the same fraction of positive samples. For this purpose we employed the `sklearn.model_selectionRepeatedStratifiedKFold` class, which creates balanced folds with respect to the label. Code for the used cross-validation scheme can be seen on Figure 4.

```
print("Evaluation ...")
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
scores = cross_validate(model, Xtr, Ytr, scoring=scoring, cv=cv, n_jobs=-1)

aucs = scores['test_auc']
recalls = scores['test_recall']
print(f'Mean AUC: {np.mean(aucs):.3f}')
print(f'Mean REC: {np.mean(recalls):.3f}')
```

Figure 4: The code used for cross-validation of each model.

The combination of feature set and model with the highest average AUC during the aforementioned cross-validation scheme will be selected as the best and final model.

6 Results and performance estimates

With the Morgan Fingerprints:

```
Model:  LogisticRegression-C:0.3
Evaluation ...
Mean AUC: 0.870
Mean REC: 0.712
---------------------
Done in 46.31537485122681s.

Model:  LogisticRegression-C:1
Evaluation ...
Mean AUC: 0.870
Mean REC: 0.711
---------------------
Done in 45.0991325378418s.

Model:  LogisticRegression-C:2
Evaluation ...
Mean AUC: 0.870
Mean REC: 0.712
---------------------
Done in 44.594712257385254s.

Model:  LogisticRegression-C:3
Evaluation ...
Mean AUC: 0.870
Mean REC: 0.713
---------------------
Done in 43.94617986679077s.

Model:  BalancedRandomForestClassifier-500
Evaluation ...
Mean AUC: 0.892
Mean REC: 0.783
---------------------
Done in 180.02169704437256s.

Model:  GaussianNB
Evaluation ...
Mean AUC: 0.780
Mean REC: 0.073
---------------------
Done in 6.079476594924927s.

Model:  MLPClassifier-(5,2)
Evaluation ...
```

```
Mean AUC: 0.831
Mean REC: 0.043
---------------------
Done in 386.6387276649475s.

Model:  MLPClassifier-(10,5)
Evaluation ...
Mean AUC: 0.825
Mean REC: 0.090
---------------------
Done in 464.70179176330566s.

Model:  MLPClassifier-(20)
Evaluation ...
Mean AUC: 0.862
Mean REC: 0.114
---------------------
Done in 283.6934690475464s.

Model:  RandomForestClassifier-500
Evaluation ...
Mean AUC: 0.669
Mean REC: 0.000
---------------------
Done in 414.7898042201996s.

Model:  RandomForestClassifier-500-balanced
Evaluation ...
Mean AUC: 0.819
Mean REC: 0.000
---------------------
Done in 225.21877932548523s.

Model:  RandomForestClassifier-500-balanced_subsample
Evaluation ...
Mean AUC: 0.814
Mean REC: 0.000
---------------------
Done in 262.42063188552856s.

BEST MODEL: BalancedRandomForestClassifier-500
Estimate of AUC: 0.9118235319025821
Estimate of REC: 0.8346456692913385
TN: 36971 FP: 7472 FN: 21 TP: 106
```

Without the Morgan Fingerprints:

```
Model:  LogisticRegression-C:0.3
```

```
Evaluation ...
Mean AUC: 0.862
Mean REC: 0.784
---------------------
Done in 16.957573175430298s.

Model:  LogisticRegression-C:1
Evaluation ...
Mean AUC: 0.862
Mean REC: 0.783
---------------------
Done in 15.96344256401062s.

Model:  LogisticRegression-C:2
Evaluation ...
Mean AUC: 0.862
Mean REC: 0.784
---------------------
Done in 16.14865517616272s.

Model:  LogisticRegression-C:3
Evaluation ...
Mean AUC: 0.862
Mean REC: 0.783
---------------------
Done in 16.011752128601074s.

Model:  GaussianNB
Evaluation ...
Mean AUC: 0.661
Mean REC: 0.067
---------------------
Done in 1.3186535835266113s.

Model:  MLPClassifier-(5,2)
Evaluation ...
Mean AUC: 0.852
Mean REC: 0.001
---------------------
Done in 72.31956577301025s.

Model:  MLPClassifier-(10,5)
Evaluation ...
Mean AUC: 0.850
Mean REC: 0.003
---------------------
Done in 119.6124324798584s.

Model:  MLPClassifier-(20)
Evaluation ...
```

```
Mean AUC: 0.847
Mean REC: 0.016
---------------------
Done in 173.98492574691772s.

Model:  RandomForestClassifier-500
Evaluation ...
Mean AUC: 0.755
Mean REC: 0.000
---------------------
Done in 170.48565340042114s.

Model:  RandomForestClassifier-500-balanced
Evaluation ...
Mean AUC: 0.837
Mean REC: 0.000
---------------------
Done in 118.73747897148132s.

Model:  RandomForestClassifier-500-balanced_subsample
Evaluation ...
Mean AUC: 0.835
Mean REC: 0.000
---------------------
Done in 141.27116322517395s.

Model:  BalancedRandomForestClassifier-500
Evaluation ...
Mean AUC: 0.878
Mean REC: 0.797
---------------------
Done in 131.2870647907257s.
```

The model with the best AUC score during cross-validation is chosen as the final model, and an estimate of it's performance is made on the hold out test set after it is retrained on the entire training set.

BEST MODEL:

```
BalancedRandomForestClassifier-500
-- trained on feature set with Morgan Fingerprints--
Estimate of AUC: 0.9118235319025821
Estimate of REC: 0.8346456692913385
TN: 36971 FP: 7472 FN: 21 TP: 106
```

The model that achieved the highest average AUC score during cross-validation, `BalancedRandomForestClassifier-500` with Morgan fingerprints, was chosen as the best performing model. It was retrained on the entire training set and

scored: 0.9118 AUC on the hold-out test set. It's ROC curve can be seen on Figure 5. It is curious that the AUC estimate achieved on the test set is higher than the mean AUC during cross validation. This can be the result of a possibly worse model during cross validation because one fold is not used for training.

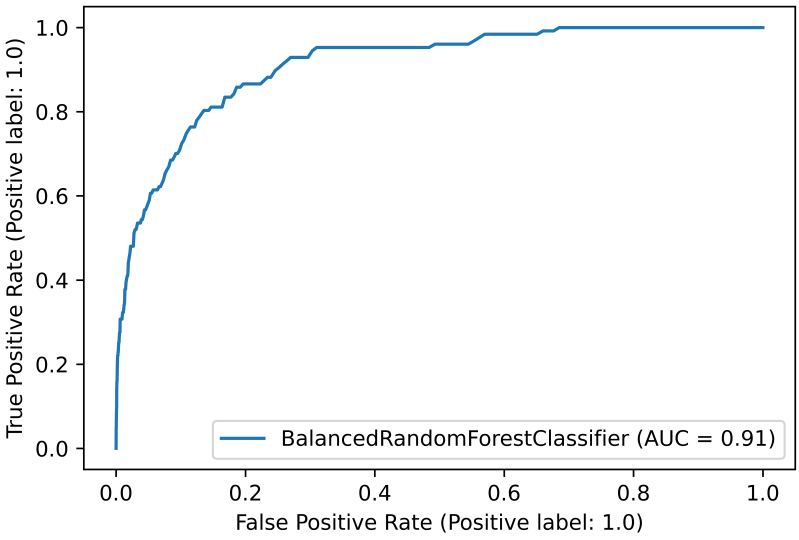


Figure 5: ROC curve of chosen model on the hold out test set.

6.1 Final Thoughts

The `imblearn` implementation of the Balanced Random Forest proved itself to be a very robust model resistant to over fitting and yet out performing other tested algorithms. Inspecting the ROC curve leads us to conclude that the model provides very good discrimination abilities, and can be adapted to be either very sensitive so as to maximise recall or very precise to minimize false positives.

For more details, please refer to our source code available through this link in the form of a python notebook named *Lab4.ipynb* intended to be run on google colaboratory.