Programming Assignment № 4a - group nr. 7

David Happel, Albin Bååw, Petrus Oskarsson, Kungliga Tekniska Högskolan 2020-12-10

Project Task: Develop a predictive model, using a training set with 121 374 chemical compounds represented by SMILES together with their activity label (0.0 = inactive, 1.0 =active) w.r.t. some biological activity.

1 External packages used in the project

- Numpy (1.19.2)
- Pandas (1.1.3)
- RDKit (2020.09.1.0)
- Scipy (1.5.2)

2 Data exploration

The original data set:

	INDEX	SMILES
0	121375	Cc1ccc(-c2csc(NC(=O)C3=NN(c4ccccc4)C(=O)CC3)n2...
1	121376	O=C(Nc1ccccc1)N1CC[C@@]2(CCCN(C(=O)c3cccc(F)c3...
2	121377	CC(=O)N1C(=O)N(C(C)=O)C2C1N(C)C(=O)N2C(C)=O
3	121378	CCOC(=O)Cn1/c(=N/C(=O)c2ccc([N+](=O)[O-])s2)sc...
4	121379	Cc1ccc(S(=O)(=O)N2CCC(C(=O)Nc3nnc(C45CC6CC(CC(...
...	...	...
40453	161828	O=C(CSc1nnc(-c2ccncc2)o1)N1CCc2ccccc2C1
40454	161829	N=c1sccn1CC(=O)Nc1ccc(Cl)c(S(=O)(=O)N2CCOCC2)c1
40455	161830	CC(/C=C/c1ccc2c(c1)OCO2)=N\NC(=O)c1cccc([N+](=...
40456	161831	CC(C)Cn1c(=O)c(C(=O)Nc2cnccn2)c(O)c2ccccc21
40457	161832	O=C(CSc1nc2c(c(C(F)(F)F)n1)CCc1ccccc1-2)NCc1cc...

40458 rows × 2 columns

Figure 1: Original data set

2.1 Balanced vs. imbalanced

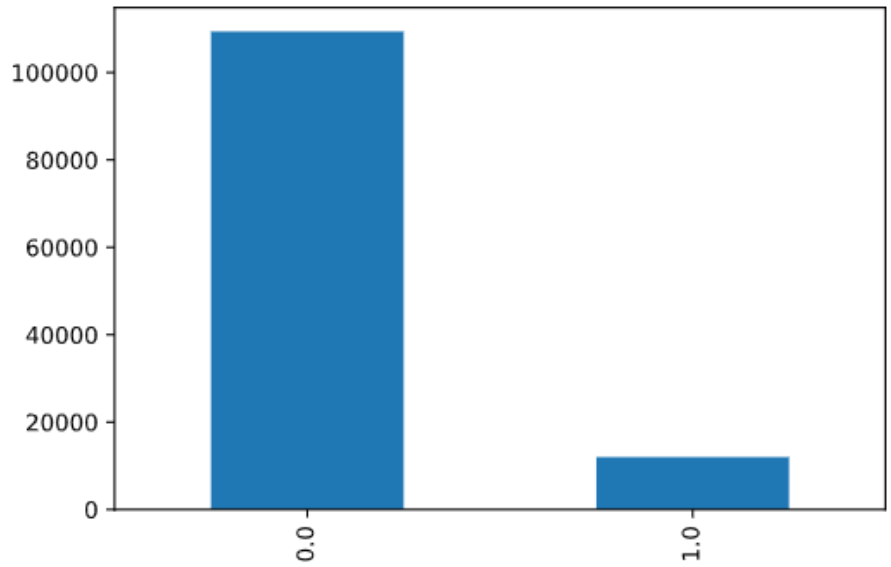

Figure 2: Fraction positive cases

An initial important observation, can be seen in Figure 2, is that the data set is imbalanced with only c. 10 percentage of the entries being part of the ACTIVE class. Later on, both undersampling and oversampling will be applied in an effort to handle the imbalance in the data set.

2.2 Motivation for metrics

The problem is a binary classification problem and for evaluation of models, multiple metrics are applied:

- **Accuracy** - Shows the fraction that the model predicts correct but does not take into account whether falsely classified are positive or negative.

  Accuracy = Correctly Classified Instances/Total Number of Instances

- **AUC** - The metric measures performance of classification over different thresholds where the ROC is the probability curve and AUC measures the separability between the classes.

  AUC = Area under the ROC curve

- **Precision** - Shows how precise/accurate the model is on instances that have been classified as positive.

  Precisions = TP/(TP+FP)

- **Recall** - Gives the fraction of how many actual positives were caught by the model.

  Recall = TP/(TP+FN)

- **F1 score** - A metric that seeks to balance between precision and recall and can be useful if there is an uneven distribution between positives and negatives in the data set.

F1 = 2*(Precision*Recall)/(Precision+Recall)

The models seeks to identify positive/active instances and one can therefore argue that it is more important that positive instances are in fact identified. However, whether this is more important is not specified and therefore the authors have put emphasis on the **F1 score** as it balances between precision and recall and will be especially suitable as the data set is imbalanced. A valuable insight during modelling is that only looking at the accuracy metrics would not make sense as a model could learn to simply classify everything as negative and achieve an accuracy of c. 90 percentage.

However, most emphasis have for evaluation have been made to the **AUC** metrics. This is a good metrics to show how well models can distinguish between positive and negative classes. On top of this, an informal competition among all the course projects is present and the AUC will be they key metric for this.

2.3 Different features

Features proposed in the assignment have been imported from the rdkit library and used in the predictions. These are:

- nrAtoms
- ExactMolWT - rdMolDescriptors
- Fragments
- Lipinski
- Fingerprint (124 bit vector)

Fingerprint consists of a 124 bits vector and every bit has been represented as a column in the prediction data set.

And of course the target variable, whether a chemical compound is active or not:

- Active

	INDEX	SMILES	nrAtoms	ExactMolWT	Fragments	Lipinski	fp_0	fp_1	fp_2	fp_3	...
0	121375.0	0.0	28.0	390.115047	0.0	28.0	0.0	0.0	0.0	1.0	...
1	121376.0	0.0	28.0	381.185255	0.0	28.0	0.0	0.0	1.0	1.0	...
2	121377.0	0.0	20.0	282.096420	0.0	20.0	0.0	0.0	1.0	1.0	...
3	121378.0	0.0	26.0	391.029663	0.0	26.0	0.0	0.0	0.0	0.0	...
4	121379.0	0.0	34.0	500.191583	0.0	34.0	0.0	0.0	0.0	0.0	...
...	...	...	...	...	...	...	...	...	...	...	...
40453	161828.0	0.0	25.0	352.099397	0.0	25.0	0.0	0.0	0.0	1.0	...
40454	161829.0	0.0	26.0	416.037975	0.0	26.0	0.0	0.0	1.0	0.0	...
40455	161830.0	0.0	26.0	353.101171	0.0	26.0	0.0	1.0	1.0	1.0	...
40456	161831.0	0.0	25.0	338.137890	0.0	25.0	1.0	1.0	1.0	0.0	...
40457	161832.0	0.0	31.0	447.102846	0.0	31.0	0.0	0.0	0.0	1.0	...

40458 rows × 130 columns

Figure 3: Features

2.4 Missing values and outliers

Listing 1: Looking for missing values

```
#Find columns with null values
null_columns = training.columns[training.isnull().any()]
training[null_columns].isnull().sum()
```

```
Series([], dtype: float64)
```

Figure 4: Missing values

As seen in figure 4, no missing values were observed in the data set.

Listing 2: Looking for outliers

```
# import to detect outliers
from scipy import stats

# Get Z-value
z_df = X_train.copy().drop(columns = ['ID'], axis = 1)
z_training = np.abs(stats.zscore(z_df))

# Nr. of outliers over 3,4,5 stds
traing_z_3 = np.where(z_training > 3)[0]
```

```
traing_z_4 = np.where(z_training > 4)[0]
traing_z_5 = np.where(z_training > 5)[0]
```

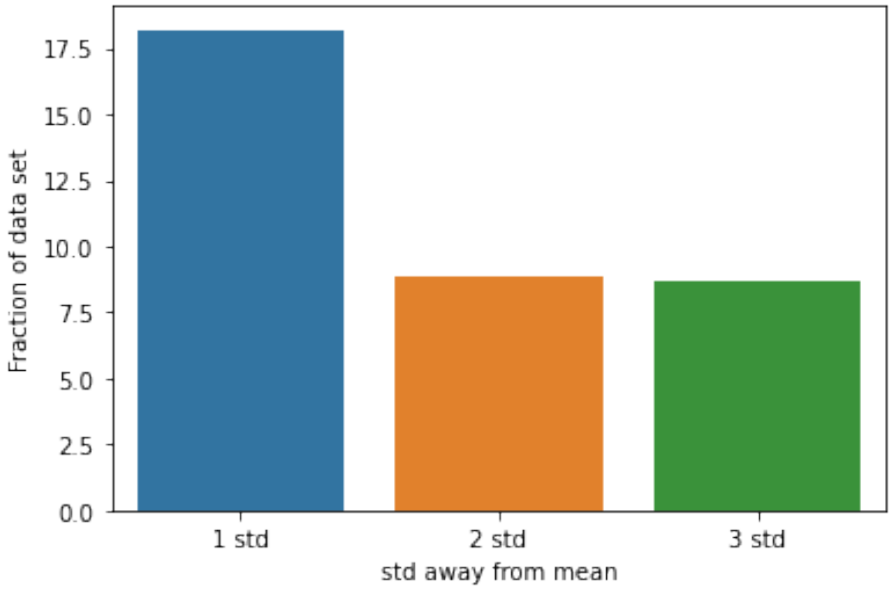

Figure 5: Output outliers

As observed in figure 5, c. 9 percentage of outliers > 5std (this is for the undersampled training set). Let's take a note of these outliers. However, we will not remove these outliers for now as they probably serve legitimate observations of the chemical compounds and are interesting for the predictions.

2.5 Feature dependency - hierarchical clustering

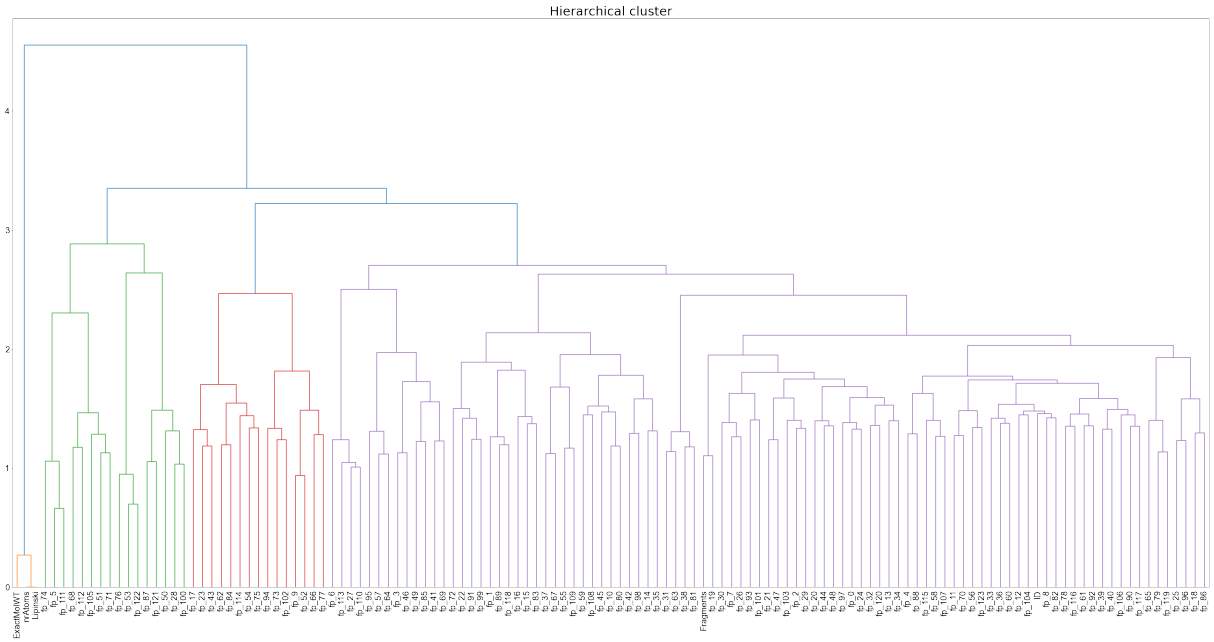

Figure 6: Hierarchical cluster

At a close look at Figure 6, it can be seen that the features 'Lipinksi' and 'nrAtoms' upshow a high collinearity. Later on, excluding the 'Lipinski' feature will be evaluated, to see whether it has an impact on the prediction performance.

3 Alternative representations - feature selection and data prep.

3.1 Different techniques to handle the imbalanced data set

Due to the imbalanced data set under- and oversampling was performed to improve the signal-to-noise ratio. Both of these alternative data sets will be used in the model and be evaluated. Later on, a conclusion was made that under-sampling will be most effective to fit our models on, since it yields the most stable results. It seems that enough data is available, for under-sampling not to cause a problem with over-fitting.

3.1.1 Undersampling

Listing 3: Undersampling

```
1 def under_sample(train_df):
2     false_sample = train_df[train_df["CLASS"] == 1]
3     true_sample = train_df[train_df["CLASS"] == 0].sample(n=false_sample.↩
          shape[0])
4     return pd.concat([false_sample, true_sample]).sample(frac=1).↩
          reset_index(drop=True)
```

3.1.2 Oversampling

Listing 4: Oversampling

```
1 def over_sample(train_df):
2     false_sample = train_df[train_df["CLASS"] == 0]
3     true_sample = train_df[train_df["CLASS"] == 1].sample(n=false_sample.↩
          shape[0], replace=True)
4     return pd.concat([false_sample, true_sample]).sample(frac=1).↩
          reset_index(drop=True)
```

3.2 Different feature sets to evaluate

In order to evaluate different set of features. Three alternatives have been evaluated:

- All features
- Only fingerprint

- Only columns ['nrAtoms', 'ExactMolWT', 'Fragments', 'Lipinski']
- Only columns ['nrAtoms', 'ExactMolWT', 'Fragments']

As earlier seen in figure 6, 'Lipinksi' and 'nrAtoms' are highly correlated and therefore removal of 'Lipinski' for a feature set will be evaluated. Removal of highly correlated features can be beneficial to avoid "Multicollinearity" (When one predictor variable in a multiple regression model can be linearly predicted from the others with a high degree of accuracy. This can otherwise lead to skewed or misleading results.)

3.3 Minmax vs. Zscore

Zscore handles outliers better which, as seen, is apparent in the data set. However the features does not belong to the same scale. Two alternative data sets will be evaluated, one normalized with 'minmax' and one with 'zscore'.

3.4 Dimensionality reduction - PCA

Listing 5: PCA explained variance analysis

```
1
2 from sklearn.decomposition import PCA
3 pca = PCA()
4 pca.fit(X_train)
5 # cum sum of explained variance for PCAs
6 np.cumsum(pca.explained_variance_, dtype=float)/(np.cumsum(pca.↩
    explained_variance_, dtype=float)[len(pca.explained_variance_)-1])
```

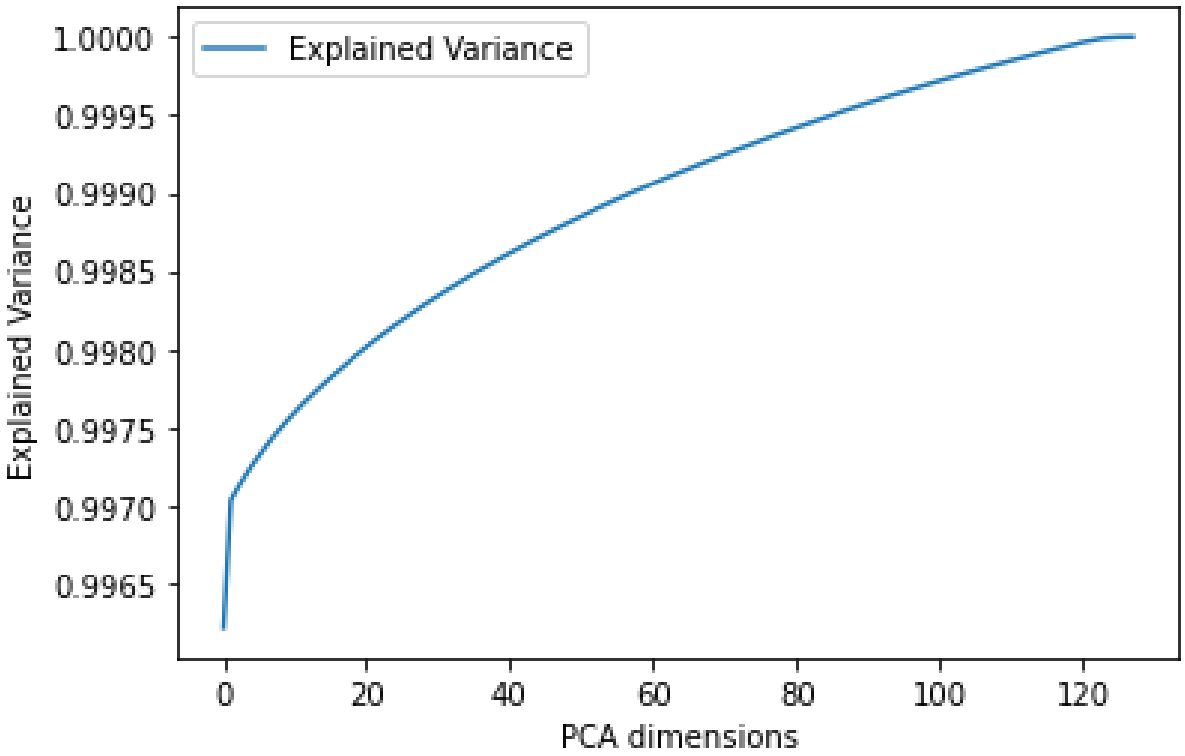

Figure 7: Output - PCA explained variance analysis

Only two PCA dimensions will be used covering more 99.7 percentage of explained variance, see figure 7.

Listing 6: PCA dimensionality reduction

```
1 # two PCA dimensions used
2 pca = PCA(n_components=2)
3 pca.fit(X_train)
4 X_train_pca = pca.transform(X_train)
5 X_test_pca = pca.transform(X_test)
```

3.5 Data sets to evaluate

- **Data set 1** - all features and minmax scaling

  To evaluate feasibility of min-max scaling and compare with z-score.

- **Data set 2** - all features and zscore scaling

  To evaluate feasibility of z-score scaling and compare with min-max.

- **Data set 3** - all features except 'Lipinski'

  As seen in figure 6, 'Linski' and 'nrAtoms' upshows high collinearity and hence it would be beenficial to evaluate the potntial positive impact of removing one of these.

- **Data set 4** - only features ['nrAtoms', 'ExactMolWT', 'Fragments']

  To evaluate the impact of features without the 124 bits fingerprint features, but also the impact of removing the 'Lipinski' feature.

- **Data set 5** - only features ['nrAtoms', 'ExactMolWT', 'Fragments', 'Lipinski']

  To evaluate the impact of features without the 124 bits fingerprint features.

- **Data set 6** - only fingerprint binaries

  To evaluate the impact of only train and predict on the 124 bits fingerprint features.

- **Data set 7** - PCA dimensionality reduction

  To evaluate performance using a data set with only two PCA dimensions

4 Different learning algorithms

4.1 Models comparison

model	AUC score	f1 score
logisticregression	0.6489 (+/- 0.01)	0.6024 (+/- 0.01)
randomforestclassifier	0.7529 (+/- 0.01)	0.6761 (+/- 0.01)
gaussiannb	0.6244 (+/- 0.01)	0.6369 (+/- 0.01)
adaboostclassifier	0.5762 (+/- 0.01)	0.5780 (+/- 0.01)
mlpclassifier	0.6941 (+/- 0.01)	0.6432 (+/- 0.01)
gradientboostingclassifier	0.6829 (+/- 0.01)	0.6268 (+/- 0.01)

Table 1: 5-fold CV Scores Dataset 1

model	AUC score	f1 score
logisticregression	0.6549 (+/- 0.01)	0.6075 (+/- 0.00)
randomforestclassifier	0.7566 (+/- 0.00)	0.6810 (+/- 0.00)
gaussiannb	0.6299 (+/- 0.00)	0.6390 (+/- 0.01)
adaboostclassifier	0.5774 (+/- 0.01)	0.5802 (+/- 0.01)
mlpclassifier	0.6900 (+/- 0.00)	0.6370 (+/- 0.01)
gradientboostingclassifier	0.6860 (+/- 0.01)	0.6263 (+/- 0.01)

Table 2: 5-fold CV Scores Dataset 2

Comparing table 2 and 1, the z-score normalization processing resulted in slightly better results, but not by any significant margin. However, Z-score is known to handle outliers more efficiently and will therefore be applied onward.

Comparing table 3 and 2, the performances are very similar and including or excluding 'Lipinski' does not seem to have an impact. However, as it seems to be that there is an opportunity to decrease complexity without hurting the performance, 'Lipinski' could be beneficial to exclude.

model	AUC score	f1 score
logisticregression	0.6574 (+/- 0.00)	0.6111 (+/- 0.00)
randomforestclassifier	0.7558 (+/- 0.01)	0.6794 (+/- 0.01)
gaussiannb	0.6310 (+/- 0.01)	0.6415 (+/- 0.01)
adaboostclassifier	0.5761 (+/- 0.01)	0.5762 (+/- 0.01)
mlpclassifier	0.6877 (+/- 0.00)	0.6352 (+/- 0.01)
gradientboostingclassifier	0.6870 (+/- 0.00)	0.6294 (+/- 0.00)

Table 3: 5-fold CV Scores Dataset 3

model	AUC score	f1 score
logisticregression	0.5587 (+/- 0.01)	0.5390 (+/- 0.01)
randomforestclassifier	0.5501 (+/- 0.01)	0.5387 (+/- 0.01)
gaussiannb	0.5512 (+/- 0.01)	0.6484 (+/- 0.00)
adaboostclassifier	0.5474 (+/- 0.01)	0.5508 (+/- 0.01)
mlpclassifier	0.5685 (+/- 0.01)	0.5788 (+/- 0.01)
gradientboostingclassifier	0.5636 (+/- 0.01)	0.5781 (+/- 0.03)

Table 4: 5-fold CV Scores Dataset 4

Both table 4 and 5 up-shown worse performance and hence excluding the fingerprint columns will not be an option.

Comparing table 6 with table 3, they both perform well but table 6 performs slightly worse but not with a large margin. Even tough there would here be an opportunity to decrease the complexity here, the authors argued that including the three features [’nrAtoms’, ’ExactMolWT’, ’Fragments’] is worth it because the results were slightly higher and also that a model will be more general with these features as they are quite different from the fingerprint.

model	AUC score	f1 score
logisticregression	0.5596 (+/- 0.00)	0.5371 (+/- 0.01)
randomforestclassifier	0.5529 (+/- 0.01)	0.5426 (+/- 0.01)
gaussiannb	0.5429 (+/- 0.01)	0.6369 (+/- 0.00)
adaboostclassifier	0.5501 (+/- 0.01)	0.5498 (+/- 0.01)
mlpclassifier	0.5682 (+/- 0.00)	0.5651 (+/- 0.02)
gradientboostingclassifier	0.5662 (+/- 0.01)	0.5709 (+/- 0.02)

Table 5: 5-fold CV Scores Dataset 5

model	AUC score	f1 score
logisticregression	0.6476 (+/- 0.01)	0.6041 (+/- 0.01)
randomforestclassifier	0.7519 (+/- 0.01)	0.6735 (+/- 0.01)
gaussiannb	0.6232 (+/- 0.01)	0.6300 (+/- 0.01)
adaboostclassifier	0.5788 (+/- 0.01)	0.5836 (+/- 0.01)
mlpclassifier	0.6775 (+/- 0.01)	0.6340 (+/- 0.00)
gradientboostingclassifier	0.6802 (+/- 0.01)	0.6184 (+/- 0.01)

Table 6: 5-fold CV Scores Dataset 6

As seen in table 7, the performance actually went down and therefore PCA dimensionality reduction will not be considered. One might try to evaluate further with more PCA dimensions as the last variance might be important for the predictions, but due to time constraint that has not been evaluated here.

model	AUC score	f1 score
logisticregression	0.5551 (+/- 0.00)	0.5337 (+/- 0.01)
randomforestclassifier	0.5224 (+/- 0.01)	0.5157 (+/- 0.01)
gaussiannb	0.5570 (+/- 0.00)	0.5376 (+/- 0.01)
adaboostclassifier	0.5050 (+/- 0.01)	0.5070 (+/- 0.01)
mlpclassifier	0.5615 (+/- 0.01)	0.5295 (+/- 0.02)
gradientboostingclassifier	0.5597 (+/- 0.01)	0.5479 (+/- 0.01)

Table 7: 5-fold CV Scores Dataset 7

4.2 Model selection

Finally, based on 4.1 Models comparison, one can argue that data set 3 is the optimal to use because of higher performance and slightly less complexity since 'Lipinski' is excluded.

Further, the Random Forest Classifier consistently performs the best and if one looks specifically at table 3, the Random Forest performs best and will therefore be the selected model.

5 Parameter setting

5.1 Hyperparameter tuning

For the hyperparameter tuning we made use of sklearn's GridSearchCV, it performs a grid search in combination with cross validation in order to find what parameter's gives the best result on the validation set.

For the grid search we limited the depth to be 5, 10, 15 and None, and we limited the number of trees to be 50, 100, 200 and 300.

The grid search was performed twice, the first time with f1 scoring and the second with AUC scoring. The results as seen in tables 8 and 9, shows that more and deeper trees often gives a better result on the validation set.

As the top performing models seems to have similar scores, we are going to with the model that has 300 trees of depth None for the test, as it ranks highest on the AUC score which is going to be used in the competition.

rank	mean f1 score	no. trees	depth
1	0.6953 (+/- 0.01)	300	None
2	0.6886 (+/- 0.00)	200	None
3	0.6859 (+/- 0.01)	200	15
4	0.6854 (+/- 0.01)	300	15
5	0.6810 (+/- 0.01)	100	None

Table 8: Top 5 best performing configurations based on f1 score

rank	mean AUC score	no.trees	depth
1	0.7730 (+/- 0.01)	300	None
2	0.7700 (+/- 0.01)	200	None
3	0.7628 (+/- 0.01)	300	15
4	0.7582 (+/- 0.01)	200	15
5	0.7547 (+/- 0.01)	100	None

Table 9: Top 5 best performing configurations based on AUC score

6 Summary

We found that a random forest with 300 trees of depth None was the best performing model of those we tried.

The model received an AUC score of 0.7730 (+/- 0.01) and a f1 score of 0.6953 (+/- 0.01) when it was run on a dataset containing most of the available features excluding the *Lipinski* feature.