ID2214 – Programming for Data Science
Assignment 4a - Project report

Group 3: Francesco Gelain, Borja Javierre, Jingyi Hu

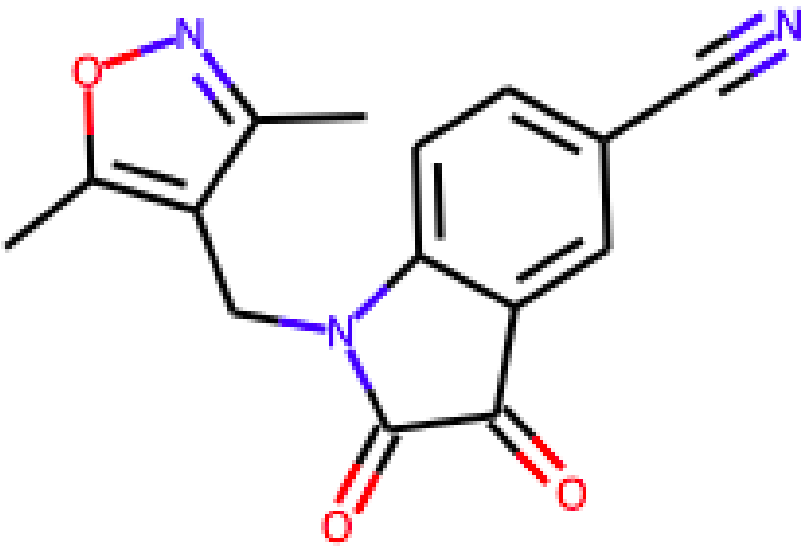


Table of Contents

Objective

The objective of this project is to predict the activity of chemical compounds from a given data set. This is going to be achieved thanks to an optimized model developed in Python.

Methodology

The flow diagram presented below in Figure 1 summarizes the project's methodology.

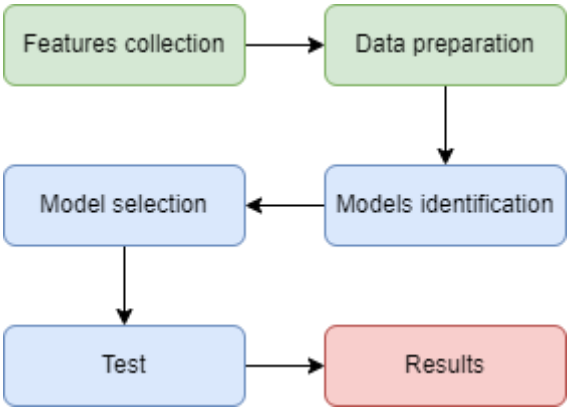


Figure 1 - Methodology

Features collection

Features extraction and selection was achieved thanks to the open-source cheminformatics toolkit ”RDkit” [1].

Library

```
m = Chem.MolFromSmiles(df_copy['SMILES'][i]) # convert SMILES to molecule. SMILES string and m will be assigned an object representing
#chemical compound, for which various properties might be derived
numatoms = m.GetNumAtoms()  # numbers of atoms
molwt = d.CalcExactMolWt(m)  # molecule's exact molecular weight
heavyatom = l.HeavyAtomCount(m)  # Number of heavy atoms a molecule
aroring = d.CalcNumAromaticRings(m)  # number of aromatic rings for a molecule, they are very stable and do not break apart easily
amidebond = d.CalcNumAmideBonds(m)  # number of amide bonds in a molecule
rotabond= d.CalcNumRotatableBonds(m)  # number of rotatable bonds for a molecule
saturatedring = d.CalcNumSaturatedRings(m)  # returns the number of saturated rings for a molecule
alcoo = f.fr_Al_COO(m)  # Number of aliphatic carboxylic acids
benzene = f.fr_benzene(m) # Number of benzene rings
```

Figure 2.1 – ”Basic” features

We selected these features because we thought they were the most representative ones for chemistry elements. Those are:

- Number of atoms (numatoms): the total number of atoms that a molecule contains
- Molecular weight (molwt): total weight of the molecule
- Heavy atoms (heavyatom): number of heavy atoms that a molecule has. These atoms are all of those which are not hydrogen atoms.
- Aromatic rings (aroaring): number of aromatic rings for a molecule. They are very stable, are not easy to break and contain benzene or another related ring structure.
- Amide bonds (amidebond): number of amide bonds in a molecule. This one is syntetized when the carboxyl of the other amino acid molecule reacts with the other amino acid molecule.
- Rotatable bonds (rotabond): number of rotatable bonds for a molecule. It is any single non-rigid bond which are attached to a non-terminal, non-hydrogen atom. Amide C-N bonds are not counted because of their high barrier to rotation.
- Saturated rings (saturatedring): number of saturated rings for a molecule. A saturated molecule contains just single bonds and no rings. Also, a saturated molecule has a maximum number of hydrogen atoms possible to be an acyclic alkane.
- Aliphatic carboxylic acids (alcoo): number of aliphatic carboxylic acids. These carbon acids perform different types of industrial functions and others are important in processes of nutrition and many are intermediates in biochemical processes.
- Benzene rings (benzene): number of Benzene rings. The Benzene is an organic chemical compound composed of 6 atoms of carbon (C) and 6 of hydrogen (H).

```
['INDEX', 'SMILES', 'NumAtoms', 'MolWt', 'HeavyAtom', 'AroRing', 'AmideBond', 'RotatableBond', 'SaturatedRing', 'AL_COO', 'Benzene', 'ACTIVE']
```

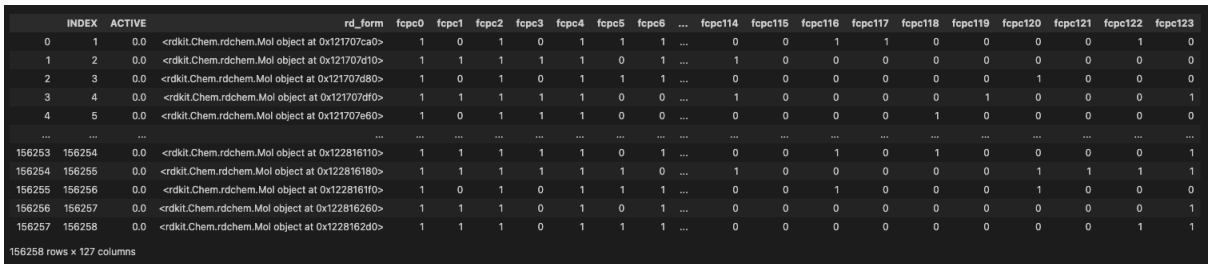
	INDEX	SMILES	NumAtoms	MolWt	HeavyAtom	AroRing	AmideBond	RotatableBond	SaturatedRing	AL_COO	Benzene	ACTIVE
0	1	CC(C)N1CC(=O)C(c2nc3ccccc3[nH]2)=C1N	19.0	256.132411	19.0	2.0	0.0	2.0	0.0	0.0	1.0	0.0
1	2	COc1ccc(-c2ccc3c(N)c(C(=O)c4ccc(OC)c(OC)c4)sc3...	30.0	420.114378	30.0	4.0	0.0	6.0	0.0	0.0	2.0	0.0
2	3	CCc1ccc(C(=O)COC(=O)CCc2nc(=O)c3ccccc3[nH]2)cc1	27.0	364.142307	27.0	3.0	0.0	7.0	0.0	0.0	2.0	0.0
3	4	O=C(CN1CCOCC1)Nc1ccc(S(=O)(=O)N2CCCCCC2)cc1	26.0	381.172227	26.0	1.0	1.0	5.0	2.0	0.0	1.0	0.0
4	5	C=CCC(Nc1ccccc1)c1ccc(OC)c(OC)c1	21.0	283.157229	21.0	2.0	0.0	7.0	0.0	0.0	2.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
156253	156254	O=C(N/N=C\c1ccc(F)cc1)c1ccc(Cn2cc(Br)c([N+](=O...	27.0	434.997844	27.0	3.0	1.0	6.0	0.0	0.0	1.0	0.0
156254	156255	COc1ccc(NS(=O)(=O)c2cc(NC(=O)/C=C/c3cc(OC)ccc3...	39.0	551.209007	39.0	3.0	1.0	10.0	1.0	0.0	3.0	0.0
156255	156256	O=c1nc(N2CCN(Cc3ccccc3)CC2)[nH]c2c1CCC2	23.0	310.179361	23.0	2.0	0.0	3.0	1.0	0.0	1.0	0.0
156256	156257	Cc1onc(-c2ccccc2)c1C(=O)N/N=C/c1ccco1	22.0	295.095691	22.0	3.0	1.0	4.0	0.0	0.0	1.0	0.0
156257	156258	CC(C)CCNC(=O)c1ccc2nc[nH]c2c1	17.0	231.137162	17.0	2.0	1.0	4.0	0.0	0.0	1.0	0.0

156258 rows × 12 columns

Figure 2.2: Result for the features collection using original library features

Fingerprint

A way to vectorizing the features that represents the molecules. By applying fingerprints to the DataFrame we can know and select which is the best model according to its performance. The fingerprints are based on the Morgan algorithm, so they are similar to the ECFP and FCFP fingerprints [10].

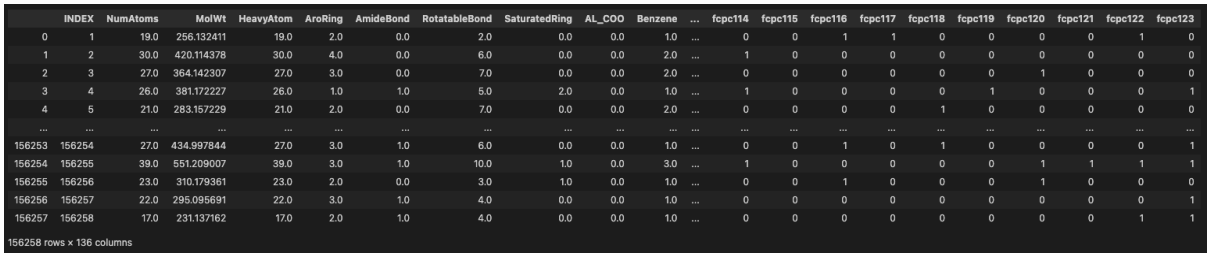


	INDEX	ACTIVE	rd_form	fcpc0	fcpc1	fcpc2	fcpc3	fcpc4	fcpc5	fcpc6	...	fcpc114	fcpc115	fcpc116	fcpc117	fcpc118	fcpc119	fcpc120	fcpc121	fcpc122	fcpc123
0	1	0.0	<rdkit.Chem.rdchem.Mol object at 0x121707ca0>	1	0	1	0	1	1	1	...	0	0	1	1	0	0	0	0	1	0
1	2	0.0	<rdkit.Chem.rdchem.Mol object at 0x121707d10>	1	1	1	1	1	0	1	...	1	0	0	0	0	0	0	0	0	0
2	3	0.0	<rdkit.Chem.rdchem.Mol object at 0x121707d80>	1	0	1	0	1	1	1	...	0	0	0	0	0	0	1	0	0	0
3	4	0.0	<rdkit.Chem.rdchem.Mol object at 0x121707df0>	1	1	1	1	1	0	0	...	1	0	0	0	0	1	0	0	0	1
4	5	0.0	<rdkit.Chem.rdchem.Mol object at 0x121707e60>	1	0	1	1	1	0	0	...	0	0	0	0	1	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
156253	156254	0.0	<rdkit.Chem.rdchem.Mol object at 0x122816110>	1	1	1	1	1	0	1	...	0	0	1	0	1	0	0	0	0	1
156254	156255	0.0	<rdkit.Chem.rdchem.Mol object at 0x122816180>	1	1	1	1	1	1	0	...	1	0	0	0	0	0	1	1	1	1
156255	156256	0.0	<rdkit.Chem.rdchem.Mol object at 0x1228161f0>	1	0	1	0	1	1	1	...	0	0	1	0	0	0	1	0	0	0
156256	156257	0.0	<rdkit.Chem.rdchem.Mol object at 0x122816260>	1	1	1	0	1	0	1	...	0	0	0	0	0	0	0	0	0	1
156257	156258	0.0	<rdkit.Chem.rdchem.Mol object at 0x1228162d0>	1	1	1	0	1	1	1	...	0	0	0	0	0	0	0	0	1	1

156258 rows × 127 columns

Library + Fingerprint

Combination of both the original “basic” library of features and the fingerprint features. We get a bigger table as we add the columns representing the fingerprints together with the columns of the features from the original library in the same DataFrame.



	INDEX	NumAtoms	MolWt	HeavyAtom	AroRing	AmideBond	RotatableBond	SaturatedRing	AL_COO	Benzene	...	fcpc114	fcpc115	fcpc116	fcpc117	fcpc118	fcpc119	fcpc120	fcpc121	fcpc122	fcpc123
0	1	19.0	256.132411	19.0	2.0	0.0	2.0	0.0	0.0	1.0	...	0	0	1	1	0	0	0	0	1	0
1	2	30.0	420.114378	30.0	4.0	0.0	6.0	0.0	0.0	2.0	...	1	0	0	0	0	0	0	0	0	0
2	3	27.0	364.142307	27.0	3.0	0.0	7.0	0.0	0.0	2.0	...	0	0	0	0	0	0	1	0	0	0
3	4	26.0	381.172227	26.0	1.0	1.0	5.0	2.0	0.0	1.0	...	1	0	0	0	0	1	0	0	0	1
4	5	21.0	283.157229	21.0	2.0	0.0	7.0	0.0	0.0	2.0	...	0	0	0	0	1	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
156253	156254	27.0	434.997844	27.0	3.0	1.0	6.0	0.0	0.0	1.0	...	0	0	1	0	1	0	0	0	0	1
156254	156255	39.0	551.209007	39.0	3.0	1.0	10.0	1.0	0.0	3.0	...	1	0	0	0	0	0	1	1	1	1
156255	156256	23.0	310.179361	23.0	2.0	0.0	3.0	1.0	0.0	1.0	...	0	0	1	0	0	0	1	0	0	0
156256	156257	22.0	295.095691	22.0	3.0	1.0	4.0	0.0	0.0	1.0	...	0	0	0	0	0	0	0	0	0	1
156257	156258	17.0	231.137162	17.0	2.0	1.0	4.0	0.0	0.0	1.0	...	0	0	0	0	0	0	0	0	1	1

156258 rows × 136 columns

Data preparation

The open-source machine learning library Scikit-learn [2] was exploited to prepare the data. Firstly, the percentage of active substances in the test set was evaluated.

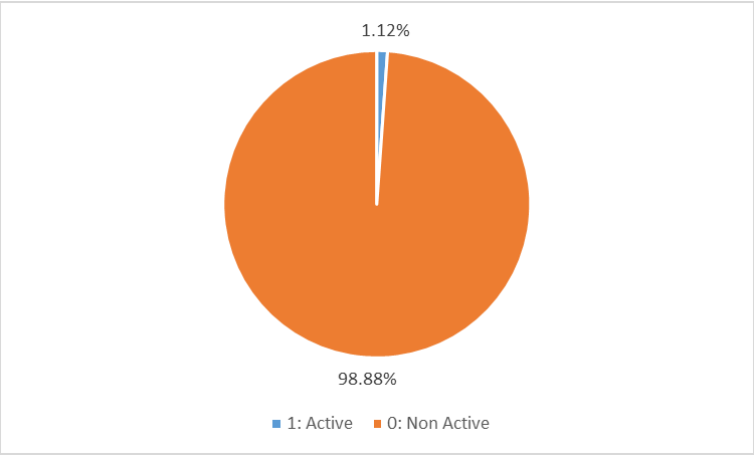


Figure 3 – Percentage of active substances in the test set

Afterwards, the columns "INDEX", "SMILES", "HeavyAtom", "ACTIVE" (stored first) were dropped. "HeavyAtom" was dropped since it was containing the same values as the column ”NumAtoms”.

The remaining data frame was normalized thanks to the functions shown in Figure 4.

```
imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')
scaler = MinMaxScaler()
kbd = KBinsDiscretizer(n_bins=10, encode="ordinal")
```

Figure 4

NumAtoms	MolWt	HeavyAtom	AroRing	AmideBond	RotatableBond	SaturatedRing	AL_COO	Benzene	ACTIVE
19.0	256.132411	19.0	2.0	0.0	2.0	0.0	0.0	1.0	0.0
30.0	420.114378	30.0	4.0	0.0	6.0	0.0	0.0	2.0	0.0
27.0	364.142307	27.0	3.0	0.0	7.0	0.0	0.0	2.0	0.0
26.0	381.172227	26.0	1.0	1.0	5.0	2.0	0.0	1.0	0.0
21.0	283.157229	21.0	2.0	0.0	7.0	0.0	0.0	2.0	0.0
...	...	...	...	...	...	...	...	...	...

Figure 5 - Before data preparation

	NumAtoms	MolWt	AroRing	AmideBond	RotatableBond	SaturatedRing	AL_COO	Benzene
0	1.0	1.0	2.0	0.0	1.0	0.0	0.0	1.0
1	8.0	8.0	3.0	0.0	5.0	0.0	0.0	2.0
2	7.0	5.0	3.0	0.0	6.0	0.0	0.0	2.0
3	6.0	6.0	1.0	1.0	4.0	1.0	0.0	1.0
4	3.0	2.0	2.0	0.0	6.0	0.0	0.0	2.0
...	...	...	...	...	...	...	...	...

Figure 6 - After data preparation

Modelling

Introduction

All the models where compared based on the AUC score. The data separation was done using stratified k-fold technique, on the training set that was given, and the optimization of the hyperparameters for each model was done thanks to GridSearchCV. This function pairs the input hyper-parameters for classifier, generates and learn the input data seperated by given cross validation method. Then based on the auc score for each paired parameters returns the model and the best parameters with the highest performance. Then, according on the comparison between the predicted output and the ACTIVE column for the validation set, roc_auc_score is used to calculate the auc score.

Function in code: grid = GridSearchCV(clf, param_grid=parameters, cv=skf, scoring="roc_auc")

auc = metrics.roc_auc_score(y_val, prediction[:, 1])

grid.best_params_

grid.best_estimator

Stratified k-fold (k = 5) cross-validation has been implemented to reduce the risk of overfitting. The model is trained using k-1 of the folds and validated on the remaining fold. Stratified is there to ensure that each fold of dataset has the same proportion of observations for a given label, this was done since the class distribution was extremely uneven, as depicted in Figure 3. A graphical explanation is presented below [3].

Function in code: skf = StratifiedKFold(n_splits=5)

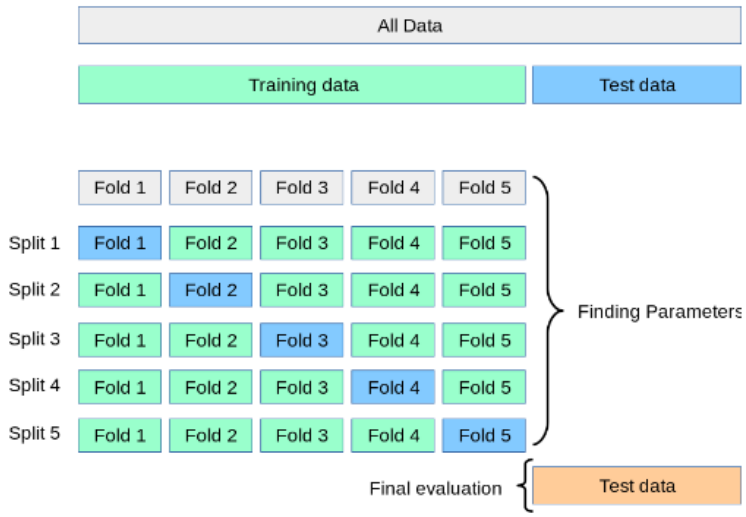


Figure 7 - k-fold cross-validation

Supervised Learning: For this project of SMILES activity prediction model training, supervised learning methods are suitable for it. Because the desired output labels for activities of compounds are already given on the training sets. Supervised learning algorithms can be trained by mapping the pairs of input compound features vectors with the supervisory active signals. Therefore, we use random forest classifier, multilayer perceptron classifier, multinomial Naïve Bayes and Bernoulli Naïve Bayes for model training.

Random Forest Classifier (RFC)

RFC offers very good predicting capabilities together with a lower chance of overfitting, robustness to outliers and runs efficiently on large datasets, as the one that we are considering [4], [5].

Function in code: model = RandomForestClassifier (**best_params_grid)

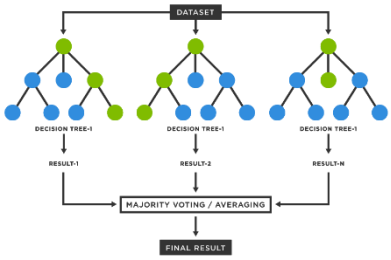


Figure 8 – RFC

Below it is the model selection with basic and fingerprint features for different parameter pairs based on GridSearchCV method.

ım_state	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	mean_test_score	std_test_score	rank_test_score
0	{'max_depth': 10, 'n_estimators': 400, 'random...	0.772856	0.759384	0.746942	0.754230	0.759567	0.758596	0.008483	1
0	{'max_depth': 10, 'n_estimators': 200, 'random...	0.773608	0.758398	0.746875	0.753095	0.760625	0.758520	0.008912	2
0	{'max_depth': 10, 'n_estimators': 300, 'random...	0.772830	0.760018	0.746288	0.753745	0.759678	0.758512	0.008727	3
0	{'max_depth': 5, 'n_estimators': 400, 'random_...	0.764216	0.767899	0.742286	0.738539	0.746499	0.751888	0.011897	4
0	{'max_depth': 5, 'n_estimators': 300, 'random_...	0.763862	0.767853	0.742509	0.738255	0.746135	0.751723	0.011875	5
0	{'max_depth': 5, 'n_estimators': 200, 'random_...	0.763775	0.767773	0.740932	0.737815	0.745577	0.751174	0.012239	6
0	{'max_depth': 15, 'n_estimators': 400, 'random...	0.688763	0.674773	0.685518	0.704456	0.689147	0.688531	0.009512	7
0	{'max_depth': 15, 'n_estimators': 300, 'random...	0.688856	0.671688	0.684814	0.706097	0.686368	0.687564	0.011011	8
0	{'max_depth': 15, 'n_estimators': 200, 'random...	0.682612	0.670549	0.681859	0.706535	0.685447	0.685401	0.011727	9

Multilayer Perceptron Classifier (MLP)

It is a feedforward supervised artificial neural network (ANN) model with backpropagation techniques. The layers can be separated into three main parts: input layer, hidden layers and output layer.

Function in code: mlp = MLPClassifier(**best_params_grid)

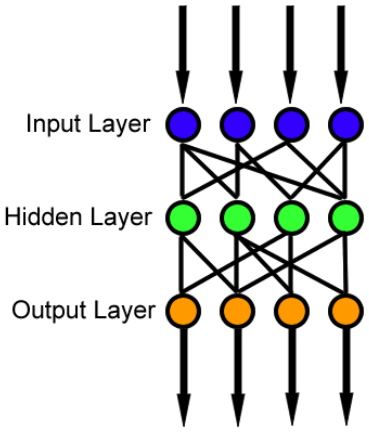


Figure 9

Below it is the model selection with basic and fingerprint features for different parameter pairs based on GridSearchCV method.

params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	mean_test_score	std_test_score	rank_test_score
{'activation': 'tanh', 'hidden_layer_sizes': (30,), 'learning_rate': 'adaptive', 'max_iter': 50, 'solver': 'adam'}	0.771860712082769	0.777151648358260	0.757890184632	0.754706900587960	0.768370720661900	0.765996031861560	0.008457312130285240	1
{'activation': 'tanh', 'hidden_layer_sizes': (30,), 'learning_rate': 'constant', 'max_iter': 20, 'solver': 'adam'}	0.771712958430730	0.775709834669288	0.755469736875082	0.752963912343720	0.769810616723074	0.765133411810847	0.009148887699624510	2
{'activation': 'tanh', 'hidden_layer_sizes': (30,), 'learning_rate': 'constant', 'max_iter': 60, 'solver': 'adam'}	0.769591478018934	0.777200930735712	0.756734339703947	0.753835593507233	0.767140771691731	0.764900622731511	0.008572549421381560	3
{'activation': 'tanh', 'hidden_layer_sizes': (20,), 'learning_rate': 'adaptive', 'max_iter': 60, 'solver': 'adam'}	0.771415300319559	0.775769403699976	0.756537584224250	0.754972398165925	0.765577431491060	0.76485442358015	0.008118237224969160	4
{'activation': 'tanh', 'hidden_layer_sizes': (25,), 'learning_rate': 'adaptive', 'max_iter': 30, 'solver': 'adam'}	0.773088095956825	0.775023995939206	0.755706329728519	0.752803435476797	0.767513814944736	0.764827134409216	0.009024368532963300	5
{'activation': 'tanh', 'hidden_layer_sizes': (30,), 'learning_rate': 'adaptive', 'max_iter': 40, 'solver': 'adam'}	0.769514795750231	0.777149403991742	0.757581398228228	0.753140661340506	0.76652439463468	0.764782130789079	0.008552863938430240	6
{'activation': 'tanh', 'hidden_layer_sizes': (30,), 'learning_rate': 'constant', 'max_iter': 50, 'solver': 'adam'}	0.769122406970381	0.777188212700903	0.756297157257378	0.751601729334629	0.76939212372803	0.764720325998266	0.009377628269487070	7
{'activation': 'relu', 'hidden_layer_sizes': (30,), 'learning_rate': 'constant', 'max_iter': 60, 'solver': 'adam'}	0.771503858988415	0.772458506476285	0.753417176977813	0.756816293297480	0.76932151764731	0.764703470483915	0.007966181356206880	8
{'activation': 'tanh', 'hidden_layer_sizes': (30,), 'learning_rate': 'adaptive', 'max_iter': 30, 'solver': 'adam'}	0.772713755564608	0.77597831612471	0.754634928810800	0.753079968166477	0.766818508706057	0.764645095474530	0.009297574923785010	9
{'activation': 'tanh', 'hidden_layer_sizes': (30,), 'learning_rate': 'constant', 'max_iter': 30, 'solver': 'adam'}	0.770050264216860	0.77452920876149	0.757853900827009	0.753233618220297	0.76711701812285	0.764556801573598	0.007865431659873910	10
{'activation': 'tanh', 'hidden_layer_sizes': (30,), 'learning_rate': 'adaptive', 'max_iter': 20, 'solver': 'adam'}	0.770862533801660	0.775311647961804	0.754105540611865	0.754444395607273	0.76796288832175	0.764537401176572	0.00870078306392276	11
{'activation': 'relu', 'hidden_layer_sizes': (30,), 'learning_rate': 'constant', 'max_iter': 30, 'solver': 'adam'}	0.772970454134839	0.776739153854182	0.752786512075213	0.753078658914649	0.76687424542673	0.764489804811240	0.009948214246918200	12
{'activation': 'tanh', 'hidden_layer_sizes': (25,), 'learning_rate': 'adaptive', 'max_iter': 60, 'solver': 'adam'}	0.770172299446878	0.775492599412951	0.756263117811271	0.753174701888035	0.76725280624101	0.764471104960030	0.008446236984224820	13
{'activation': 'tanh', 'hidden_layer_sizes': (25,), 'learning_rate': 'adaptive', 'max_iter': 50, 'solver': 'adam'}	0.770486042143827	0.775706842190509	0.754668220137213	0.752867401780395	0.76852726937762	0.764451155125914	0.009051123892324890	14
{'activation': 'tanh', 'hidden_layer_sizes': (25,), 'learning_rate': 'constant', 'max_iter': 20, 'solver': 'adam'}	0.770713751075890	0.775634929184860	0.754961202512635	0.752136371670409	0.76843272692046	0.764375954271680	0.009185145477946170	15
{'activation': 'relu', 'hidden_layer_sizes': (30,), 'learning_rate': 'adaptive', 'max_iter': 30, 'solver': 'adam'}	0.772736853760180	0.77322373940896	0.753484694780257	0.754906280948609	0.76750137705236	0.764370589190076	0.008558625028415240	16
{'activation': 'tanh', 'hidden_layer_sizes': (25,), 'learning_rate': 'adaptive', 'max_iter': 40, 'solver': 'adam'}	0.769905781805654	0.776527248950669	0.757059584741201	0.751846839980436	0.76648146987832	0.764364185071257	0.008867620045537820	17
{'activation': 'tanh', 'hidden_layer_sizes': (30,), 'learning_rate': 'constant', 'max_iter': 40, 'solver': 'adam'}	0.767915876932814	0.776169834766544	0.755655270559358	0.753586461587954	0.76848022982980	0.764361534735295	0.008496283712286840	18
{'activation': 'tanh', 'hidden_layer_sizes': (25,), 'learning_rate': 'adaptive', 'max_iter': 20, 'solver': 'adam'}	0.771611027134675	0.775825512677075	0.753301405455064	0.752840281563957	0.76812448740453	0.764340542847060	0.009520517571285840	19
{'activation': 'tanh', 'hidden_layer_sizes': (30,), 'learning_rate': 'adaptive', 'max_iter': 60, 'solver': 'adam'}	0.771754666116051	0.773922389688816	0.754874794687901	0.752128422641453	0.76900458518693	0.764336971664232	0.009025192607993520	20

Multinomial Naïve Bayes

It is a kind of Naive Bayes classifier which is working on discrete numeric features for classification. It would be better to feed integer type of features into this classifier. For other Naïve Bayes methods like BernoulliNB for binary features, multinomial is more suitable for this project.

Function in code: model = MultinomialNB(**best_params_grid)

Below it is the model selection with basic and fingerprint features for different parameter pairs based on GridSearchCV method.

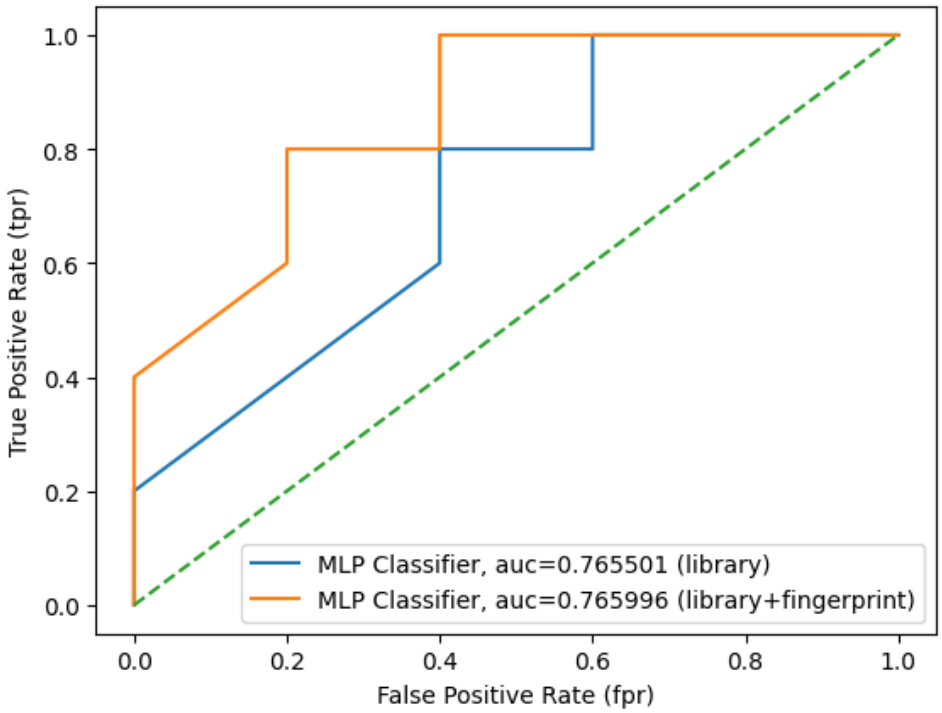
params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	mean_test_score	std_test_score	rank_test_score
{'alpha': 0, 'fit_prior': True}	0.752244	0.758346	0.737104	0.743091	0.756964	0.749550	0.008200	1
{'alpha': 0, 'fit_prior': False}	0.752244	0.758346	0.737104	0.743091	0.756964	0.749550	0.008200	1
{'alpha': 1, 'fit_prior': True}	0.747372	0.754115	0.733785	0.739245	0.753913	0.745686	0.008058	3
{'alpha': 1, 'fit_prior': False}	0.747372	0.754115	0.733785	0.739245	0.753913	0.745686	0.008058	3

Comparison and results

Based on the auc scores for different models with different input features, we found out that the fingerprint features help to improve the accuracies of the trained models but not a lot. The Naïve Bayes approaches auc scores don't change based on the input features, which is probably due to the priorities ranking of fingerprint is very low. The highest auc score in this project is 0.766, which is produced from MLP with parameters: activation = 'tanh', hidden_layer_sizes = (30,), learning_rate = 'adaptive', max_iter = 50).

	Basic Information	Basic Information + Fingerprint
Random Forest Classifier	0.758384 {'max_depth': 10, 'n_estimators': 400, 'random_state': 0}	0.758596 {'max_depth': 10, 'n_estimators': 400, 'random_state': 0}
Multilayer Perceptron Classifier	0.765501 {activation='tanh', hidden_layer_sizes=(25,), learning_rate='adaptive', max_iter=40}	**0.765996 (activation='tanh', hidden_layer_sizes=(30,), learning_rate='adaptive', max_iter=50)**
Multinomial Naïve Bayes	0.749550 {alpha=0, fitprior=True}	0.749550 {alpha=0, fitprior=True}
BernoulliNB Naïve Bayes	0.660803 {alpha=2, fitprior=False, binarize = 2}	0.660803 {alpha=2, fitprior=False, Binarize = 2}

Results



The plot shows the comparison between both Multilayer Perceptron classifiers when using just the original library and the combination of both library and fingerprints. As this is the best model classifier we tested and seen performs with the best results, we decided to plot just this one.

Future work

Different paths can be taken toward future work. One would be to increase the number of features, extract and use more such that can provide even more useful information about chemical components ana molecules.

When it comes to the results obtained from the current models used, we know that the use and combination of other models will give different results. A good approach would be to combine the outputs of the models as we did with combining the original features and the fingerprints features, such that we could see if there is an improvement on that.

Finally, a better representation of the data could be achieved, looking for new ways that the results could be represented more graphically and earlier to understand.

References

[1]. "RDKit." https://www.rdkit.org/ (accessed Dec. 14, 2022).

[2]. "scikit-learn: machine learning in Python — scikit-learn 1.2.0 documentation." https://scikit-learn.org/stable/ (accessed Dec. 14, 2022).

[3]. "3.1. Cross-validation: evaluating estimator performance — scikit-learn 1.2.0 documentation." https://scikit-learn.org/stable/modules/cross_validation.html (accessed Dec. 14, 2022).

[4] "Why Choose Random Forest and Not Decision Trees – Towards AI." https://towardsai.net/p/machine-learning/why-choose-random-forest-and-not-decision-trees (accessed Dec. 14, 2022).

[5]. "sklearn.ensemble.RandomForestClassifier — scikit-learn 1.2.0 documentation." https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (accessed Dec. 14, 2022).

[6]. "Scikit-learn: sklearn.naive_bayes.MultinomialNB" https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB (accessed Dec 15, 2022).

[7] "Angelo.edu: Aromatic Rings" https://www.angelo.edu/faculty/kboudrea/molecule_gallery/04_aromatics/00_aromatics.htm#:~:text=Aromatic%20rings%20(also%20known%20as,picture%20has%20some%20complications%2C%20however (accessed Dec 15, 2022).

[8] "cham.libretexts.com: calculating degrees of unsaturation (DoU)" https://chem.libretexts.org/Bookshelves/Organic_Chemistry/Supplemental_Modules_(Organic_Chemistry)/Alkenes/Properties_of_Alkenes/Degree_of_Unsaturation#:~:text=As%20stated%20before%2C%20a%20saturated,to%20be%20an%20acyclic%20alkane (accessed Dec. 15 2022).

[9] "Wiley Online Library: aliphatic carboxylic acids" https://onlinelibrary.wiley.com/doi/abs/10.1002/0471435139.tox070#:~:text=Aliphatic%20carboxylic%20acids%20include%20a,intermediates%20in%20normal%20biochemical%20processes. (accessed Dec. 15, 2022)

[10] "sp8rks: RDKit Tutorial" https://github.com/sp8rks/MaterialsInformatics/blob/main/worked_examples/RDKit_tutorial/RDkit_tutorial.ipynb (accessed Dec. 14, 2022)

Appendix:

Used libraries:

```
# basic
import numpy as np
import pandas as pd
import time
import itertools
# sklearn
import sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold, cross_val_score
from sklearn.neural_network import MLPClassifier
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
# rdkit
from rdkit import Chem
import rdkit.Chem.rdMolDescriptors as d
import rdkit.Chem.Fragments as f
import rdkit.Chem.Lipinski as l
from rdkit.Chem import AllChem
```