
Group: 33 Members: António Saraiva (52795) - 13h; Matilde Carvalho (54838) - 13h; Rita Rodrigues (54859) - 13h

---

Introduction

This report aims to describe the work performed for the second home assignment of the course Machine Learning. The dataset given is an augmented and edited version of the dataset QSAR (Quantitative Structure Activity Relationships) biodegradation Data Set which was built in the Milano Chemometrics and QSAR Research Group. The data have been used to develop QSAR models for the study of the relationships between chemical structure and biodegradation of molecules. It has 41 molecular descriptors and the aim of this work is to provide the best possible classification models. The variable to classify is Biodegradable which has two classes: ready biodegradable (RB) and not ready biodegradable (NRB).

Importation of data and definition of variables

We started by reading the .csv file provided using pandas. Then we observed some rows of the dataset and the types of the columns. All columns were numeric except “Biodegradable” that was defined as an object. To convert this attribute to numeric, we replaced “NRB” with 0 and “RB” with 1. We also counted the number of 0 and 1 in this attribute, obtaining 739 instances of 0 and 3825 instances of 1. We defined **X** as all the columns except “Biodegradable”. This column was the one used to define **y**.

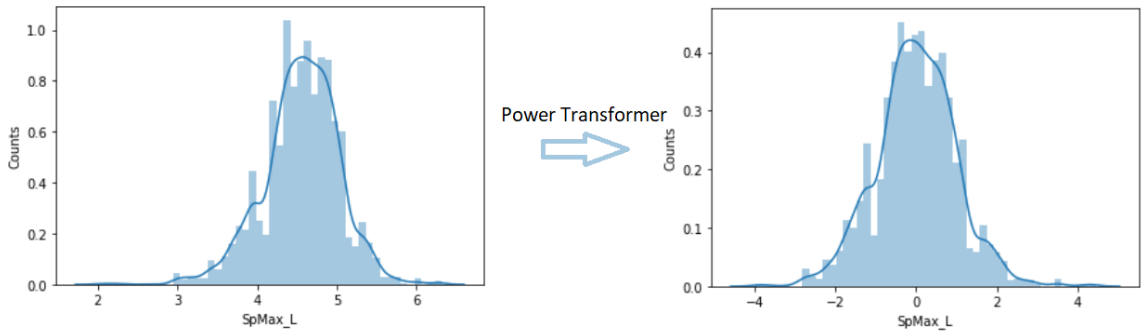
Data processing

We observed that the data wasn’t normalized and it had missing values. Because of that, we coded a Data Transformation and Classification Pipeline using KNN in order to find the best way to handle this (measured by f1). First we splitted the dataset into a training and testing set, using simple cross validation (with a test size of 0.25); then we defined a dictionary D, which would save all the scalers, imputation methods and its parameters, and also classification metrics, namely precision, recall, f1 and mcc. The modeler used in this pipeline was K-nearest neighbors (KNN) algorithm with a k of 68, because this is the approximation of the square root of the number of samples in this dataset (4563). The scalers tested were MinMax Scaler, Standard Scaler, Normalizer and Power Transformer and the imputers were Simple Imputer and KNN Imputer. For the Simple Imputer we varied the strategy parameter with the values ‘mean’, ‘median’, ‘most_frequent’ and ‘constant’ and for the KNN Imputer we varied the weights parameter with the values ‘uniform’ and ‘distance’. The best results are in the following table.

index	Data Scaling	Missing values	Parametros	precision	recall	f1 ▼	mcc
18	PowerTransformer()	SimpleImputer	mean	0.9322033898305084	0.9883720930232558	0.9594663930220626	0.7384448774921967
22	PowerTransformer()	KNNImputer	uniform	0.933	0.9862579281183932	0.9588900308324769	0.7350893286790158
23	PowerTransformer()	KNNImputer	distance	0.9338677354709419	0.985200845665962	0.9588477366255145	0.7352903081694864
20	PowerTransformer()	SimpleImputer	most_frequent	0.932067932067932	0.9862579281183932	0.9583975346687211	0.7314577426386827
19	PowerTransformer()	SimpleImputer	median	0.9310689310689311	0.985200845665962	0.9573703133025167	0.7243612833773229

The best option obtained (meaning best f1) was Power Transformer in combination with Simple Imputer with strategy ‘mean’. The next step was to define the best transformer and imputation method and transform the training set. This set is now defined as **Xt_train** and **Xt_test**. To see

the results of the transformation, we plotted the distribution of the first independent variable 'SpMax_L' before and after the transformation. The result is in the following image:



It's possible to confirm that the application of Power Transformer was successful because the data is now more Gaussian-like. We also confirmed the application of Simple Imputer by observing that the sum of missing values in Xt_train was 0.

Feature selection

The dataset had 41 molecular descriptors. In this step our goal was to select the most significant ones. We tested correlation, stepwise methods and random forest. For correlation, we started by defining and observing the correlation matrix. In the first column of this matrix we can see the variables in the X matrix that are most related to y. We chose the variables that had an absolute value of the correlation bigger than 0.35 and obtained 12 variables. This value seemed to be a good balance between the value for the correlation and the number of variables selected, meaning if we wanted to only select the variables with a correlation bigger (in absolute value) than 0.5 for example, we would only select one variable, which is unacceptable. For the stepwise method, we used SequentialFeatureSelector() and the number of features to select was 12 in order to be able to compare the results with the previous method. We also tested Random Forest. The best features obtained for each method are in the following table:

index	Method	Best features
0	Correlation	0,2,4,5,10,21,24,32,33,37,38,40
1	Stepwise,LinearRegression(),forward	[2 4 5 13 14 17 19 21 29 31 33 40]
2	Stepwise,LinearRegression(),backward	[2 4 5 12 13 17 19 29 31 32 33 34]
3	Stepwise,DecisionTreeClassifier(max_depth=5),forward	[2 5 6 9 10 15 16 29 31 32 33 34]
4	Stepwise,DecisionTreeClassifier(max_depth=5),backward	[0 1 2 5 6 7 11 26 29 31 32 33]
5	Stepwise,DecisionTreeClassifier(max_depth=10),forward	[2 4 5 6 10 20 25 27 29 31 33 39]
6	Stepwise,DecisionTreeClassifier(max_depth=10),backward	[1 2 4 5 6 10 11 15 16 17 31 33]
7	Stepwise,DecisionTreeClassifier(max_depth=20),forward	[2 5 6 8 10 15 19 20 23 24 31 34]
8	Stepwise,DecisionTreeClassifier(max_depth=20),backward	[1 2 5 6 9 10 13 16 31 32 33 38]
9	Random Forest	[1 2 4 5 6 10 21 31 33]

Observing the best features among all methods, it's possible to conclude that the most common chosen features are 2, 4, 5, 6, 10, 19, 31, 33.
We adapted a function from the practical classes to test the performance. It runs Random Forest Classifier and Decision Tree Classifier and calculates the f1 score. The results obtained for the data with all variables were 0.9816 and 0.9689 for Random Forest and Decision Tree, respectively. The data that only contains the best variables indicated before is defined as **nX_train** and **nX_test**. The results obtained were worse, 0.9678 for Random Forest and 0.9614 for Decision Tree, but we didn't think that was significant considering we selected only 8 variables in a total of 41. Then we observed the correlation matrix of nX_train:

We observed that there wasn't a correlation value above 0.8 (in absolute value) so we considered the feature selection step as concluded. After processing the data and selecting the best variables, we went on to run different models in order to find the best one.

Models

We started by defining a dictionary that would store the model, its parameters and the following metrics: MCC, precision, recall, F1. The models tested were the following: decision tree, logistic regression, categorical naive bayes, k-nearest neighbors, linear support vector classification, support vector classification, bagging classifier, random forest, adaboost, gradient boosting and xgboost. The parameters and values tested were the ones that we found most important in the practical classes and doing some research. Our approach was to use GridSearchCV in order to test different combinations of hyperparameters, and save in the dictionary the best configuration for each model, meaning the combination that scored higher on f1. We observed that for support vector classification and for gradient boosting GridSearchCV took a long time, so we did some research and found RandomizedSearchCV that in contrast to GridSearchCV, doesn't try all parameter values, but rather a fixed number of parameter settings. The fixed number is by default 10 and was the one we used.

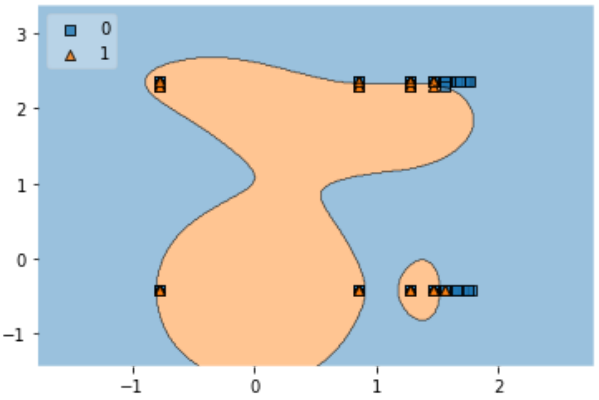
The parameters tested were:

- Decision Tree: criterion (gini, entropy); max_depth (range(1,10)); min_samples_split (range(2, 10)), min_samples_leaf (range(1, 5))
- Logistic Regression: C (np.logspace(-4, 4, 20))
- Categorical Naive Bayes: alpha ([0.0001, 0.001, 0.01, 0.1, 0.5, 1.0])
- K-Nearest Neighbors: n_neighbors (range(1, 100)); weights (uniform, distance); metric (euclidean, manhattan)
- Linear Support Vector Classification: C ([0.1,1,100,1000])
- Support Vector Classification: C ([0.1,1,100,1000]); kernel (rbf, poly, sigmoid, linear); degree ([1,2,3,4,5,6]); gamma ([1, 0.1, 0.01, 0.001, 0.0001])
- Bagging Classifier (with base estimator Decision Tree):
    - Decision Tree: max_depth ([3,5,10,20]); max_features (None, auto); min_samples_leaf ([1, 3, 5, 7, 10]); min_samples_split ([2, 5, 7])
    - Bagging Classifier: bootstrap_features (False, True); max_features ([0.5, 0.7, 1.0]); max_samples ([0.5, 0.7, 1.0]); n_estimators ([2, 5, 10, 20])
- Bagging Classifier (with base estimator GaussianNB): bootstrap_features (False, True); max_features ([0.5, 0.7, 1.0]); max_samples ([0.5, 0.7, 1.0]); n_estimators ([2, 5, 10, 20])

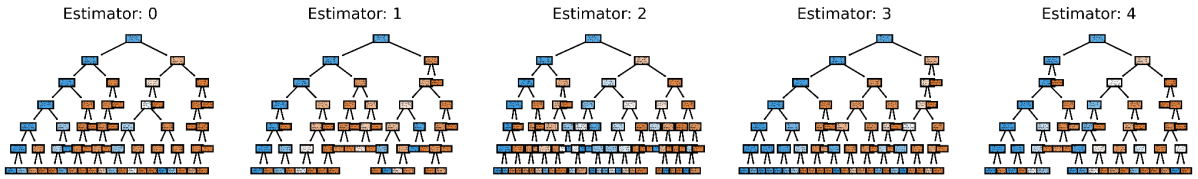
- Random Forest (with both base estimators Decision Tree and Naive Bayes): n_estimators ([50, 100, 200, 500]); max_features (auto, sqrt, log2); max_depth ([4,5,6,7,8]); criterion (gini, entropy, log_loss)
- Adaboost (with base estimator Decision Tree):
    - Decision Tree: max_depth ([i for i in range(2,11,2)]); min_samples_leaf ([5,10])
    - Adaboost: n_estimators ([10,50,250,1000]); learning_rate ([0.01,0.1,1,5])
- Adaboost (with base estimator GaussianNB): n_estimators ([10,50,250,1000]); learning_rate ([0.01,0.1,1,5])
- Gradient boosting: loss (deviance); learning_rate ([0.01, 0.025, 0.05, 0.075, 0.1, 0.15, 0.2]); min_samples_split (np.linspace(0.1, 0.5, 12)); min_samples_leaf (np.linspace(0.1, 0.5, 12)); max_depth ([3, 5, 8]); max_features (log2, sqrt); criterion (friedman_mse, mae); subsample ([0.5, 0.618, 0.8, 0.85, 0.9, 0.95, 1.0]); n_estimators ([10,50,250,1000])
- Xgboost: max_depth (range (2, 10, 1)); n_estimators (range(60, 220, 40)); learning_rate ([0.1, 0.01, 0.05])

Results

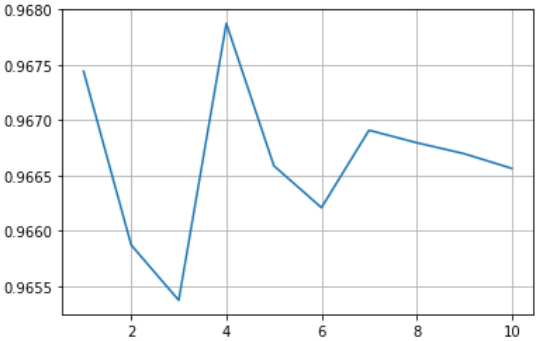
In order to observe some results, we decided to do some plots.
For Support Vector Classification we plotted the decision regions of two variables (with indices 3 and 7) using the function plot_decision_regions from mlxtend and got the following result:



For Random Forest, we plotted the first 5 decision trees:



And for Adaboost, we observed the impact on f1 score by changing the max_depth (range(1, 11)) in Decision Tree:

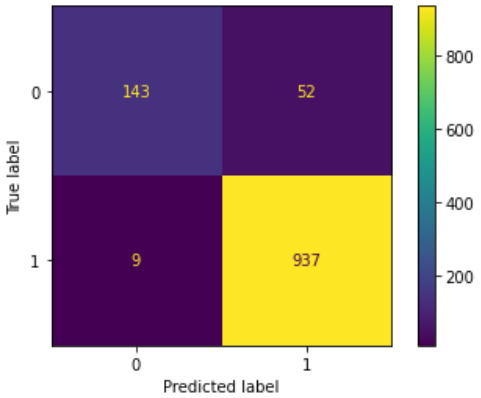


To compare models, we transformed the dictionary that had the best parameter configuration into a dataframe and got the following results:

Model	Parameters	MCC	Prc	Rec	F1 ▼
Random Forest with Decision Tree	{'criterion': 'gini', 'max_depth': 6, 'max_features': 'auto', 'n_estimators': 50}	0.8052581214543857	0.9483805668016194	0.9904862579281184	0.9689762150982419
Random Forest with Gaussian NB	{'criterion': 'entropy', 'max_depth': 6, 'max_features': 'sqrt', 'n_estimators': 200}	0.8018133223053348	0.9474216380182002	0.9904862579281184	0.9684754521963825
Decision Tree	{'criterion': 'entropy', 'max_depth': 9, 'min_samples_leaf': 1, 'min_samples_split': 4}	0.7949615514437575	0.9464105156723963	0.9894291754756871	0.9674418604651164
Adaboost with Decision Tree	{'base_estimator__max_depth': 2, 'base_estimator__min_samples_leaf': 10, 'learning_rate': 0.01, 'n_estimators': 1000}	0.7950673291219771	0.9473150962512664	0.9883720930232558	0.9674081738230729
Bagging Classifier with Decision Tree	{'base_estimator__max_depth': 20, 'base_estimator__max_features': 'auto', 'base_estimator__min_samples_leaf': 1, 'base_estimator__min_samples_split': 2, 'bootstrap_features': False, 'max_features': 1.0, 'max_samples': 0.7, 'n_estimators': 20}	0.7954077137647586	0.9491353001017294	0.9862579281183932	0.9673405909797822
XGBoost	{'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 140}	0.7849434040601347	0.9462474645030426	0.9862579281183932	0.9658385093167703
K-Nearest Neighbors	{'metric': 'euclidean', 'n_neighbors': 14, 'weights': 'distance'}	0.7753350186610302	0.9524793388429752	0.9746300211416491	0.9634273772204807
Support Vector Classification	{'kernel': 'rbf', 'gamma': 1, 'degree': 4, 'C': 1000}	0.7425449854932429	0.9427987742594485	0.9756871035940803	0.958961038961039
Logistic Regression	{'C': 0.23357214690901212}	0.7225525417514411	0.948744769874477	0.9587737843551797	0.9537329127234491
Linear Support Vector Classification	{'C': 0.1}	0.7225525417514411	0.948744769874477	0.9587737843551797	0.9537329127234491
Adaboost with GaussianNB	{'learning_rate': 0.01, 'n_estimators': 1000}	0.6996709547039556	0.9331306990881459	0.9735729386892178	0.9529229177444387
Bagging Classifier with Gaussian NB	{'bootstrap_features': False, 'max_features': 1.0, 'max_samples': 1.0, 'n_estimators': 2}	0.7387722615511156	0.9642470205850487	0.9408033826638478	0.9523809523809523
Naive Bayes	{'alpha': 0.0001}	0.627483303583556	0.931321540062435	0.9460887949260042	0.9386470896696382
Gradient boosting	{'subsample': 0.95, 'n_estimators': 1000, 'min_samples_split': 0.28181818181818186, 'min_samples_leaf': 0.1, 'max_features': 'sqrt', 'max_depth': 5, 'loss': 'deviance', 'learning_rate': 0.15, 'criterion': 'friedman_mse'}	0.4243481886578766	0.877511961722488	0.9693446088794926	0.921145153189352

Conclusion

The best model that we obtained was Random Forest with Decision Tree as base estimator, with the following parameters: {criterion:gini, max_depth:6, max_features:auto, n_estimators:50}. The f1 was 0.968976 which is pretty good. Even the others models except Naive Bayes and Gradient Boosting had a f1 above 0.95 which is high. This dataframe is in descendent order of f1 and it's possible to observe that the best 6 models are all related to Decision Trees. To our surprise, XGboost was not the best model overall but the differences in the metrics are not that far from the Random Forest.
After choosing Random Forest with Decision Tree as best model, we observed the confusion matrix:



The values in the main diagonal are much higher than the values in the anti diagonal which indicates that the model is able to predict the biodegradability of chemicals pretty well.