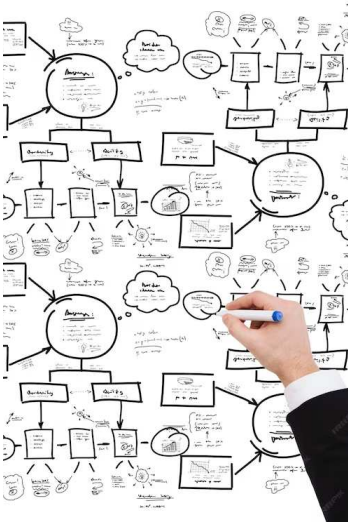


Building a Python Project: From Concept to Execution

Benchekroun Yanis
DIA A4

Introduction

Understanding the **Python project** lifecycle and best practices for successful execution.

Goal of the project :

analyze a *QSAR biodegradation* dataset containing molecular descriptors for 1055 chemicals and develop classification models to discriminate between ready and not ready biodegradable molecules

involves comprehensive data preprocessing, visualization, modeling, and the transformation of the best model into an API for practical deployment

How to do it ?

Defining project scope, setting **goals**, and creating a clear **roadmap** for development.

Data Understanding: Explore and understand the QSAR biodegradation dataset, including the 41 molecular descriptors, the target variable ('nO'), and any potential challenges in the data.

Data Pre-processing :Clean the dataset by handling missing values, encoding categorical variables, and normalizing numerical features. Explore potential feature engineering opportunities based on domain knowledge.

Data Visualization: Utilize tools like Matplotlib and Seaborn to visually explore relationships between features and the target variable ('nO'). Create histograms, scatter plots, and boxplots to gain insights into the distribution and correlation of variables.

Modeling: Split the dataset into training and testing sets. Implement classification models (e.g., RandomForest, GradientBoosting, SVM) using scikit-learn. Perform hyperparameter tuning via GridSearchCV to optimize model performance.

Evaluation: Evaluate the trained models using metrics such as accuracy, precision, recall, and F1-score. Compare model performances to select the best-performing model.

API Transformation: Choose a framework (e.g., Flask or Django) for API development.Transform the selected classification model into a functional API, defining endpoints for predictions

Presentation:Develop a PowerPoint presentation to convey the project's objectives, methodology, and results.Include visuals, code snippets, and a narrative to effectively communicate the analysis and decision-making processes.

Coding and Testing

Here are some of the codes in the project that are explained

This code is using Label Encoding from scikit-learn's LabelEncoder to convert categorical variables in the DataFrame df for columns 'nHM' and 'NssssC' into numeric representations.

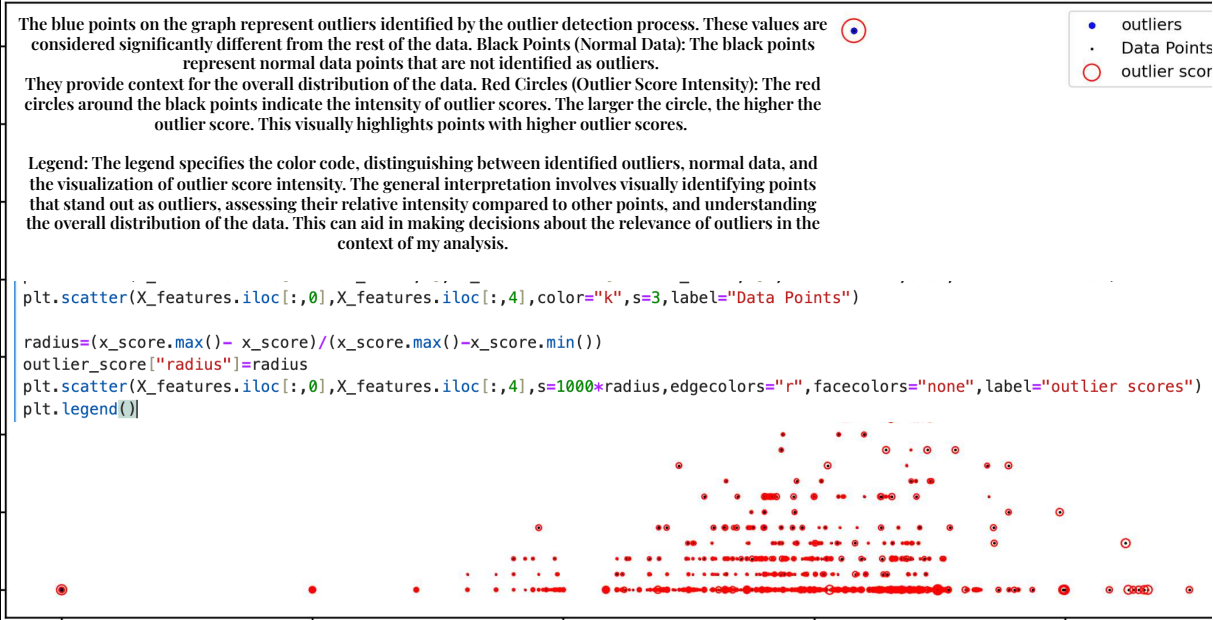
The lambda function applies the label encoder to each column specified in categorical_columns, transforming categorical values into corresponding numerical labels.

```
# Encoding categorical variables using Label Encoding
label_encoder = LabelEncoder()
categorical_columns = ['nHM', 'NssssC']
df[categorical_columns] = df[categorical_columns].apply(lambda col: label_encoder.fit_transform(col))
```

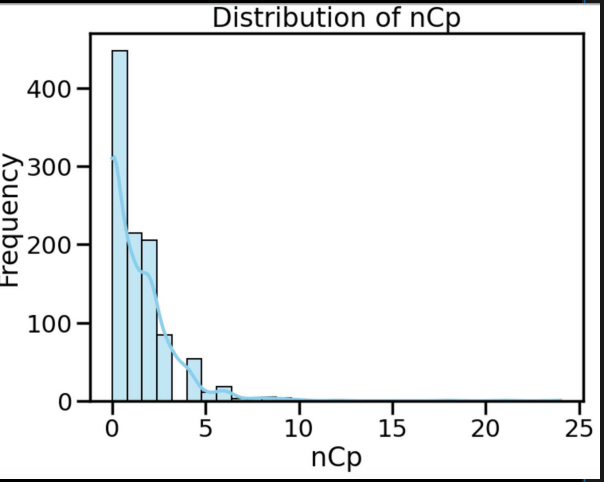
tplotlib.legend.Legend at 0x16ab44b50>

The blue points on the graph represent outliers identified by the outlier detection process. These values are considered significantly different from the rest of the data. Black Points (Normal Data): The black points represent normal data points that are not identified as outliers.
They provide context for the overall distribution of the data. Red Circles (Outlier Score Intensity): The red circles around the black points indicate the intensity of outlier scores. The larger the circle, the higher the outlier score. This visually highlights points with higher outlier scores.

Legend: The legend specifies the color code, distinguishing between identified outliers, normal data, and the visualization of outlier score intensity. The general interpretation involves visually identifying points that stand out as outliers, assessing their relative intensity compared to other points, and understanding the overall distribution of the data. This can aid in making decisions about the relevance of outliers in the context of my analysis.



SOME EXAMPLE OF DATA PROCESSING AND VISUALIZATION




```python
#let's have a quick look before diving into the data processing

# Histogram
plt.figure(figsize=(8, 6))
sns.histplot(df['nCp'], bins=30, kde=True, color='skyblue')
plt.title('Distribution of nCp')
plt.xlabel('nCp')
plt.ylabel('Frequency')
plt.show()

# Scatter plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x='C%', y='NssssC', data=df, hue='nO', palette='coolwarm')
plt.title('Scatter Plot of C% vs NssssC')
plt.xlabel('C%')
plt.ylabel('NssssC')
plt.show()

# Boxplot
plt.figure(figsize=(10, 8))
sns.boxplot(x='nO', y='nCp', data=df, palette='Set2')
plt.title('Boxplot of nCp by nO')
plt.xlabel('nO')
plt.ylabel('nCp')
plt.show()
```

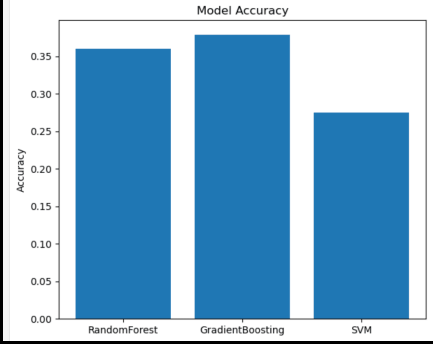
“Model Evaluation and Hyperparameter Tuning for Classification.“

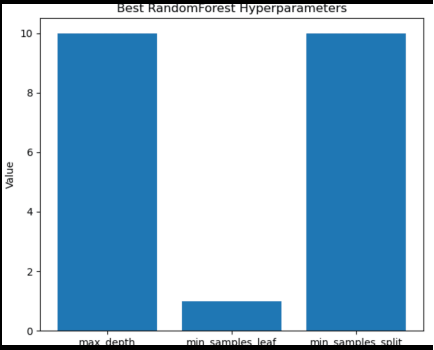
:Data Preparation:The code starts by specifying the features (‘nCp’, ‘C%’, ‘NssssC’) and the target variable (‘nO’).Train-Test

Splits the dataset into training and testing sets (80% training, 20% testing) using the scikit-learn function train_test_split.

Model Definition:Defines three classification models: RandomForest, GradientBoosting, and Support Vector Machine (SVM).

Model Training and Evaluation:Iterates through each model, trains it on the training set, predicts on the testing set, and evaluates its performance using accuracy and classification reports.Stores the results for each model in the ‘results’ dictionary.Hyperparameter Tuning for RandomForest:Defines a grid of hyperparameters for RandomForest using param_grid_rf.
Performs hyperparameter tuning using GridSearchCV to find the best combination of hyperparameters that maximizes accuracy.



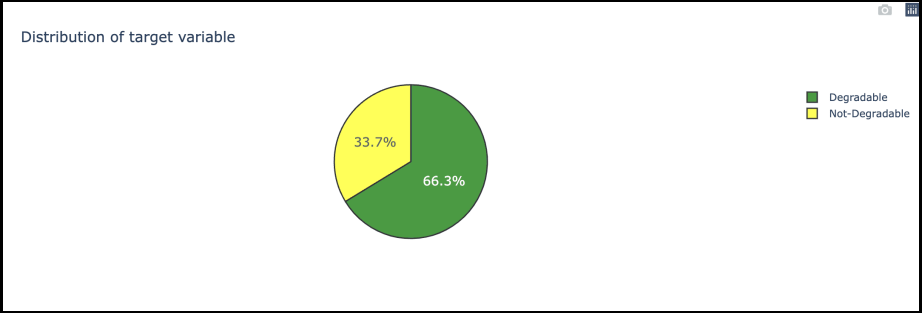


Retraining the Best Model:Retrieves the best hyperparameters found by GridSearchCV.Retrains the RandomForest model with the optimal hyperparameters on the entire training set.

Visualization of Results:Creates a bar chart to visually compare the accuracy of different models.Plots a bar chart to visualize the best hyperparameters for the RandomForest model.

Overall Goal:The primary goal of this code is to compare the performance of multiple classification models on a given dataset, identify the best hyperparameters for the RandomForest model through a grid search, and visualize the results to aid in model selection and fine-tuning.Add a body text

We observe also that there is a majority of elements that are degradable



Number of Subplots:The code creates a grid of subplots, and the number of subplots is determined by the size of the bio_df DataFrame. In this case, there are 8 rows and 4 columns, resulting in 32 subplots.

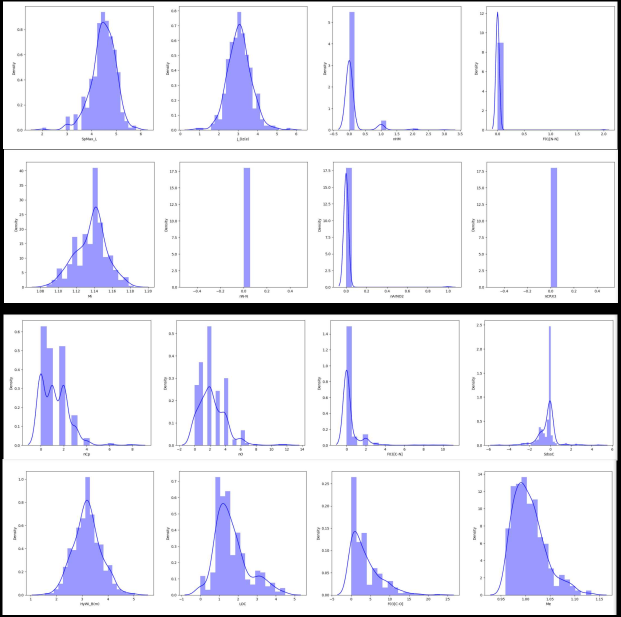
Each subplot contains a histogram, which represents the distribution of values in the corresponding column.The x-axis typically represents the range of values in the column, while the y-axis represents the frequency or density of those values

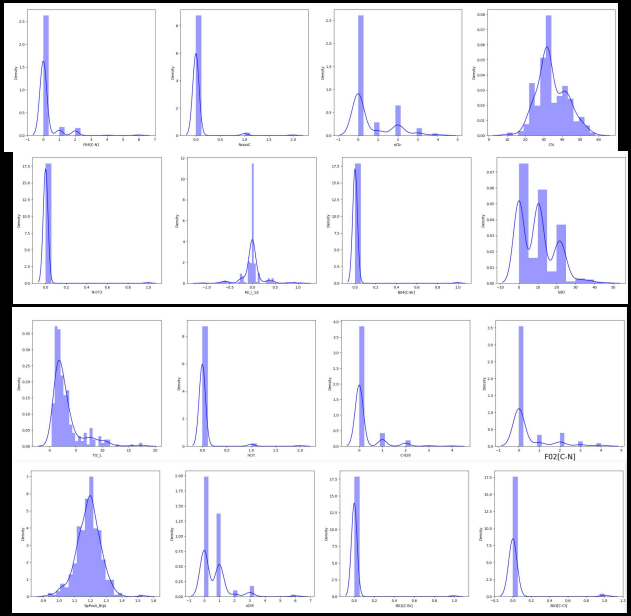
.Interpretation of Histograms:Symmetry: A symmetric distribution has a similar shape on both sides of the central point.

Asymmetric distributions may indicate skewness.Skewness: If the distribution is skewed to the right (positive skew), the tail is longer on the right side. If skewed to the left (negative skew), the tail is longer on the left side.

Central Tendency: The central peak of the histogram indicates the central tendency of the data (mean or median).

Spread: The spread of the histogram provides insights into the variability of the data.


Density
SpMax_L
J_Dz(e)
nHM
F01[N-N]
Mi
nN-N
nArNO2
nCRX3
nCp
nO
F03[C-N]
SdssC
HyWi_B(m)
LOC
F03[C-O]
Me


F02[C-N]

Same Tails of Distribution:Similar Shape: If the tails of the distribution plots look similar across different subplots, it suggests that the variability or spread of values is consistent.

Outliers: Outliers may appear as points far from the main concentration of data in the tails. If you observe consistent outliers across multiple subplots, it might indicate a pattern or commonality.

Same Central Tendencies: Peak Position: If the central peaks (mode) of the histograms are in a similar position across different subplots, it suggests that the central tendency of the data is consistent.

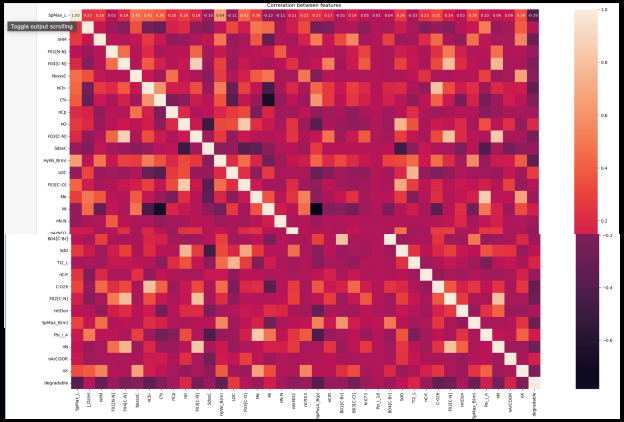
Mean/Median Comparison: the positions of the mean or median lines in each subplo consistently located, it indicates similarity in central tendencies.

Consistent Spread or Variability:
Width of Distribution: The width of the distribution plot represents the spread or variability of the data. If the widths are similar across subplots, it suggests consistent variability

Spread Metrics: Look at measures of spread (e.g., standard deviation) if available. If these metrics are consistent, it reinforces the interpretation of consistent variability.

Skewness:Similar Skewness Patterns: If the skewness of the distribution (leaning to the left or right) is consistent across subplots, it indicates a shared pattern.

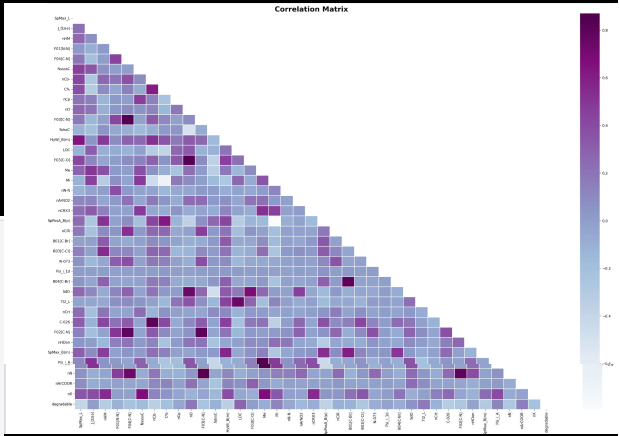
Skewness Magnitude: Compare the magnitude of skewness. If it's consistently positive or negative, it suggests a common skewness direction.

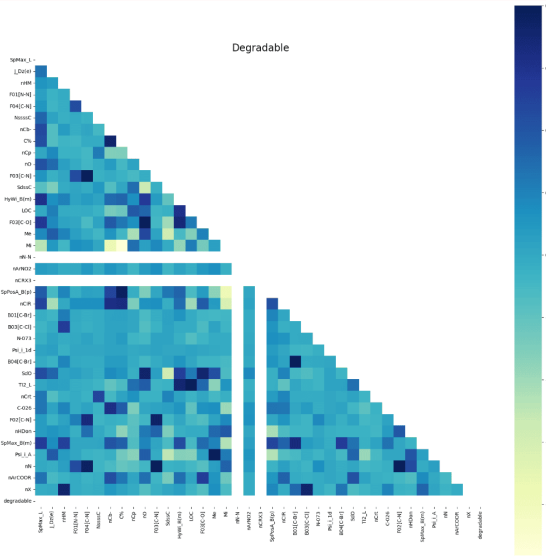


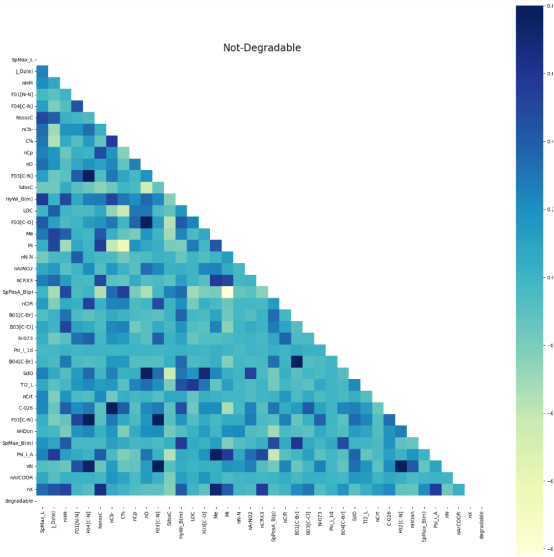

```
corrmat = df.corr()
plt.figure(figsize = (30, 20),)
sns.heatmap(corrmat,
            square=True,
            annot = True,
            annot_kws={'size': 9},
            fmt = ".2f")
plt.title('Correlation between features');
plt.show()
```

Correlation between variables

```python
# Correlation matrix
plt.figure(figsize = (30, 20), dpi = 150)

mask = np.triu(np.ones_like(corrmat, dtype = bool))
sns.heatmap(corrmat,
            mask = mask,
            cmap = 'BuPu',
            annot = True,
            annot_kws={'size': 9},
            linewidths = 0.5,
            fmt = ".2f")

plt.title('Correlation Matrix',
          fontsize = 20,
          weight = 'semibold',
          color = 'Black')
plt.show()
```







If there is no correlation between two variables, it means that the variables do not appear to be statistically related, that the value of one variable doesn't increase or decrease in association with the increase or decrease of the other variable. Here we see that nN-N and nCRX3 are two variables not correlated with the other variables for the Degradable class.

[54]:
```
print("Before OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train == 0)))

# import SMOTE module from imblearn library

from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=1)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train.ravel())

print('After OverSampling, the shape of train_X: {}'.format(X_train_res.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_res.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train_res == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train_res == 0)))
```

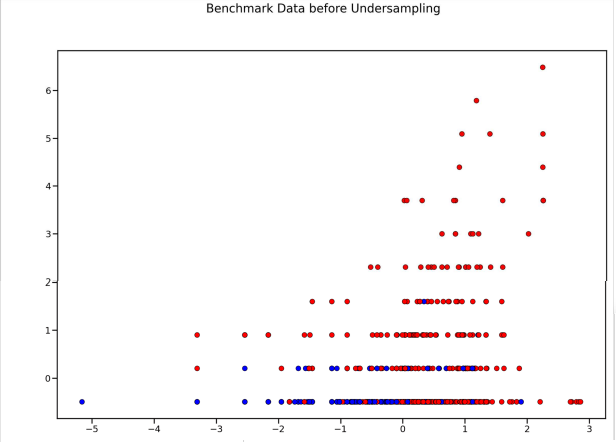
```
Before OverSampling, counts of label '1': 253
Before OverSampling, counts of label '0': 483

After OverSampling, the shape of train_X: (966, 30)
After OverSampling, the shape of train_y: (966,)

After OverSampling, counts of label '1': 483
After OverSampling, counts of label '0': 483
```

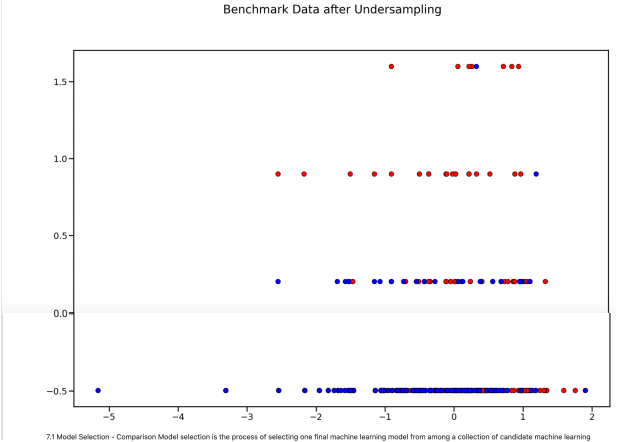
[55]:
```
colors = ['red' if v == 0 else 'blue' for v in y_train]
kwarg_params = {'linewidth': 1, 'edgecolor': 'black'}
fig, av = plt.subplots(figsize=(24,16))
av.scatter(X_train.iloc[:, 0], X_train.iloc[:, 1], c=colors, **kwarg_params)
plt.suptitle("Benchmark Data before Oversampling")
plt.show()
```

This code performs model training, evaluation, hyperparameter tuning, and visualization for a classification problem. It uses three different models (RandomForest, GradientBoosting, SVM) to classify the target variable 'nO' based on the specified features ('nCp', 'C%', 'NssssC') in the DataFrame. The RandomForest model undergoes hyperparameter tuning using GridSearchCV, and the results, including accuracy and the best hyperparameters, are visualized in a bar chart.

```
from sklearn.metrics import confusion_matrix, classification_rep
from sklearn.linear_model import LogisticRegression
# logistic regression object
lr = LogisticRegression(random_state=30)

# train the model on train set
clf = lr.fit(X_train, y_train.ravel())

predictions = lr.predict(X_test)

# print classification report
print(classification_report(y_test, predictions))
```

	precision	recall	f1-score	support
0	0.91	0.87	0.89	214
1	0.75	0.82	0.79	102
accuracy			0.85	316
macro avg	0.83	0.85	0.84	316
weighted avg	0.86	0.85	0.86	316





4. NearMiss Algorithm – Undersampling¶

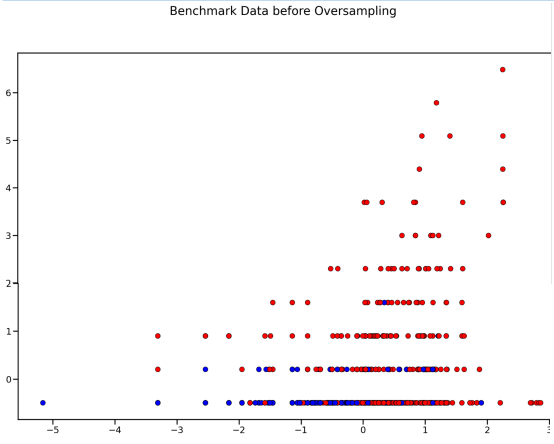
```
[59]: print("Before Undersampling, counts of label '1': {}".format(sum(y_train == 1)))
      print("Before Undersampling, counts of label '0': {} \n".format(sum(y_train == 0)))

      # apply near miss
      from imblearn.under_sampling import NearMiss
      nr = NearMiss()

      X_train_miss, y_train_miss = nr.fit_resample(X_train, y_train.ravel())

      print('After Undersampling, the shape of train_X: {}'.format(X_train_miss.shape))
      print('After Undersampling, the shape of train_y: {} \n'.format(y_train_miss.shape))

      print("After Undersampling, counts of label '1': {}".format(sum(y_train_miss == 1)))
      print("After Undersampling, counts of label '0': {}".format(sum(y_train_miss == 0)))
```

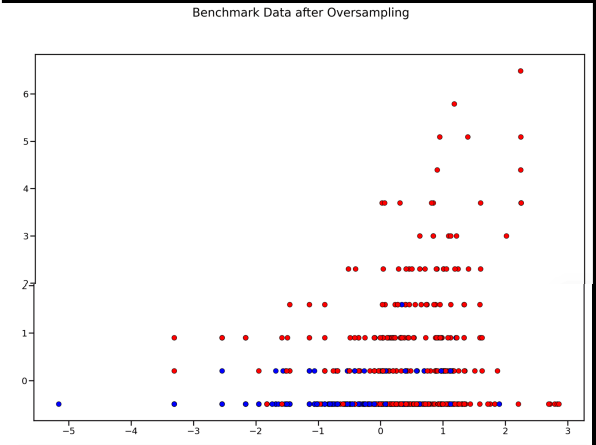
```
Before Undersampling, counts of label '1': 253
Before Undersampling, counts of label '0': 483

After Undersampling, the shape of train_X: (506, 30)
After Undersampling, the shape of train_y: (506,)

After Undersampling, counts of label '1': 253
After Undersampling, counts of label '0': 253
```

```
[60]: colors = ['red' if v == 0 else 'blue' for v in y_train]
      kwarg_params = {'linewidths': 1, 'edgecolor': 'black'}
      fig, ax = plt.subplots(figsize=(24,16))
      ax.scatter(X_train.iloc[:, 0], X_train.iloc[:, 1], c=colors, **kwarg_params)
      plt.suptitle("Benchmark Data before Undersampling")
      plt.show()
```

7.1 Model Selection - Comparison Model selection is the process of selecting one final machine learning model from among a collection of candidate machine learning models for a training dataset. Here we will be using, Cross-validation for evaluating estimator performance. We will also be using the metric AUC_ROC, it is one of the most important evaluation metrics for checking any classification model's performance.

before and after undersampling





[57]:
```
lr1 = LogisticRegression(random_state = 30)
clf_smote = lr1.fit(X_train, y_train.ravel())
predictions = lr1.predict(X_test)

# print classification report
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.91      0.87      0.89       214
           1       0.75      0.82      0.79       102

    accuracy                           0.85       316
   macro avg       0.83      0.85      0.84       316
weighted avg       0.86      0.85      0.86       316
```

Here we see that it doesn't change because we already affected the dataset with our modulation

Conclusion

Reflecting on the key stages of building a Python project and emphasizing the possibilities ,what can be achieved with this project according to me ?

Environmental Impact:

Biodegradation Assessment: The ability to accurately predict whether a substance is biodegradable or not is crucial for assessing its environmental impact. Knowing the biodegradability of a substance helps in understanding its potential harm or lack thereof in natural ecosystems.

Regulatory Compliance:

Environmental Regulations: Many countries have stringent environmental regulations that require companies to ensure the biodegradability of their products, especially in industries such as agriculture, pharmaceuticals, and chemicals. Having a reliable model can aid in compliance with these regulations.

Product Development:

Green Chemistry: In product development, particularly in fields like green chemistry, understanding the biodegradability of chemical compounds is essential. This model can assist researchers and developers in designing products that are environmentally friendly and have minimal ecological impact.

Risk Assessment:

Ecotoxicology Studies: Predicting the biodegradability of substances is integral to ecotoxicology studies, helping researchers and industries evaluate potential risks associated with the introduction of new compounds into the environment.

Data-Driven Decision Making:

QSAR (Quantitative Structure-Activity Relationship): The use of QSAR models allows for data-driven decision-making based on the chemical structure of compounds. This empowers individuals without a deep chemical engineering background, like yourself, to make informed predictions and decisions.

Cost and Time Savings

Reduced Experimental Testing: Building a reliable model for predicting biodegradability can significantly reduce the need for extensive and costly experimental testing. This can lead to substantial savings in both time and resources.

Cross-Disciplinary Collaboration

Integration of Expertise: The project provides an opportunity for cross-disciplinary collaboration between individuals with expertise in data science and those with a background in chemistry or chemical engineering. This collaboration is essential for addressing complex challenges at the intersection of different fields. Educational Value: Learning Opportunity: For someone without a chemical engineering background, this project offers a unique learning opportunity. It allows for the acquisition of knowledge and skills in the application of data science to real-world problems, bridging the gap between different domains.

Potential for Industry Adoption

Scalability: the model developed could be scaled up for broader industry adoption; this project goes beyond its technical challenges by addressing a problem with significant real world implications. It combines environmental responsibility, regulatory compliance, and the application of data science techniques, making it both interesting and strategic in its potential impact.