Practical 1

Instructions

The program assumes dataset path is `data/drug_consumption.data`

```
# For experiment results
python3 src/run.py

# For running the setup which was used to develop for experiment
python3 src/run.py -setup
```

Part 1

How did you load and clean the data, and why?

The data was loaded using `read_csv` from the `pandas` library. This was because the pandas library has features which enable for easy manipulation of a dataset. Given the data file had no headers the `header=None` parameter was set so the first line is included as a data point. As the data was split immediately only the train data was analysed to be used in the cleaning process. The training data was then checked for missing values in any of the entries and none were found, so that did not need to be taken into account. This agreed with the information present in the dataset abstract [4] There were several unnecessary columns within the data file since only columns 2-13 and 30 were required. The extraction of the appropriate features was done using `df._loc` was used to extract the required features and response which were `[1,12]` and `[29]`, since the data frame uses zero index.

Finding Outliers

Logistic Regression assumes that there are no significant outliers[1]. As well, Polynomial Regression or Basis Expansion is sensitive to outliers[12].

The several tests were performed to determine if there existed significant outliers with the 12 respective fields. The first stage was using the `describe` function form the available in the `dataframe` class.

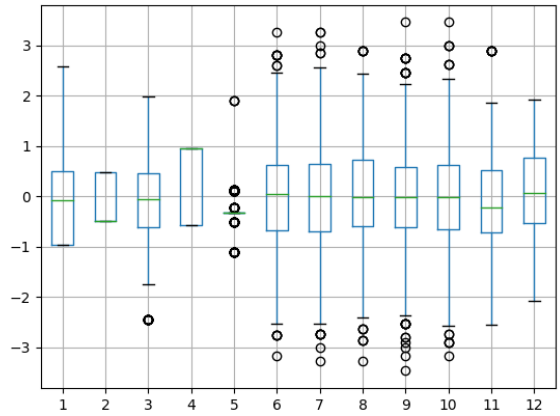
	1	2	3	4	5	6	7	8	9	10	11	12
count	1508.000000	1508.000000	1508.000000	1508.000000	1508.000000	1508.000000	1508.000000	1508.000000	1508.000000	1508.000000	1508.000000	1508.000000
mean	0.044268	-0.011518	-0.014314	0.351340	-0.306679	0.002955	0.006294	0.014810	-0.007587	-0.003141	0.003208	0.000401
std	0.886127	0.482483	0.941600	0.699487	0.168918	0.984704	1.002705	0.989915	0.997438	0.992936	0.963720	0.959899
min	-0.951970	-0.482460	-2.435910	-0.570090	-1.107020	-3.157350	-3.273930	-3.273930	-3.464360	-3.157350	-2.555240	-2.078480
25%	-0.951970	-0.482460	-0.611130	-0.570090	-0.316850	-0.678250	-0.695090	-0.583310	-0.606330	-0.652530	-0.711260	-0.525930
50%	-0.078540	-0.482460	-0.059210	0.960820	-0.316850	0.042570	0.003320	-0.019280	-0.017290	-0.006650	-0.217120	0.079870
75%	0.497880	0.482460	0.454680	0.960820	-0.316850	0.629670	0.637790	0.723300	0.590420	0.628243	0.529750	0.765400
max	2.591710	0.482460	1.984370	0.960820	1.907250	3.273930	3.273930	2.901610	3.464360	3.464360	2.901610	1.921730

The output of the describe function on the training set without the validation set.

It could be seen that there were some abnormalities in the distribution of the data. For example in column 6 the upper-semi-interquartile rate is `0.629670` but the maximum value is `3.273930` which is

significantly far for this value given the standard deviation is below one and the mean if `0.002955`.

To further investigate the existence of outliers box plot was used by the method `boxplot` of the pandas `dataframe` class which could then be visualized using `matplotlib`.



A plot of the boxplot of all the 12 features on the training set without the validation set

The boxplot of all the data showed that significant outliers did exist in several of the columns. Given the assumption of no significant outliers. It was decided that significant outliers would be omitted from training.

> Any z-score greater than 3 or less than -3 is considered to be an outlier. This rule of thumb is based on the empirical rule
> . From this rule we see that almost all of the data (99.7%) should be within three standard deviations from the mean. [9]

There rows with had significant outliers in their features (columns 1-12, inclusively) with $|z_{\text{score}}(x)| < 3$ were removed. This was valid even if the distribution of the features weren't normally distributed[10]

How did you split the data into test/train set and why?

To split the training and testing data the `train_test_split` function was used from `sklearn` [3]. This function shuffles the rows of the data and then splits it by a given ratio, the shuffle feature is enable by default. The ratio used was used 0.2 (The Pareto Principle) which is a standard training testing split in machine learning. Shuffling the data was necessary as prevents bias based upon the ordering of the data. To ensure the reproducibility of the results a seed was used to ensure that the shuffle produces same result on every given running of the program. This ensures that results obtained can be independently verified by others. Additionally for pre-processing and testing the affects that scaling, removing parameters and additionally modifications to the dataset had a validation set was created which was age was 20% of the training data. So for analysis purposes the split constituted of

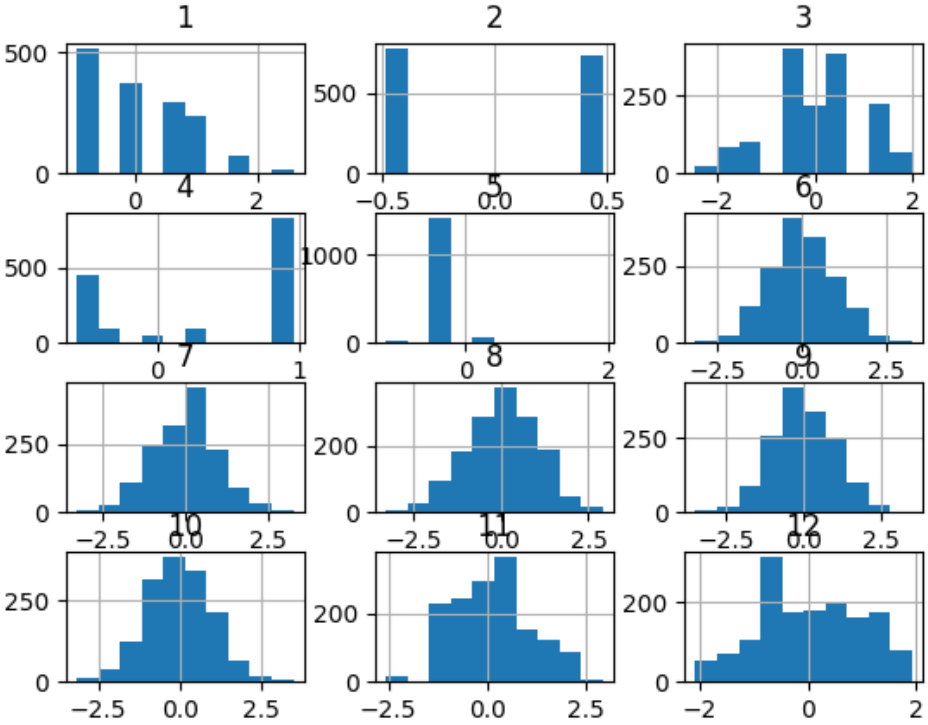
$$\text{Training Data}_{60\%} + \text{Valiodation Set}_{20\%} + \text{Test Set}_{20\%}$$

In the Validation set stratification was used to ensure the validation set was representative sample of the testing set.

How did you process the data including encoding, conversion, and scaling, and why?

The data must be scaled as `sklearn`'s `LogisticRegression` algorithm makes use of quasi-newton optimisation methods. These methods are iteratively and their convergence may be affect by the scaling of the weights. Additionally if penalties such l_1 or l_2 are added feature scaling will influence the penalties applied on each weight. So to adjust for this the data features were normalised to account for this. The prescience outliers outliers affects the computation of the empirical mean and the empirical standard deviation. The outliers would be the values out with the 95% confidence interval or for z score $|z_{\text{score}}| > 1.96$. To adjust for these values it was opted to use the `RobustScaler` over normalization and minmax scaling. Robust scaler use IQR to scale values. Which provide more reflective scaling.

Additionally histograms were plotted to determine the nature of the distributions of the columns. After removal of outliers to determine the distributions of data.

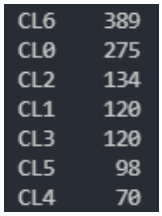


Histogram of all the features of the training set 60% without the validation set

The response variable $\mathbf{y}$ was transformed to make its interpretation as an index even though `sklearn`'s `LogisticRegression` accepts string input.

$$[\mathrm{CL0}, \ldots, \mathrm{CL06}] \to [0, 6]$$

This is advantages as the classes have a clear order but they would not come in order necessary order when indexed. This can be seen in the mere counting of occurrences done in the initial setup of experiment.



How did you ensure that there is no data leakage?

To ensure there was no data leakage the training data and testing data were separated immediately after being loaded into the program. This insured no information from the testing test was used. Also, the use of a seeds in components of the program which relied on randomness ensured that the same partition of testing set was set aside. No data from testing data was used in analysis of the dataset. For measuring the performance of transformations to data a validation set was created to ensure that testing data was not been tuned to model parameters or alterations to dataset. In turn this ensured no testing data was used for any encoding, scaling or conversion process.

Encoders and Scalers were fit the training data and the the transformations could be applied to the testing data based on the values of training. This ensured the values from the testing data weren't used for computation of mean and standard deviation for scaling.

Part 2

Train using `penalty='none'` and `class weight=None`. What is the best and worst classification accuracy you can expect and why?

Theoretically classification accuracy can be $\mathrm{Classification\ Accuracy} \in [0, 1]$ [11] As class can be completely separable or model or model can assign each class incorrectly.

The expectation of randomly guessing is the class with equal probability of each is $\frac{1}{n}$ where n is the number of classes. So for this dataset if the model $f_{\boldsymbol{\theta}}(\cdot)$ was to randomly guess the expectation would be

$$\mathrm{Random\ Guessing\ Accuracy} = \mathbb{E}(f_{\boldsymbol{\theta}}(X)) = \frac{1}{7} = 0.14285...$$

In practice this would be expected worst behaviour of model randomly assigning values to classes.

Explain each of the following parameters used by `LogisticRegression` in your own words: `penalty`, `tol`, `max_iter`.

`penalty`

For selecting the type of regularization penalty that is added to the cost function being optimised.

`tol`

$$|\theta_{t+1} - \theta_t| < \text{Tolerance Level}$$

How close to zero the previous values of theta must be to each other before the algorithm terminates.

`max_iter`

The maximum number of iterations that can happen before convergence to parameter values within tolerance level. i.e. before

$$|\theta_{t+1} - \theta_t| < \text{Tolerance Level}$$

Train using balanced class weights (setting `class_weight='balanced'`). What does this do and why is it useful?

It penalizes mistakes in classes inversely proportionally to the number of samples available from that class. Useful when there are imbalances in the frequencies of classes.

$$\text{learning rate} \propto \frac{1}{\text{number samples in class}}$$

`LogisticRegression` provides three functions to obtain classification results. Explain the relationship between the vectors returned by `predict()`, `decision_function()`, and `predict_proba()`.

`decision_function()`

For each sample input will determine the distance and which side of the decision boundary (for Logistic Regression, the hyperplane) a sample input being predicted would be on for every given classes one for all model's prediction. The higher the value of the index the higher the probability of the sample input belong to the class of that index.

`predict_probability()`

For each sample input that it intends to generate a prediction for returns a vector of the prediction for returns a vector containing the probabilities of that sample belong to sample $0, \ldots, n$. For a vector of sample inputs returns and array of arrays.

`predict()`

predict returns the index of the element in the `predict_probability()` array with the largest probability value for a given sample input.

Part 3

What is the classification accuracy of your model and how is it calculated? Give the formula

$$\text{Classification Accuracy} = \frac{\text{True Positive}}{\text{Total Number of Samples}} = \frac{\text{True Positive}}{\text{Positive} + \text{Negative}}$$

[Refer to results for section for exact values of Classification Accuracy for the 3 models]

What is the balanced accuracy of your model and how is it calculated? Give the formula.

$$\text{Balanced Accuracy} = \frac{\text{Specificity} + \text{Sensitivity}}{2} = \frac{1}{2}(\frac{\text{True Postiive}}{\text{Positive}} + \frac{\text{True Negative}}{\text{Negative}})$$

[Refer to results for section for exact values of Balanced Accuracy for the 3 models]

Show the confusion matrix for your classifier for both unbalanced (2a) and balanced (2c) cases. Discuss any differences.

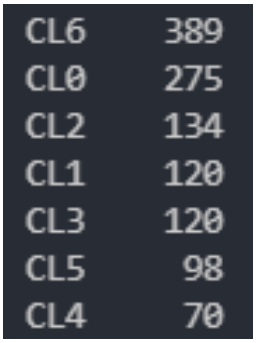
Comparison between 2 unpenalized models.

```
Confusion Matrix
[[35 17  4  3  3  3 19]
 [15  8  3  0  1  0 16]
 [16  4  5  2  1  0  8]
 [ 4  0  5  4  2  1 19]
 [ 2  1  1  2  1  0 14]
 [ 5  0  2  1  5  5 16]
 [25  8  5  5  6  8 67]]
```

penalty=none, class_weight=none

```
Confusion Matrix
[[21 27  8  4  9  7  8]
 [ 7 21  6  0  1  3  5]
 [ 2 10 15  2  4  1  2]
 [ 0  3  7  6  6 11  2]
 [ 3  1  1  5  6  2  3]
 [ 2  1  4  3  8 11  5]
 [15 16 10 18 16 28 21]]
```

penalty=none, class_weight=balanced



Occurrences of the Difference classes from setup phase

As can be send from a sample of the class occurrence data from subset of full training set from which validation set was extracted from the order of occurrence. It can be clearly seen when class weight is balanced that recall is increase on classes which have a low number of samples. e.g. CL_4 and CL_5. This comes with a trade off of having recall reduced on more frequent classes such as CL_6 which recall was significantly reduced from 67 to 21. Even in CL_0 from 35 to 21

Show the precision and recall of your algorithm for each class, as well as the micro and macro averages. Explain the difference between the micro and macro averages.

A micro average will aggregate the contribution of each class to the metric.

A macro average will compute the metric for each class independently. Then average out the metric by summing the metric for each class and dividing by the number of classes.

Example Formula Comparison

$$\text{Precision Micro} = \frac{\sum_{i=1}^{n} \text{True Positive}(i)}{\sum_{i=1}^{n} \text{True Positive}(i) + \text{False Positive(i)}}$$

$$\text{Precision Macro} = \frac{1}{n} \sum_{i=1}^{n} \frac{\text{True Positive}(i)}{\text{True Positive}(i) + \text{False Positive}(i)}$$

where n is the number of classes

[refer to results section for the values requested for all the models]

Part 4

Set the penalty parameter in `LogisticRegression` to `l2`. Give the equation of the cost function used by `LogisticRegression` as the result. Derive the gradient of this `l2`-regularised cost.

$$\min_{\boldsymbol{\theta},\eta}\{\frac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\theta}+\eta\sum_{i=0}^{n}\log(\exp(-y_i(\mathbf{X}_i^T\theta))+1)\}$$

Where:

- $\mathbf{X}$: feature matrix (including intercept)
- n: number of samples
- $\mathbf{y}$: feature vector
- $\boldsymbol{\theta}$: Weight Vector
- η Learning rate

This is equivalent to another common notation

$$\min_{\boldsymbol{\theta},\eta}\{-\frac{\lambda}{2}\boldsymbol{\theta}^T\boldsymbol{\theta}-\alpha\sum_{i=0}^{n}\log(\exp(-y_i(\mathbf{X}_i^T\theta))+1)\}$$

By setting $\alpha=\lambda^2$ and dividing though by $-\lambda$ and subbing in η for λ.

Implement a 2nd degree polynomial expansion on the dataset. Explain how many dimensions this produces and why

In the case of this dataset, including the intercept of 1 for polynomial regression there are $\binom{13}{2}$ possible ways to choose 2 items from from the set of features $\{1,x_1,\ldots,x_{12}\}$ to multiply each other where ordering doesn't matter. Additionally, 13 needs to be added on to represent the multiplication of each item with itself. e.g. $[1,x_1^2,\ldots,x_{12}^2]$

The intuition for the $\binom{13}{2}$ comes from the idea that there are 13 possible features I can select for the first item to be multiplied. Then 12 items to choose from for the second item to multiple the first item. 1 less as there has been 1 item already selected to be multiplied for the set already. Also, there are 2 possible was to order these items. e.g. x_1x_2 or x_2x_1. So, we can divide by 2 (the number of possible orderings to product same output), since we only care about the numerical value produced. This leaves the following expression.

$$\frac{13\times12}{2}=\frac{13!}{2!(13-2)!}=\binom{13}{2}$$

This takes a general form for n features for 2nd degree polynomial polynomial expansion expansion

$$\binom{n+1}{2}+n=\binom{n+1}{2}+\binom{n}{1}=\binom{n+2}{2}=\text{Dimension of Feautres}$$

but for the features used in this dataset it takes the form.

$$\binom{13}{2} + 13 = \binom{14}{2} = 91$$

In the `PolynomialFeatures` class `include_bias=False` set to false in the because logistic regression automatically adds intercept to the data. This altered the total number of columns is 90.

Compare the results of regularised and unregularised classifiers on the expanded data and explain any differences.

```
CLASS WEIGHTS
CL_0:
        Magnitude: 49.96962045709244
        Maximum: 46.376962740556124
        Minimum: -12.032046971154738
        Closest to Zero: 0.004449618688060643
CL_1:
        Magnitude: 49.76656343249826
        Maximum: 42.89751465881249
        Minimum: -12.880248101287073
        Closest to Zero: 8.782091859140689e-05
CL_2:
        Magnitude: 41.31388330664906
        Maximum: 35.7330643596467
        Minimum: -9.681103022759073
        Closest to Zero: 0.009040721931754148
CL_3:
        Magnitude: 125.27311833197466
        Maximum: 29.305465483027884
        Minimum: -111.18247799371206
        Closest to Zero: 0.002952287707458181
CL_4:
        Magnitude: 51.71328520844759
        Maximum: 38.97415804449136
        Minimum: -15.241655603506832
        Closest to Zero: 0.005647385543282585
CL_5:
        Magnitude: 131.6904021869745
        Maximum: 45.16722210571893
        Minimum: -74.5028069552473
        Closest to Zero: 0.0017774025833317145
CL_6:
        Magnitude: 28.843960926462238
        Maximum: 21.703585145331413
        Minimum: -8.581625747279187
        Closest to Zero: 0.010973664688361539
```

penalty=none, class_weight=balanced

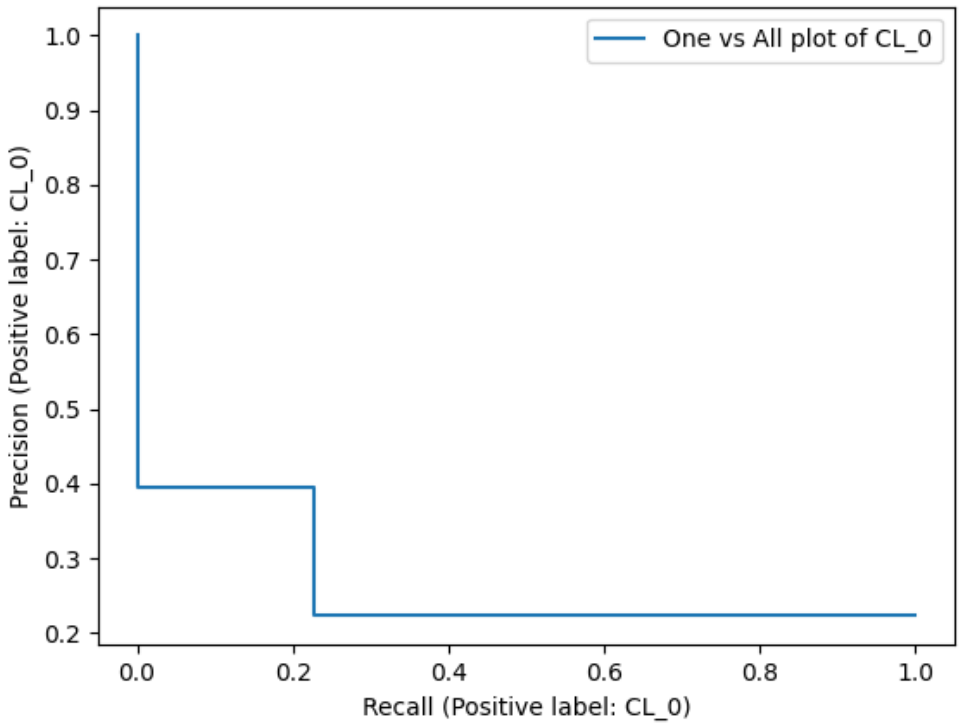
```
CLASS WEIGHTS
CL_0:
        Magnitude: 2.2477195133149723
        Maximum: 0.7927268513674963
        Minimum: -0.7521793745281746
        Closest to Zero: 0.006421392282702541
CL_1:
        Magnitude: 3.900943501799571
        Maximum: 2.567989189894592
        Minimum: -0.7090670220067421
        Closest to Zero: 0.0017231051216098
CL_2:
        Magnitude: 3.160329747440048
        Maximum: 0.9239494548415804
        Minimum: -0.708532640166635
        Closest to Zero: 0.0018734562384956886
CL_3:
        Magnitude: 3.045378965788155
        Maximum: 0.7914982259131105
        Minimum: -0.8701037712717608
        Closest to Zero: 0.00025812964639053735
CL_4:
        Magnitude: 4.9810340628353025
        Maximum: 1.15023772998873
        Minimum: -1.9198632095134596
        Closest to Zero: 4.58644225396458e-05
CL_5:
        Magnitude: 3.6319909242956983
        Maximum: 0.6742762571664562
        Minimum: -1.0404437744509536
        Closest to Zero: 0.026644483213657595
CL_6:
        Magnitude: 2.19964910912693
        Maximum: 0.5640227199800937
        Minimum: -0.5485283452788818
        Closest to Zero: 0.0037580629568805483
```

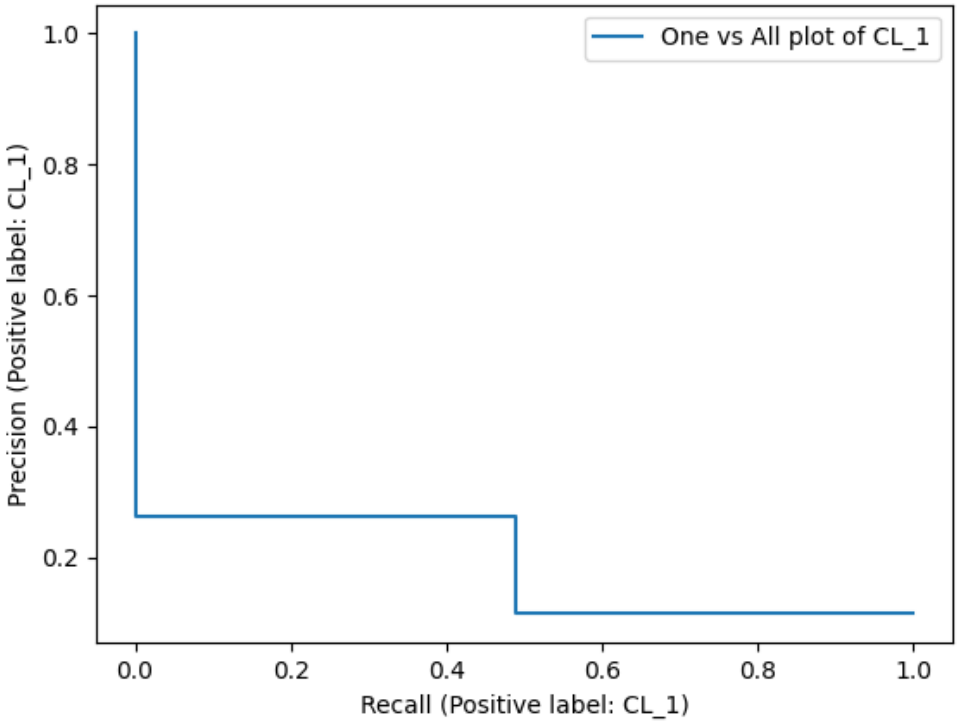
penalty=l2, class_weight=balanced

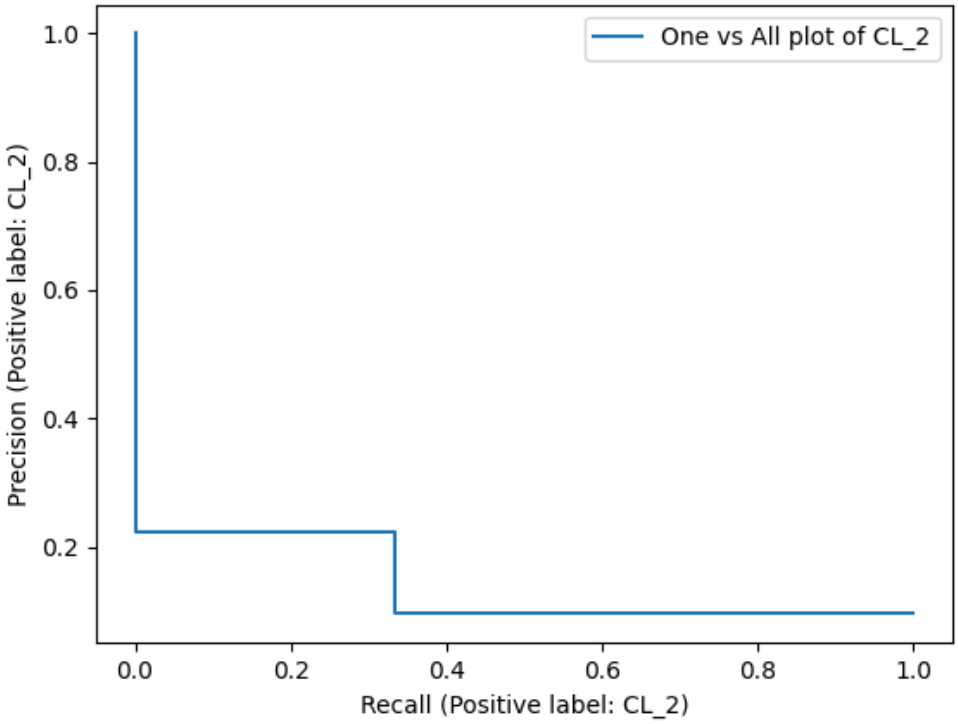
The weights of the classed were not analysed at individually compared as there were too many of them. The `l2` penalty produced a significantly lower magnitude (the square root of the sum of the squares of the weights) of weights for each class as the `l2` punishes for large weights. This effect is also evident in the maximum weight was much larger in these models with `l2` and with minimum being smaller too. These finding coinciding the with idea that larger square values of weights are punished.

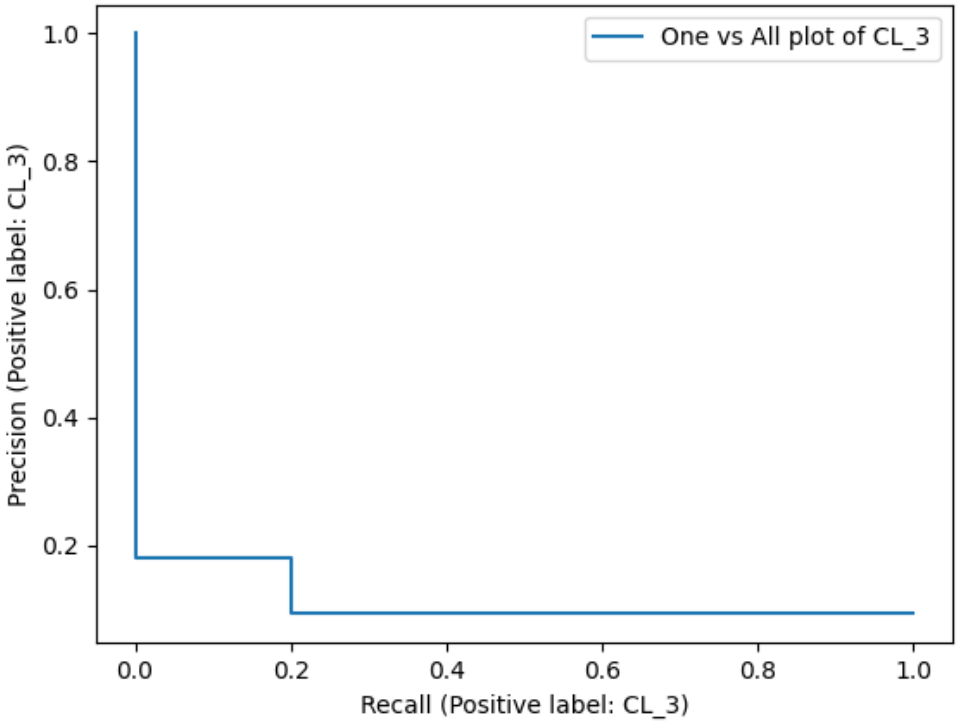
Extend your solution to provide Precision-Recall plots for each class of nicotine consumption. You may need to independently explore advanced scikit-learn functionality for this

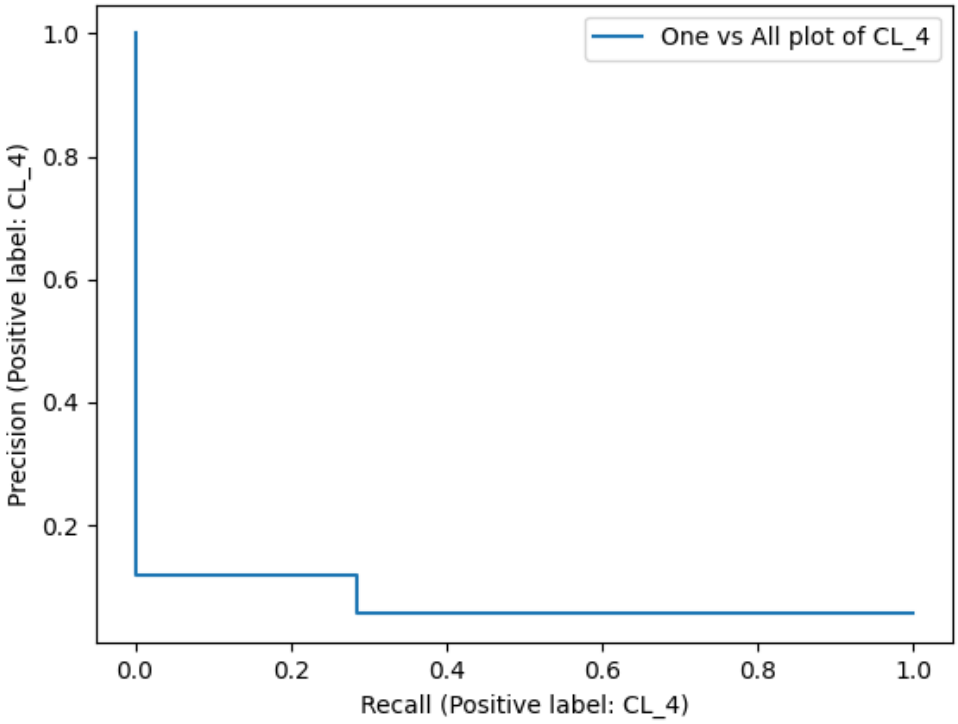
The plots were made with `pentaly = 'l2', class_weight='balanced'` in addition to having polynomial features of degree 2.

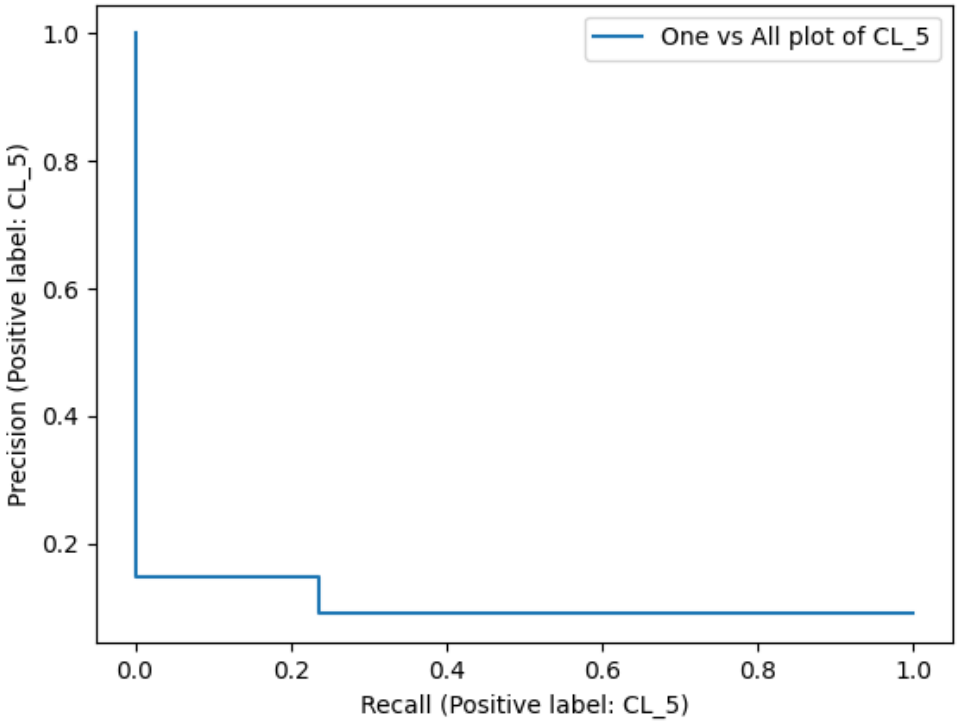


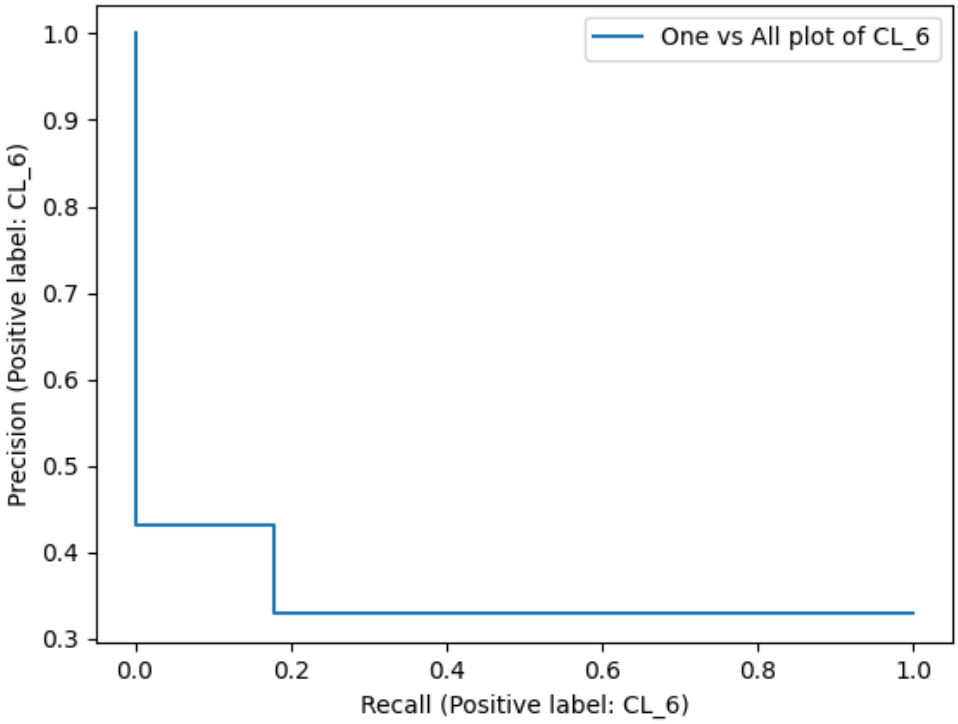

One vs All plot of CL_1
Precision (Positive label: CL_1)
Recall (Positive label: CL_1)
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0


One vs All plot of CL_2
Precision (Positive label: CL_2)
Recall (Positive label: CL_2)
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0


One vs All plot of CL_3
Precision (Positive label: CL_3)
Recall (Positive label: CL_3)
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0


One vs All plot of CL_4
Precision (Positive label: CL_4)
Recall (Positive label: CL_4)
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0


One vs All plot of CL_5
Precision (Positive label: CL_5)
Recall (Positive label: CL_5)
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0


One vs All plot of CL_6
Precision (Positive label: CL_6)
Recall (Positive label: CL_6)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.0
0.2
0.4
0.6
0.8
1.0

Results

Intentionally emission of full `CLASS WEIGHT` section for brevity. (Only Magnitude of Class Weights are shown)

```
Model: Penalty='none', class_weight=None

Classification Accuracy
0.33156498673740054

Balanced Accuracy
0.22726974760897106

Confusion Matrix
[[35 17  4  3  3  3 19]
 [15  8  3  0  1  0 16]
 [16  4  5  2  1  0  8]
 [ 4  0  5  4  2  1 19]
 [ 2  1  1  2  1  0 14]
 [ 5  0  2  1  5  5 16]
 [25  8  5  5  6  8 67]]

Precision: micro
0.33156498673740054

Precision: macro
0.2510129374490611

Precison: samples
CL_0: 0.3431372549019608
CL_1: 0.21052631578947367
CL_2: 0.2
CL_3: 0.23529411764705882
CL_4: 0.05263157894736842
CL_5: 0.29411764705882354
CL_6: 0.42138364779874216

Recall: micro
0.33156498673740054

Recal: macro
0.22726974760897106

Recall: samples
CL_0: 0.3431372549019608
CL_1: 0.21052631578947367
CL_2: 0.2
CL_3: 0.23529411764705882
CL_4: 0.05263157894736842
CL_5: 0.29411764705882354
CL_6: 0.42138364779874216

CLASS WEIGHTS
CL_0: 34.88172684451584
CL_1: 32.24455152132051
CL_2: 28.412407914472325
CL_3: 85.58906827276068
CL_4: 34.659267521888964
CL_5: 84.84214131894473
CL_6: 17.23830262577064
```

```
Model: Penalty='none', class_weight=balanced

Classification Accuracy
0.26790450928381965

Balanced Accuracy
0.3007236953295947

Confusion Matrix
[[21 27  8  4  9  7  8]
 [ 7 21  6  0  1  3  5]
 [ 2 10 15  2  4  1  2]
 [ 0  3  7  6  6 11  2]
 [ 3  1  1  5  6  2  3]
 [ 2  1  4  3  8 11  5]
 [15 16 10 18 16 28 21]]

Precision: micro
0.26790450928381965

Precision: macro
0.26985144034923786

Precison: samples
CL_0: 0.42
CL_1: 0.26582278481012656
CL_2: 0.29411764705882354
CL_3: 0.15789473684210525
CL_4: 0.12
CL_5: 0.1746031746031746
CL_6: 0.45652173913043476

Recall: micro
0.26790450928381965

Recal: macro
0.3007236953295947

Recall: samples
CL_0: 0.42
CL_1: 0.26582278481012656
CL_2: 0.29411764705882354
CL_3: 0.15789473684210525
CL_4: 0.12
CL_5: 0.1746031746031746
CL_6: 0.45652173913043476

CLASS WEIGHTS
CL_0: 49.96962045709244
CL_1: 49.76656343249826
CL_2: 41.31388330664906
CL_3: 125.27311833197466
CL_4: 51.71328520844759
CL_5: 131.6904021869745
CL_6: 28.843960926462238
```

```
Model: Penalty='l2', class_weight=balanced

Classification Accuracy
0.2519893899204244

Balanced Accuracy
0.2780462372495885

Confusion Matrix
[[19 27 14  4  6  5  9]
 [ 7 21  6  0  1  2  6]
 [ 4 10 12  1  4  3  2]
 [ 0  3  8  7  6  9  2]
 [ 1  1  1  5  6  2  5]
 [ 2  1  2  6 10  8  5]
 [15 17 11 16 18 25 22]]

Precision: micro
0.2519893899204244

Precision: macro
0.2510300701477172

Precison: samples
CL_0: 0.3958333333333333
CL_1: 0.2625
CL_2: 0.2222222222222222
CL_3: 0.1794871794871795
CL_4: 0.11764705882352941
CL_5: 0.14814814814814814
CL_6: 0.43137254901960786

Recall: micro
0.2519893899204244

Recal: macro
0.2780462372495885

Recall: samples
CL_0: 0.3958333333333333
CL_1: 0.2625
CL_2: 0.2222222222222222
CL_3: 0.1794871794871795
CL_4: 0.11764705882352941
CL_5: 0.14814814814814814
CL_6: 0.43137254901960786

CLASS WEIGHTS
CL_0: 2.2477195133149723
CL_1: 3.900943501799571
CL_2: 3.160329747440048
CL_3: 3.045378965788155
CL_4: 4.9810340628353025
CL_5: 3.6319909242956983
CL_6: 2.19964910912693
```

References

1. Logistic regression: a brief primer, https://pubmed.ncbi.nlm.nih.gov/21996075/

2. What are outliers in the data?**,** https://www.itl.nist.gov/div898/handbook/prc/section1/prc16.htm
3. https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#standardscaler
4. https://archive.ics.uci.edu/ml/datasets/Drug+consumption+(quantified)
5. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
6. https://scikit-learn.org/stable/modules/preprocessing.html
2. https://medium.com/swlh/stop-one-hot-encoding-your-categorical-features-avoid-curse-of-dimensionality-16743c32cea4#:~:text=One-hot encoding categorical variables,problem of parallelism and multicollinearity.
3. Pattern Recognition and Machine Learning, Christopher M. Bishop. Chapter 1.4. The Curse of Dimensionality

9. https://www.ctspedia.org/do/view/CTSpedia/OutLier#:~:text=Any z-score greater than,standard deviations from the mean.

10. https://sonalsart.com/does-z-score-have-to-be-normal-distribution/

11. http://37steps.com/4891/classification-errors/

12. https://www.analyticsvidhya.com/blog/2021/07/all-you-need-to-know-about-polynomial-regression/#:~:text=Polynomial Regression is a form of Linear regression known as,also badly affect the performance.