190004947

Practical 1 - Drug Consumption

CS5014 - Machine Learning

Part 1 - Data Processing

(a) How did you load and clean the data, and why?

I loaded the dataset using pandas with `pandas.read_csv()` which returns a pandas.DataFrame which is the data type we will be using to store our training, validation, and testing subsets. It is a two-dimensional, size-mutable table structure.

To clean the dataset I first dropped the first column which stores the record ID as it is completely arbitrary and has no relevance to the classification model. I then dropped the columns containing the consumption data for all drugs except for nicotine; the target class stored in the thirtieth column.

Since the UCI Machine Learning Repository [1] has no information about whether or not the drug consumption dataset contains missing values and so I calculated the counts of entries containing null of NaN values and summed them to obtain 0. This shows that the dataset contained no missing values.

Lastly I checked if there were any duplicate records. By using the pandas.DataFrame.duplicated() function I obtained a boolean series where each entry would contain False if it was a unique row and True if it was a duplicate. By converting this series to a numpy array and calculating the sum of values, we get an integer representing the number of duplicate rows in our dataset. In this case, the result was zero indicating no duplicate records.

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

Sklearn provides a method called train_test_split() to which I passed five parameters: a DataFrame of the input features, a DataFrame of the output classes, a test size proportion, a boolean for shuffling, and a random state.

The test size proportion I passed in was 0.3 which tells the train_test_split() method to put 70% of the data into the training subset and 30% into the testing subset for final model evaluation. By setting shuffle to true, my testing subset won't just contain the bottom 30% of the complete dataset, rather it will randomly select samples from across the dataset. The random state parameter is simply a seed for the random shuffling which I set to an arbitrary integer to allow for reproducibility of results.

Once my model was built, I tried varying the proportion of data for training and testing. I found that a 70-30 split produced the best classification accuracy on the test set.

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

All input features are originally categorical and have been quantified by the researchers that produced the drug consumption dataset [1]. After quantification, values of all input attributes are real-valued.

The researchers have already encoded the input data and so no further encoding was required. To quantify the ordinal inputs, they used a polychoric correlation technique which is based on the assumption that values of ordinal features come from the discretisation of continuous random values with fixed thresholds [2]. For the nominal features, such as gender, country, and ethnicity, the researchers implemented non-linear categorical principal component analysis (CatPCA).

As part of my logistic regression implementation, I trained my model on a dataset that used one-hot encoding for each of the three categorical features mentioned above. However, this resulted in a decline in classification accuracy and so I continued with the preset encoding.

Since the output classes represent increasing likelihood of nicotine use (they are ordinal), encoding via enumeration is appropriate as the machine learning algorithm treats the order of numbers as significant. If the output classes were not related and ordered then perhaps a one-hot encoding would produce better results.

For data conversion, in my first attempt at building the classifier, I implemented whitening to remove correlation between features. This process begins by centering the data by subtracting each datapoint by the mean of its row before normalising the data using the MinMaxScaler() provided by sklearn.preprocessing. Once the data is normalised, it can be decorrelated. To decorrelate the features, I initially calculated the covariance matrix of the input dataset and then used numpy's linear algebra module to obtain the eigenvalues and eigenvectors. The DataFrame can then be multiplied by the matrix of eigenvectors using the dot product which achieves decorrelation.

Lastly, the decorrelated data is divided by the square root of the eigenvalues to normalise the scale. The resultant dataset will have a correlation matrix equal to the identity matrix.

In Lecture 9 on data preparation, we learnt that some algorithms work better when features are not correlated. Unfortunately, my efforts were not rewarded and my one-vs-all classifier produced a lower classification accuracy.

After I split the data into training and testing subsets, I instantiated a MinMaxScaler like the one mentioned above and fit it with the training input set. I then used the transform method to apply the normalisation to both the training input and the testing input. Min-max normalisation is achieved using the formula:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

I also ran some tests using the sklearn StandardScaler to standardise the data instead of normalising it. I thought that scaling with respect to the mean and standard deviation would produce better results than scaling with respect to the minimum and maximum values, however, the model preferred min-max normalisation.

(d) How did you ensure that there is no data leakage?

To ensure that there was no data leakage, I split the dataset into training and testing subsets before any calculations or conversions. This is critical otherwise the model will be trained with some knowledge of the test set.

Additionally, when scaling the data, the MinMaxScaler is only fit to the training data before being used to transform the training and testing data. If new, unseen data became available, we would apply the fitted scaler to the new data and run the classifier as we did with the testing set and we could expect to achieve a similar level of accuracy.

Part 2 - Training

(a) Train using `penalty='none'` *and* `class_weight=None`*. What is the best and worst classification accuracy you can expect and why?*

Since penalty='none' and class_weight=None, there is nothing stopping us from applying an extremely complex model with a high degree polynomial basis expansion to closely fit the skewed distribution of the training data. Since there are only 1885 samples and 428 of them fall into CL0 and 610 of them fall into CL6, if the model can accurately classify the binary classes of user and non-user then it could achieve a classification accuracy of above 55% assuming it can guess some of the remaining classes correctly.

On the other hand, a model that was only trained on the CL4 samples would only ever know CL4. In the case that a model predicts all samples to be in CL4, the maximum classification accuracy it could achieve would be around 6% given that CL4 labels only account for 108 of the 1885 samples in the dataset.

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

The penalty parameter is used to specify whether the logistic regression model should use L1 regularisation or L2 regularisation. Regularisation is used to punish large weights which reduces the risk of overfitting the data. When no regularisation is used, the loss function of logistic regression is:

$$L(\theta) = -\sum_n [y^n ln(\sigma(\theta^T x)) + (1 - y^n) ln(1 - \sigma(\theta^T x))]$$

When regularisation is applied, we transform the loss function into a cost function:

$$J(\theta) = L(\theta) + R(\theta)$$

For L1 regularisation (a.k.a basis pursuit), we add the term:

$R(\theta) = \lambda ||\theta||_1$ which is the L1 norm (sum of absolute values).

For L2 regularisation (a.k.a. Tikhonov regularisation), we add the term:

$R(\theta) = \lambda ||\theta||_2^2$ which is the Euclidean norm (sum of squares of all elements).

Introducing regularisation by setting the penalty parameter to something other than 'none' will potentially reduce performance on the training set in order to improve performance on the validation and test sets. This is an effect of avoiding overfitting the training data allowing for increased generalisation.

The tol parameter represents the tolerance for the stopping criteria. Logistic regression can be used with different algorithms for optimising the loss/cost function by changing the 'solver' parameter. Each of these will iteratively find a better θ (bias and weights) by running some calculation. As θ approaches the global minimum, the differences between iterations of the convergence function will become smaller and smaller. The model will stop optimising θ once these differences fall below the selected tolerance value.

As mentioned above, the model iteratively improves our matrix of weights (and bias) by applying some convergence function; gradient descent, for example. To avoid running arbitrarily long computations to optimise θ, sklearn allows you to pass in a maximum number of iterations to the LogisticRegression object. If the optimisation function has not converged after this many iterations, the computation will stop and an error will be output explaining that the maximum number of iterations was reached.

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

The class_weight parameter allows us to choose between ‘balanced’, None, and a custom dictionary of weightings. It is used to counteract the effect of a dataset with imbalanced target class frequency. Training a model on such a dataset without setting class weights may result in the model learning the distribution of the output to gain false accuracy. For example, imagine we have a dataset with 99% of the examples belonging to class A and 1% belonging to class B. The classification model could completely ignore the training inputs and just guess in accordance to this distribution. Such a model would know nothing about the correlation between features and classes but would be able to achieve a high classification accuracy.

In logistic regression, class weights alter the loss function by weighting the loss of each sample by its class weight. This is useful for our case of drug consumption classification as the model will be penalised more for incorrectly predicting CL6, the modal class, than it would for incorrectly predicting CL4, the anti-modal class. When this difference in penalisation is proportioned to the frequency of classes, the model will not learn the distribution and instead will have to find correlation with the input features as intended.

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

The predict method takes input features as a parameter and returns class predictions by comparing the probability of samples being in a specific class to a threshold. For logistic regression, each binary case is calculated by $\sigma(\theta^T X) > 0.5$.

The decision_function method takes input features as a parameter and returns the confidence scores for each classification prediction calculated by $\theta^T X$. These confidence scores represent the distance of the classification probability to the decision boundary hyperplane.

The predict_proba method takes input features as a parameter and returns the probability of each sample being in each of the classes calculated by $\sigma(\theta^T X)$.

Since we are using the one-vs-all approach where be train N binary classifiers (one class vs all other classes), the predict_proba method will calculate the probability for each class and then normalise these values across all the classes.

The relationship between the vectors returned by decision_function and predict_proba is exactly the mapping predict_proba = σ(decision_function). It then follows that the relationship between the vectors returned by predict_proba and the predict method is the comparison of the values to some threshold, e.g. 0.5.

Part 3 - Evaluation

(a) What is the classification accuracy of your model and how is it calculated? Give the formula.

Classification accuracy is defined as the number of testing examples that are correctly classified as a proportion of the total number of examples:

Number of true predictions / total number of predictions = (TP + TN) / (P + N) where:

TP = true positives = total number of data points correctly classified as positive.

TN = true negatives = total number of data points correctly classified as negative.

P = total number of positive examples = TP + FN.

N = total number of negative examples = TN + FP.

Calculated in the model using accuracy_score from sklearn's metrics module as 40.64%.

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

For multiclass classification, the balanced accuracy is calculated by averaging the recall value for each class:

$\frac{\sum_n \frac{TP_n}{TP_n + FN_n}}{n}$ where n = the number of classes.

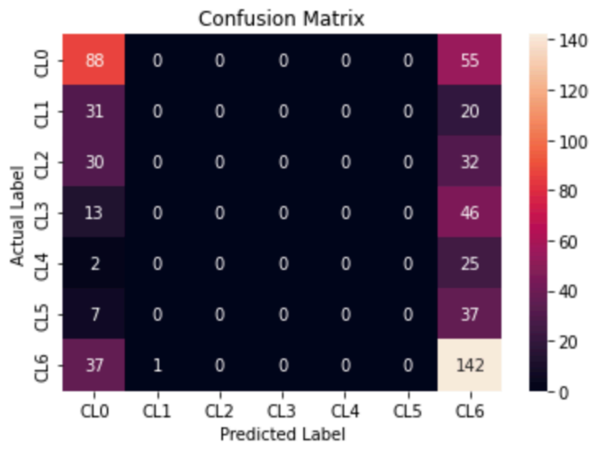
The balanced accuracy for my model was calculated using sklearn's classification_report as:

(0.62+0+0+0+0+0+0.79)/7 = 0.20

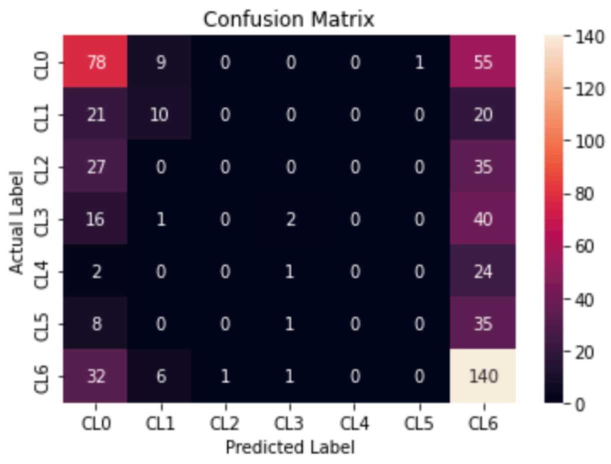
(c) Show the confusion matrix for your classifier for both unbalanced (2a) and balanced (2c) cases. Discuss any differences.

The confusion matrix is a cross table that records the number of occurrences of the actual classification and the predicted classification. Using columns for the predicted class and rows for the actual class, we can easily see the correct classifications across the diagonal of the matrix.

Confusion matrix for my model when unbalanced (class_weight='balanced'):



Confusion matrix for my model when balanced (class_weight=None):

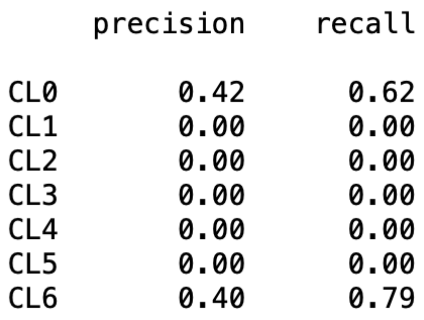


Both confusion matrices show the interesting behaviour of the model which is that it only really predicts CL0 and CL6. We know that this is not an example of the model learning the distribution of the output labels since the behaviour is reproduced in the unbalanced classifier. I predict that the way the researchers encoded the data in the dataset is the reason behind this.

One difference between the two confusion matrices is that the unbalanced model predicted CL3 (the median class) five times whereas the balanced model never predicted CL2, CL3, CL4 or CL5 with only a single prediction for CL1. Since the output labels represent ordinal data and are encoded as such, I think the reason behind this is that the unbalanced model had a very low confidence for those samples and didn't know whether the input features corresponded to a user or non-user of nicotine.

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

Using classification_report from sklearn's metrics module I calculated the following precision and recall values of my model for each class:



```
         precision    recall

CL0         0.42       0.62
CL1         0.00       0.00
CL2         0.00       0.00
CL3         0.00       0.00
CL4         0.00       0.00
CL5         0.00       0.00
CL6         0.40       0.79
```

Micro average precision = 0.41, macro average precision = 0.12.

Micro average recall = 0.41, macro average recall = 0.20.

Micro averages for precision and recall are calculated by summing the true positives for each class and dividing that sum by the number of positives predicted in total for precision or by the number of actual positives for recall.

Macro averages for precision and recall are calculating by summing the precision and recall values for each class and dividing the sum by the total number of classes to give a mean class value.

In other words, micro averages represent a class-wide average while macro averages represent a mean class average, essentially an average of averages.

Part 4 - Advanced Tasks

(a) 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.*

Cost function of L2-regularised LogisticRegression:

$$J(\theta) = L(\theta) + R(\theta)$$

Where:

$$L(\theta) = -\sum_n [y^n ln(f_\theta(x^n)) + (1-y^n) ln(1-f_\theta(x^n))]$$

$$R(\theta) = \lambda ||\theta||_2^2$$

So:

$$J(\theta) = \lambda ||\theta||_2^2 - \sum_n [y^n ln(f_\theta(x^n)) + (1-y^n) ln(1-f_\theta(x^n))]$$

Substituting $f_\theta(x^n) = \sigma(z)$, this gives the gradient of the L2-regularised cost as:

$$\frac{\partial J(\theta)}{\partial \theta_i} = \lambda \nabla_\theta ||\theta||_2^2 - \sum_n [y^n \frac{\partial ln(\sigma(z))}{\partial \theta_i} + (1-y^n)\frac{\partial ln(1-\sigma(z))}{\partial \theta_i}]$$

Expanding $\nabla_{\theta_i} ||\theta||_2^2$ we obtain:

$$\nabla_{\theta_i} ||\theta||_2^2 = \frac{\partial \sqrt{\sum_i \theta_i^2}^2}{\partial \theta_i} = \frac{\partial \sum_i \theta_i^2}{\partial \theta_i} = \sum_i \frac{\partial \theta_i^2}{\partial \theta_i} = \sum_i 2\theta_i = 2\theta$$

Given $z = \theta^T x$, we can use the chain rule to obtain:

$$\frac{\partial ln(\sigma(z))}{\partial \theta_i} = \frac{\partial ln(\sigma(z))}{\partial z}\frac{\partial z}{\partial \theta_i}$$ and

$$\frac{\partial ln(1-\sigma(z))}{\partial \theta_i} = \frac{\partial ln(1-\sigma(z))}{\partial z}\frac{\partial z}{\partial \theta_i}$$

Which allows us to apply to chain rule again to obtain:

$$\frac{\partial ln(\sigma(z))}{\partial z} = \frac{\partial ln(\sigma(z))}{\partial \sigma(z)}\frac{\partial \sigma(z)}{\partial z} \text{ and}$$

$$\frac{\partial ln(1-\sigma(z))}{\partial z} = \frac{\partial ln(1-\sigma(z))}{\partial \sigma(z)}\frac{\partial \sigma(z)}{\partial z}$$

Then, using $\frac{\partial ln(x)}{\partial x} = \frac{1}{x}$ we evaluate:

$$\frac{\partial ln(\sigma(z))}{\partial \sigma(z)} = \frac{1}{\sigma(z)} \text{ and}$$

$$\frac{\partial ln(1-\sigma(z))}{\partial \sigma(z)} = \frac{-1}{1-\sigma(z)}$$

Now, where $z = \frac{1}{1+e^{-z}}$, we can substitute $g = 1 + e^{-z}$ and use the chain rule to obtain:

$$\frac{\partial \sigma(z)}{\partial z} = \frac{\partial \frac{1}{g}}{\partial g}\frac{\partial g}{\partial z} \text{ which,}$$

with $\frac{\partial g}{\partial z} = -e^{-z}$ and the rule $\frac{\partial \frac{1}{g}}{\partial g} = \frac{-1}{g^2}$, becomes$\frac{-1}{g^2} \cdot -e^{-z}$

This then simplifies:

$$\frac{-1}{g^2} \cdot -e^{-z} = \frac{-1}{g^2}(1-g) = \frac{1}{g^2}(g-1) = \sigma(z)^2(\frac{1}{\sigma(z)} - 1) = \sigma(z)(1-\sigma(z))$$

Coming back up, we collect up our pieces:

$$\frac{\partial ln(\sigma(z))}{\partial z} = \frac{\partial ln(\sigma(z))}{\partial \sigma(z)}\frac{\partial \sigma(z)}{\partial z} = \frac{1}{\sigma(z)}\sigma(z)(1-\sigma(z)) = 1-\sigma(z) \text{ and}$$

$$\frac{\partial ln(1-\sigma(z))}{\partial z} = \frac{\partial ln(1-\sigma(z))}{\partial \sigma(z)}\frac{\partial \sigma(z)}{\partial z} = \frac{-1}{1-\sigma(z)}\sigma(z)(1-\sigma(z)) = -\sigma(z)$$

Evaluating $\frac{\partial z}{\partial \theta_i} = x_i$, we can put it all together to get:

$$\frac{\partial ln(\sigma(z))}{\partial \theta_i} = \frac{\partial ln(\sigma(z))}{\partial z}\frac{\partial z}{\partial \theta_i} = (1 - \sigma(z))x_i \text{ and}$$

$$\frac{\partial ln(1 - \sigma(z))}{\partial \theta_i} = \frac{\partial ln(1 - \sigma(z))}{\partial z}\frac{\partial z}{\partial \theta_i} = -\,\sigma(z)x_i$$

Finally, we have concluded that:

$$\frac{\partial J(\theta)}{\partial \theta_i} = 2\lambda\theta - \sum_n [y^n(1 - \sigma(z))x_i^n - (1 - y^n)\sigma(z)x_i^n]$$

$$= 2\lambda\theta - \sum_n [y^n - y^n\sigma(z) - \sigma(z) + y^n\sigma(z)]x_i^n$$

$$= 2\lambda\theta - \sum_n [y^n - \sigma(z)]x_i^n = 2\lambda\theta - \sum_n [y^n - f_\theta(x^n)]x_i^n$$

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

Applying a 2nd degree polynomial expansion using PolynomialFeatures on the dataset which includes 12 feature columns produces 91 dimensions. We get all linear combinations of 1, X, and X^2 for each feature X with an order of interaction less than or equal to 2. This gives us one dimension for the bias (combinations of 1), one dimension for each X, one dimension for each X^2, and $\frac{12(12-1)}{2} = 66$ dimensions for each unique pair of X which totals 91 dimensions.

The PolynomialFeatures class from sklearn excludes all features that are a product of more than 2 terms, e.g. $X_1^2 X_2$, as this is a third order interaction term. This is different to the notion of basis expansion discussed in lectures in which all linear combinations of 1, X, and X^2 are included.

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

L2-regularised results:

K fold cross validation accuracy = 39.50% (1.33% std)
Training accuracy = 40.03%
Testing accuracy = 39.93%
Precision:
    Macro average = 0.18
    Micro average = 0.40
    Weighted average = 0.27
Recall:
    Macro average = 0.20
    Micro average = 0.40
    Weighted average = 0.40
F1:
    Macro average = 0.16
    Weighted average = 0.29

L1-regularised results:

- K fold cross validation accuracy = 36.54% (1.37% std)
- Training accuracy = 35.71%
- Testing accuracy = 35.69%
- Precision:
    - Macro average = 0.22
    - Micro average = 0.36
    - Weighted average = 0.30
- Recall:
    - Macro average = 0.24
    - Micro average = 0.36
    - Weighted average = 0.36
- F1:
    - Macro average = 0.22
    - Weighted average = 0.32

Unregularised results:

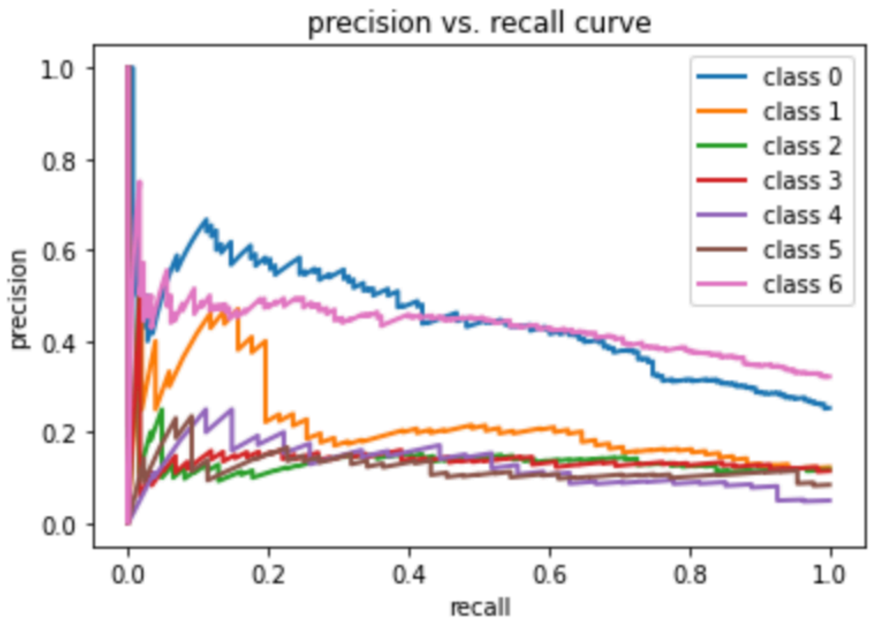
- K fold cross validation accuracy = 27.98% (2.18% std)
- Training accuracy = 35.78%
- Testing accuracy = 23.32%
- Precision:
    - Macro average = 0.23
    - Micro average = 0.23
    - Weighted average = 0.32
- Recall:
    - Macro average = 0.24
    - Micro average = 0.23
    - Weighted average = 0.23
- F1:
    - Macro average = 0.21
    - Weighted average = 0.25

L1 and L2 regularisation is used to punish large weights in order to avoid overfitting the data. It will potentially reduce the performance on the training set in favour of performance on the test set (increased generalisation).

As expected, the unregularised model suffered when it came to the unseen testing data. Curiously however, my regularised model performed better on the training data when we would've expected the unregularised model to have a higher training accuracy due to the overfitting. I suspect that this result is due to the unregularised model using a different solver parameter for LogisticRegression (lbfgs vs liblinear).

(d) 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.

```
# Plot the Precision-Recall curves for each class of nicotine consumption
precision = dict()
recall = dict()
y_test_for_curve = y_test.copy()
for i in range(7):
    precision[i], recall[i], _ = precision_recall_curve(y_test_for_curve, probEstimates[:, i], pos_label=i)
    plt.plot(recall[i], precision[i], lw=2, label='class {}'.format(i))
plt.xlabel("recall")
plt.ylabel("precision")
plt.legend(loc="best")
plt.title("precision vs. recall curve")
```



References

Data Preparation

[1] archive.ics.uci.edu. (n.d.). *UCI Machine Learning Repository: Drug consumption (quantified) Data Set*. [online] Available at: https://archive.ics.uci.edu/ml/datasets/Drug+consumption+(quantified).

[2] Fehrman, E., Mirkes, E., Muhammad, A., Egan, V. and Gorban, A. (2017). *The Five Factor Model of personality and evaluation of drug consumption risk*. [online] Available at: https://arxiv.org/pdf/1506.06297.pdf.