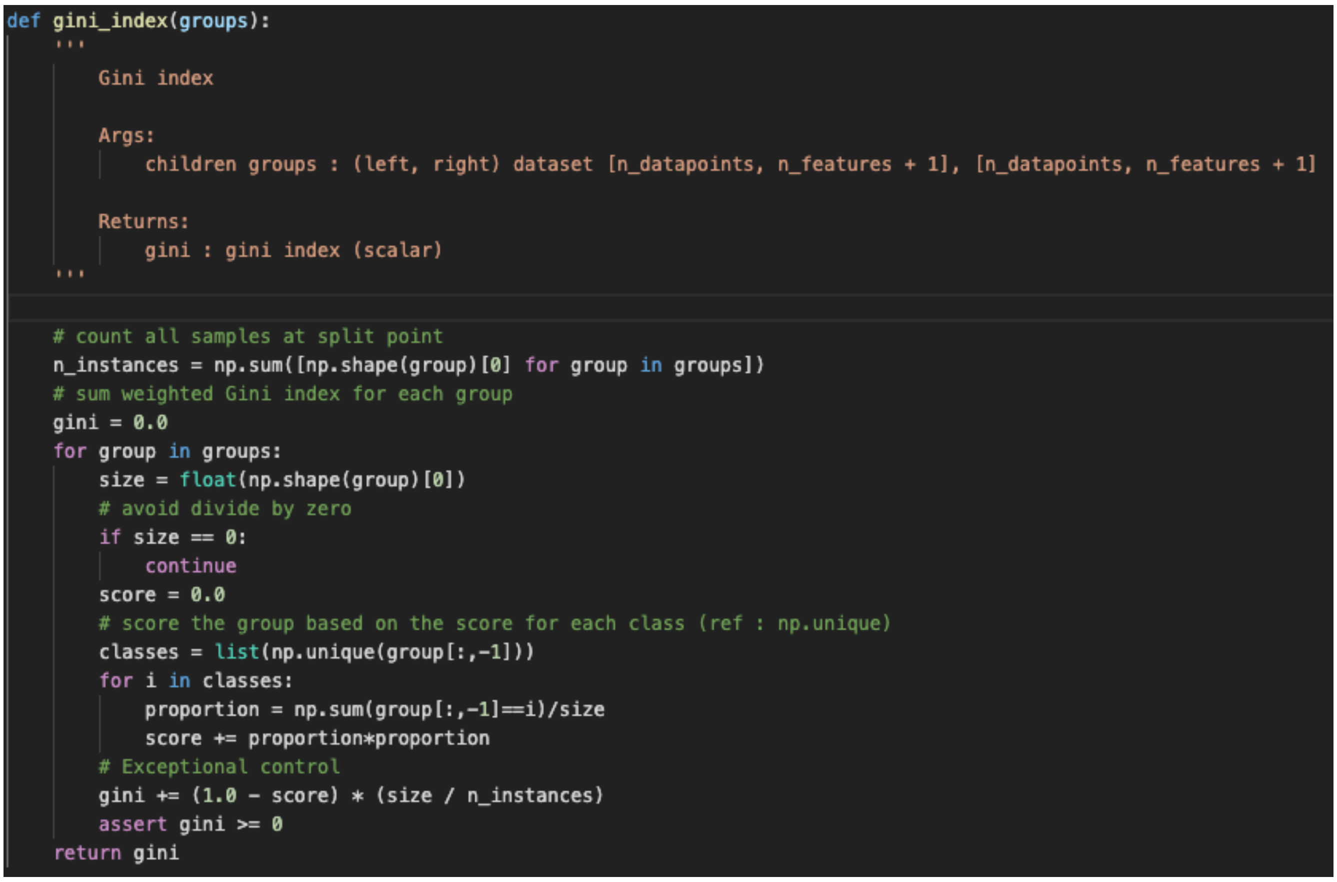
Assignment 3 Report

20180396 오지오

A.

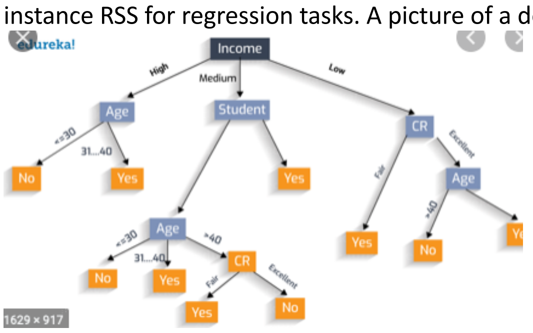


```python
def gini_index(groups):
    '''
        Gini index

        Args:
            children groups : (left, right) dataset [n_datapoints, n_features + 1], [n_datapoints, n_features + 1]

        Returns:
            gini : gini index (scalar)
    '''

    # count all samples at split point
    n_instances = np.sum([np.shape(group)[0] for group in groups])
    # sum weighted Gini index for each group
    gini = 0.0
    for group in groups:
        size = float(np.shape(group)[0])
        # avoid divide by zero
        if size == 0:
            continue
        score = 0.0
        # score the group based on the score for each class (ref : np.unique)
        classes = list(np.unique(group[:,-1]))
        for i in classes:
            proportion = np.sum(group[:,-1]==i)/size
            score += proportion*proportion
        # Exceptional control
        gini += (1.0 - score) * (size / n_instances)
        assert gini >= 0
    return gini
```

B.

<Decision Tree>

Before starting to talk about the metrics in the decision tree algorithm that we used, CART. Let's first look simply at what exactly is the decision tree. A decision tree is used, when we classify or predict a phenomenon based on the data. The internal nodes, where the predictor is split, work as criteria for judgement and the terminal nodes denote the result of the decision. In other words, in each internal node, we decide, where the newly arrived data will be classified to (left and right), and this data will finally arrive at one of the several terminal nodes, in which the value of the node will be the result (value) of the data. We tend to divide the predictor space into distinct, non-overlapping regions, so that every observation falls into one of those regions. For the observations that fall into a specific region, let's say R_2, we tend to make a same prediction for the observations, which is the mean of the response values in that region, R_2. For trees, we divide the predictor space into boxes, where its edges are parallel to the axes. Due to computational reasons, we cannot take every possible partition into consideration, thus we take a greedy approach, applying recursive binary splitting. The best split is determined at each step, in other words, at the immediate space. Using these properties, we try to minimize the error rate by training, for instance RSS for regression tasks. A picture of a decision tree is displayed below.



<CART Decision Tree>

There are several types of decision trees that we utilize and for this assignment I've used the CART (Classification and Regression Tree), which can solve both regression and classification problems. Now, let's see the methods that we use to make such trees above. For trees for regression, we use RSS for evaluation and training of the model. This RSS formulation is written as $\sum_{j=1}^{J}\sum_{i\in R_j}(y_i - \hat{y}_{R_j})^2$, where $i \in R_j$, indicates the observations falling in the region/box. Plus, $\hat{y}_{R_j}$ indicates the mean of the response values in that region and J indicates the total number of regions. For trees used for classification, we usually use various measures. In our model, we use the Gini index, which is one of the most widely used measure. This measures the total variance across J classes. This is defined as,

$$G = \sum_{j=1}^{J} \hat{p}_{mj}(1-\hat{p}_{mj}),\ \hat{p}_{mj} = \frac{\sigma_{y=R_j}}{|S|}, S{:}\,training\ set$$

In our implementation it has the form of

$$G = \left(1 - \sum_{j=1}^{J} \hat{p}_{mj}{}^{2}\right) * w$$

, where w indicates the weighting term telling us that the Gini index for each group must be weighted by the size of the group. The above two are the same formulas since, $\sum_{j=1}^{J}\hat{p}_{mj}$=1.
This shows a similar form as the binomial distribution, thus when $\hat{p}_{mj}$ is close to 1 or zero (e.g. One-hot vector), the Gini index tends to be small. On the other hand, if $\hat{p}_{mj}$ is close 0.5, the Gini index tends to have maximal values. Due to these properties, we Gini index is referred to as a measure of node purity. (Since, a small value tells us that a node contains predominantly observations from a single class. Purity is a measure of the extent which clusters contain a single class)

Using the information above, we tend to make the tree by splitting and pruning, though pruning seems to be omitted in our assignment. Starting with the splitting part, we recursive split the regions into two by the following steps. After ordering the data based on the standard input, such as age in the above tree, we divide the dataset into two, such that it is first data vs the rest. We calculate the Gini index and now split in another way, for instance first and second data vs the rest. We repeat this, until we get the optimal cutpoint (where it leads to the smallest Gini index+ greatest reduction to the Gini index). Then, we peat the process, looking for the other best predictor and cutpoint to split the data furthermore. We use a greedy approach, thus we split one of the previously identified regions, so we get three regions after this process. We repeat this until the criterion we've set, determined by max_depth or min_size is reached. Pruning is done to prevent overfitting, which happens a lot for trees. When we grow a very large tree, we tend to prune it back to get a subtree, by giving a penalty to the original cost function. This penalty term is related to the number of terminal nodes, which is a bit similar to LASSO. The modified cost function is denoted as the following, $ERR(T) + \alpha|T|$. We tend to minimize this, where alpha is a hyperparameter. The bigger the alpha the simpler the model and we tend to find the optimal alpha by cross validation.

Now that we've seen briefly about how we construct our decision tree, let's explore some metrics that can be used to measure the node purity besides the Gini index.

<LSD / Variance Reduction>
For regression trees, the CART algorithm tends to minimize the Least Squares Deviation, which is very similar to the RSS in normal regression tasks. This metric measures the sum of squared distances between the observed and predicted values and tries to minimize this. The mathematical equations look like the following.

$$LSD = \sum_{j=1}^{J}\sum_{i\in R_j}(y_i - \hat{y}_{R_j})^2$$

<Cross Entropy>
Cross entropy measures the amount of randomness in the data. This entropy value ranges from 0 to 1, where 0 indicates that all members belong to the same class, while 1 indicates that there is perfect randomness: the same is divided equally. The ID3 tree uses this metric by setting the node as the leaf node if the entropy is zero and split the tree further when the entropy is bigger than zero. One thing to take note is that the information gain is the decrease in Entropy. The mathematical equation of cross entropy can be denoted as,

$$D = -\sum_{j=1}^{J} \hat{p}_{mj}\, log\hat{p}_{mj}$$

This uses the binomial log likelihood and performs very similar to the Gini index.

<Information Gain>

For a ID3 decision tree, which is mostly used for classification, it uses a metric called the information gain. This concept, extracted from the Information Theory refers to the decrease in the level of randomness in a set of data. This measures how well a given attribute separate the training samples, and tells us how much information a feature gives us about the class. Unlike, the Gini index, the tree tries to maximize this metric, in other words select the attribute with the highest information gain. This concept is related to cross entropy, which is mentioned above. This information gain term can be expressed as the following. Before and after tells us the state before and after the split.

$$IG = Entropy(before) - \sum_{j=1}^{J} Entropy(j, after)$$

<Gain Ratio>

Information gain is a good technique that we could use in the ID3 trees, however it is quite biased. It selects attributes that have large numbers as values of root nodes. In order to overcome this shortcoming, we use the term called the gain ratio, and the tree that uses this is called C4.5. It tries to erase the negative parts of the ID3, by taking the number of branches that would result before making the split into consideration. It normalizes the information gain by using the split information value. The mathematical equation can be expressed as the following.

$$GR = \frac{IG}{split\ info} = \frac{IG}{-\sum_{j=1}^{J} \hat{p}_{mj} * log_2^{\hat{p}_{mj}}}$$

Additionally, in C4.5, we use the windowing techniques, where it randomly selects a subset of a training data (window) and build a decision tree from the particular selection. Then, this tree is used to classify the rest of the data and if this classification is precise, then we finish the tree. However, if it is wrong, we add the misclassified points to the dataset and repeat the process until every instance is classified correctly.

<Chi-Square/F-tests>

The CHAID tree, standing for Chi-Squared Automatic Interaction Detector, uses this method, which is one of the algorithms that allow the tree to have more than two branches. This tree is applied both for regression and classification. For classification, CHAID uses Chi-square tests in order to find the best split. This checks if there are relationship between the two variables and is applied at each stage, so that each branch is associated with the statistically most important predictor of the response variable. It tends to choose the independent variable that has the strongest interaction with the dependent variable. If the categories of each predictors aren't much that different w.r.t to the dependent variable, the categories are merged. We measure this by the sum of squares of standardized differences between the observed and the expected values of the variable. The higher the value of the Chi-Square , the higher the statistical difference between the parent node and the child node. Chi-Squared is represented mathematically as the following.

$$\chi^2 = \sum \frac{(O-E)^2}{E}, O = observed, E = expected$$

We calculate the Chi-Square in each split by calculating the chi-square for an individual node by deviation for the success and failure both. Then we calculate the chi-square using the sum of all chi-square of success and failure of each node at the split.

For regression trees, we use the F-tests to find the difference between the two population means. If this test result is significant, we create a new partition, since it differs from the parent node statistically. If the test result is insignificant, the categories are merged to a single node. The F-test statistic can be written as the variance of 1st dataset divided by the variance of the 2nd dataset.

$$F = \frac{\frac{SS_{bewteen}}{B-1}}{\frac{SS_{within}}{N_w(r) - B}}$$

Besides this, we have various other methods such as the likelihood-ratio Chi-Squared, where it is also useful for measuring the significance of the information gain criterion.

$$G^2 = 2ln2|S|$$

Also, DKM criterion, which requires smaller trees, is a criterion based on the impurity metrics, used for binary class attributes. Its mathematical formula looks like the following.

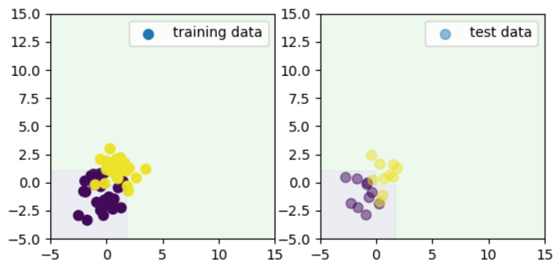
$$DKM = 2\sqrt{\hat{p}_{mj}(1-\hat{p}_{mj})}$$

Plus, in order to overcome some problems with the Gini index, that occurs when the domain of the target is wide, Twoing Criterion was proposed. Not only that there are ORT, Kolmogorov-Smirnov, AUC-splitting criterion that helps the performance of trees.

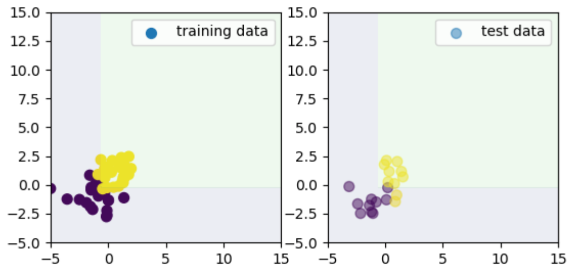
C.

Dataset 0~4 (without outliers)

- Average Accuracy: 87.00
- Standard error: 6.7823



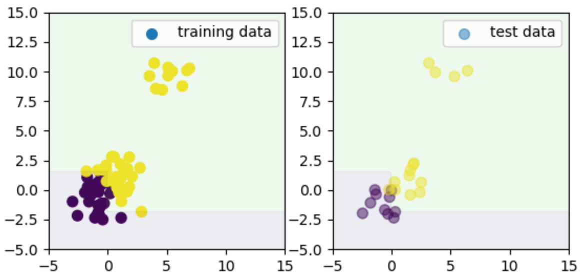
dataset1



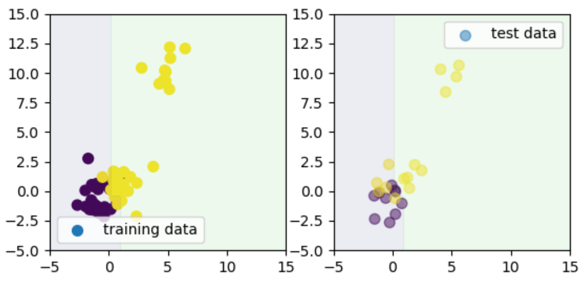
dataset2

Dataset 5~9 (with outliers)

- Average Accuracy: 86.6667
- Standard Error: 10.3414



dataset6



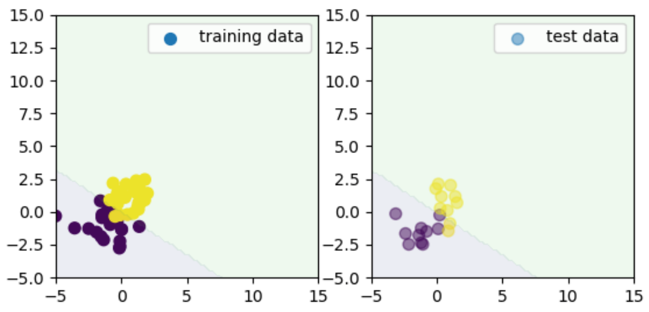
dataset8

My Interpretation

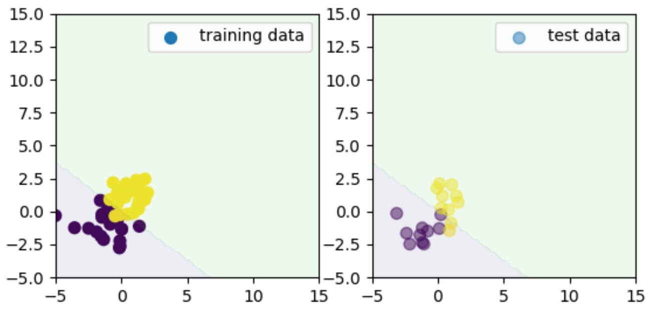
The outliers didn't really affect the accuracy of the model itself in this dataset, however the standard error of the model increased drastically. Though the standard error increased a lot, I believe that the decision tree itself is quite robust. The high standard error rate was invoked since the dataset8 showed a very low accuracy (66%) compared to others and this seems to happen due to the high variance the decision tree itself contains, rather than the outliers. The decision trees divide the items by lines and the distance between the point and the line doesn't really give an effect to the split decisions. The nodes are determined by the sample proportions in each of the regions, not the values itself. Therefore, it seems that the decision trees are robust to outliers, theoretically and both by experience, since the accuracy itself didn't really have much change.

D.

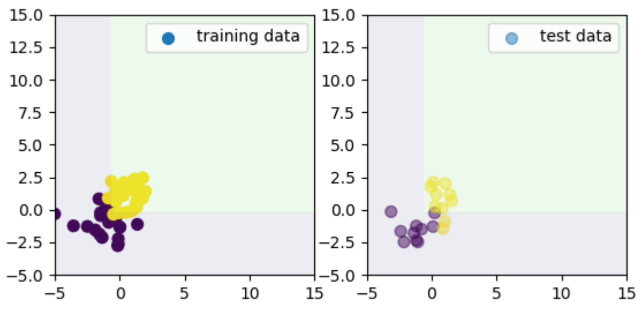
Without Outliers



LDA dataset2

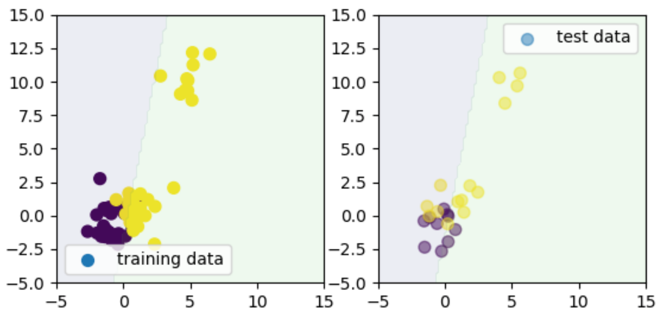


Logistic Regression dataset2

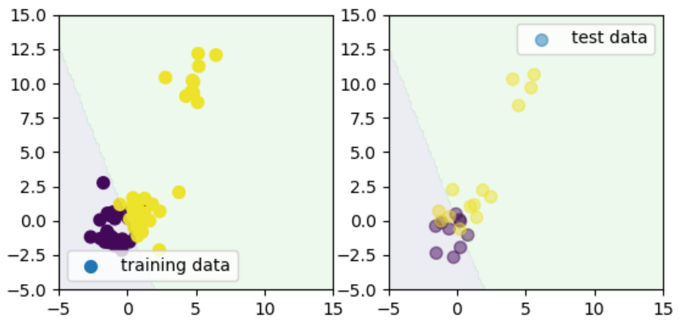


CART tree dataset2

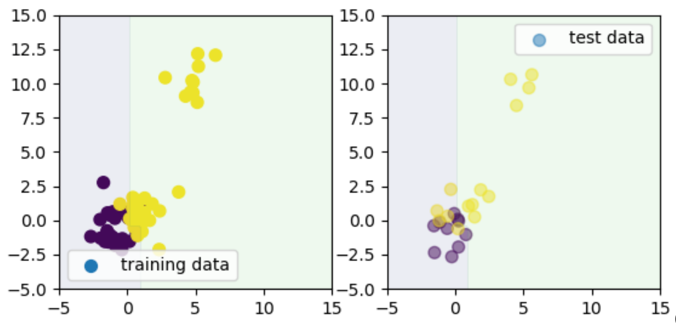
With outliers



LDA dataset8



Logistic Regression dataset8



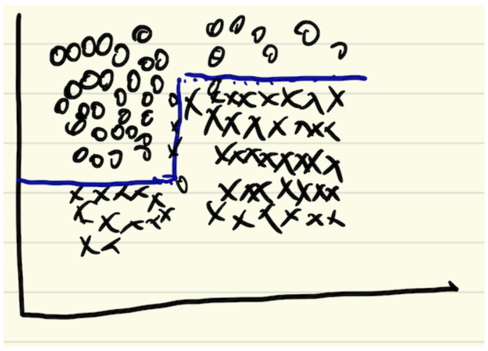
CART Tree dataset8

The biggest difference between the LDA and the logistic regression and the decision tree is the shape of the boundary. (When there is no feature engineering in logistic regression and it is linear) Both the logistic regression model and the LDA shows a linear boundary as a form of a continuous linear function, that divides the two regions. Trees divide the "predictor space" into distinct, "non-overlapping regions", so that every observation falls into one of those regions. Due to this property, the decision boundaries are noncontinuous and are parallel to the axes. This has some limitations for some datasets, especially when the dataset cannot be divided in a horizontal or vertical way, leading to bad performances for some dataset, such as dataset 8.
Now let's compare some properties of LDA/LR/DT. Logistic regression has good probabilistic interpretation and the algorithm can be regularized to prevent overfitting. Plus, it can be updated easily by the SGD, however, they ought to underperform when multiple or nonlinear decision boundaries are required. Plus, due to low flexibility it is not optimal to capture some complex relationships. LDA provides a low-dimensional view for the data, which might be useful when there are numerous response classes. Also, it is easy to compute, since there is an explicit solution for the process. But it requires the normal distribution for inputs. On the other hand, trees are the most interpretative and is very good for visualization. It also performs feature selection implicitly without any external efforts. Plus, it allows to perform on both regression and classification tasks. However, it is very easily overfitted, since it is prone to create over-complex trees. Also, if two variables have similar expressiveness in the model, a slight change in the selection of the variable, might lead to a substantial change in the model. Also, a slight change in the dataset might also give a big change in the model, leading to high variance and low stability. Moreover, as mentioned above when the dataset cannot be divided well by horizontal or vertical ways, the performance will be very poor. Lastly, it is likely to create biased trees if some classes dominate.

G.
Bank Dataset

- Average Accuracy: 96.7233
- Standard Error: 0.7618

Unlike the synthetic dataset, the trees work optimally for the bank dataset, with a very high accuracy and a very low standard error rate. It seems that this could happen due to the dataset's distribution, which has a distribution that gives the best result for trees. It might have a distribution that allows the two classes to be divided well by horizontal or vertical lines. The following graph might be one of the cases.



The following figure is the actual distribution of one of the training datasets. It definitely seems like the area for each class can be well divided by horizontal +vertical lines. (done in tempo_plot method)

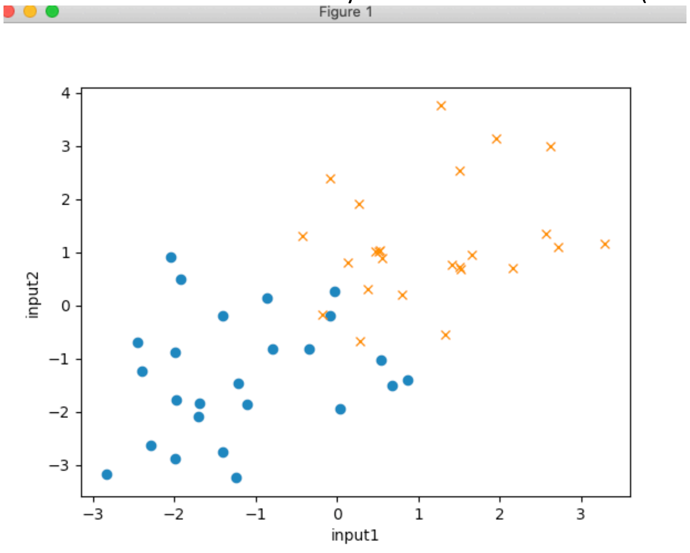


Figure 1
input1
input2