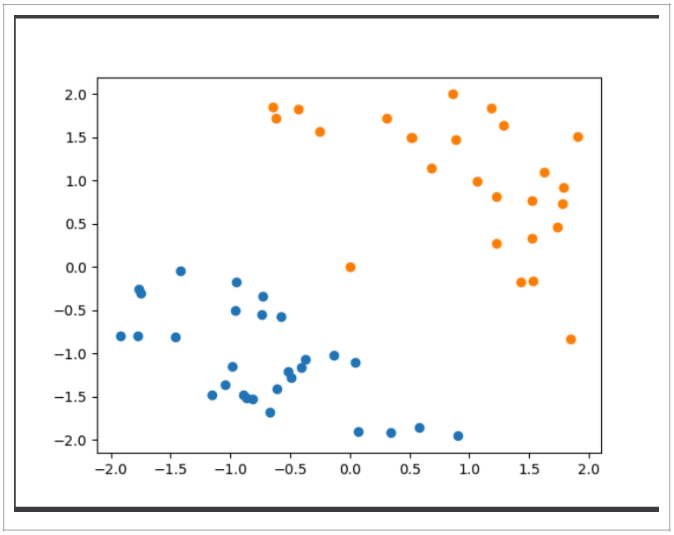
Support Vector Machines

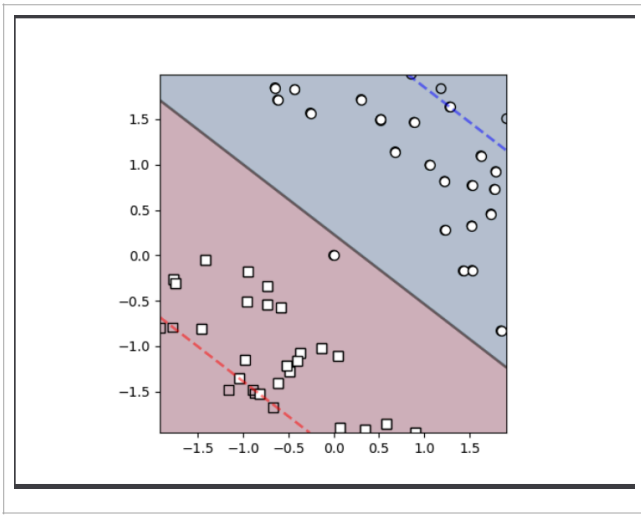
Task 1

Before any implementation is applied, our training data distribution on the graph looks like this:

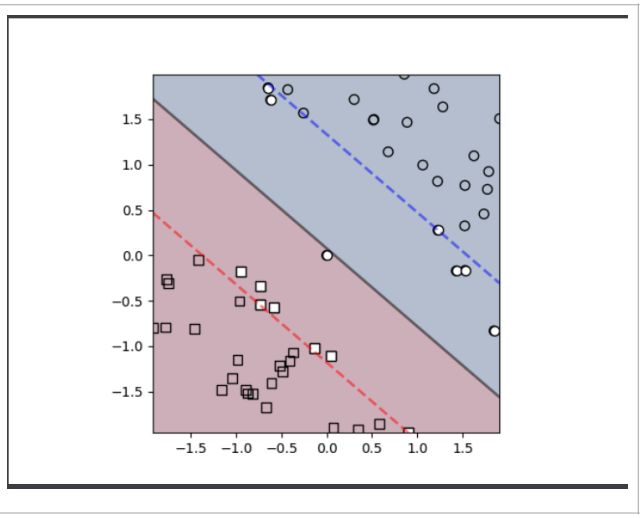


SVM implementation with:

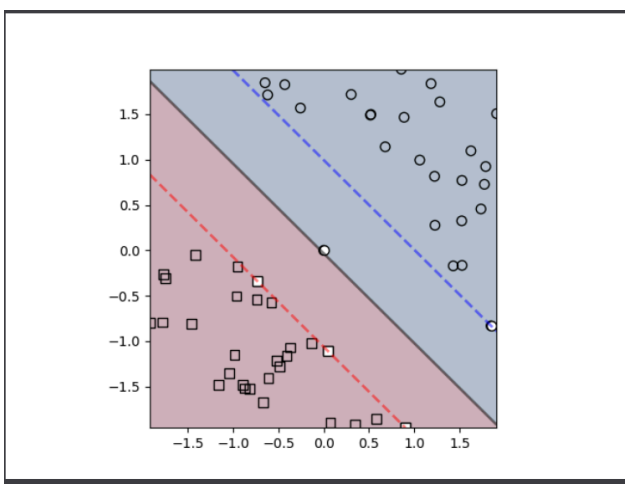
C = 0.01:



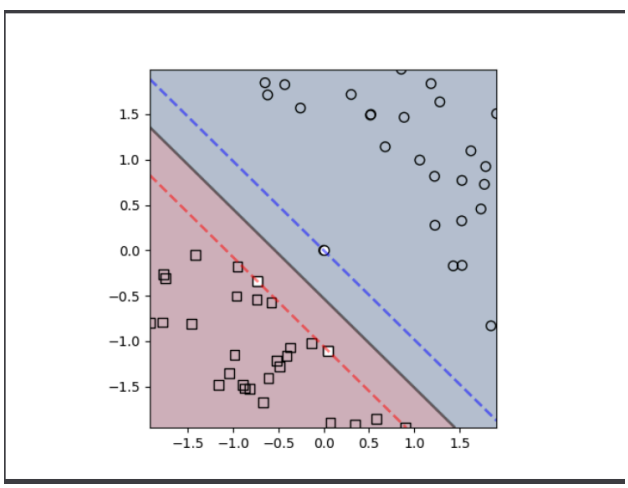
C = 0.1



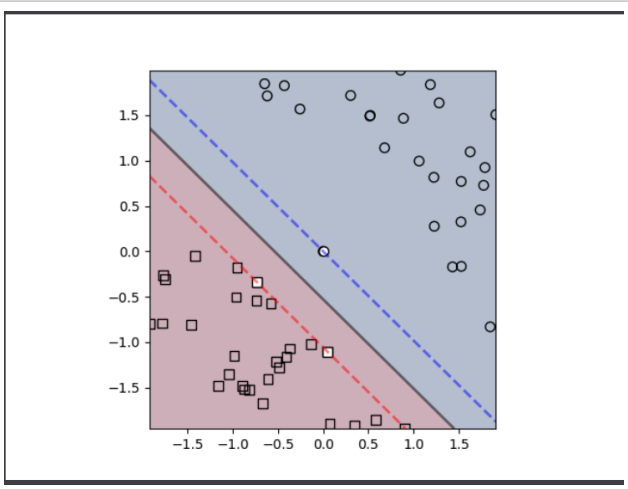
C = 1



C = 10



C = 100



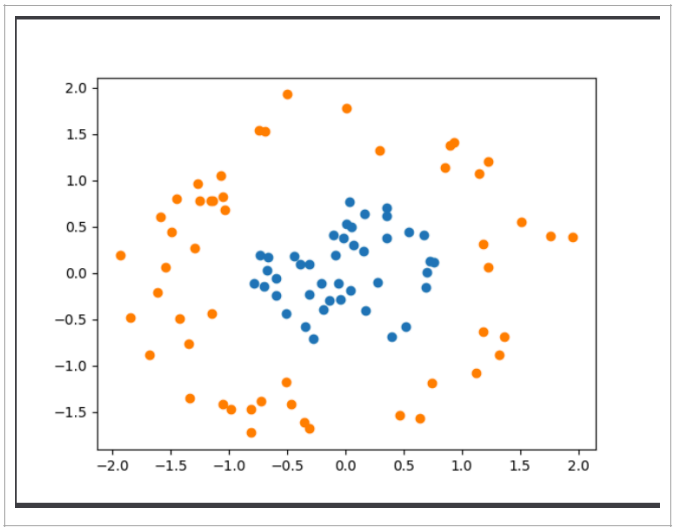
Before SVM implementation, it can be seen from the first figure, our data points are linearly separable so SVM with linear kernel can work with no harm.When we use C = 0.01 and C = 0.1, SVM misclassified a data point which belongs to ‘circle’ label but SVM classified as ‘square’.We can also clearly see that when these C values are applied a lot data points violated the margin.With these small C values, SVM ignored a data point to be misclassified and therefore we can conclude that these two SVMs made soft margin classification because in the hard margin classification there will be no misclassified examples.

Starting from C = 1, SVM classifies data points’ labels correctly and support vectors start to shift towards its closest values to the margin.With C =1, ‘square’ labeled data points’ support vectors are on the closest data points to margin but not ‘circle’ labeled one.When we continue to increase C value to 10 and 100, ‘circle’ labeled data points’ support vector resides its closest values to the

margin and SVM reached its minimum loss value and minimized the distance between support vectors and margin.

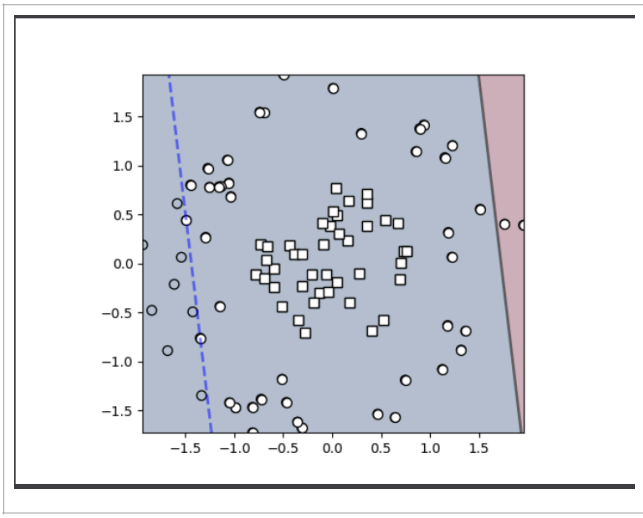
Task 2

Before any implementation is applied, our training data distribution on the graph looks like this:

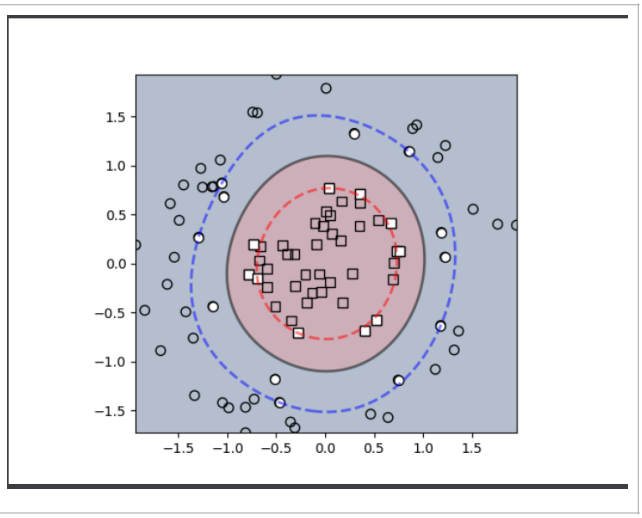


SVM implementation with:

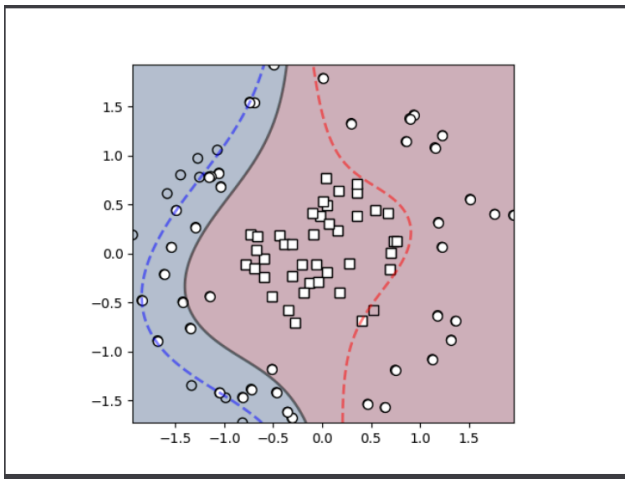
Linear Kernel



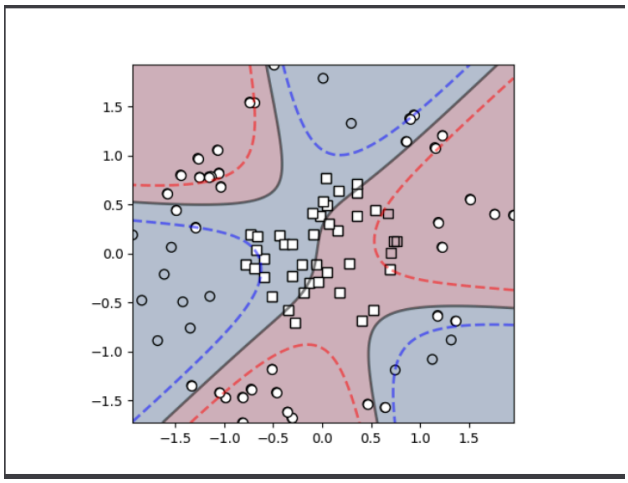
Rbf Kernel



Polynomial Kernel



Sigmoid Kernel



We can see that from the data points distribution, our train set is not linearly separable and we need to carry the set to another dimension using kernel to make linearly separable so that we can apply SVM.Because at this dimension, our data set is not linearly separable, if we use linear kernel, we get a lot of misclassified data point and SVM makes so much sacrificing.With rbf kernel usage, we get all data points correctly classified but some data points are between margin and support vectors which means margin is violated.Again, if polynomial and sigmoid kernels are used, we obtain quite misclassified data points.From this four types of kernels, we can conclude that for this train set, rbf is the most suitable kernel in this experiment and if we increase C value, we can obtain support vectors which are closest data points to the margin so that loss function can be minimized.

Task 3

Kernel Types	C Value	Gamma	Validation Accuracy
Rbf	1	scale	.76
Rbf	1	Auto	.705
Polynomial	1	Scale	.77
Polynomial	1	Auto	.72
Sigmoid	1	scale	.305
Sigmoid	1	Auto	.705
Linear	1	Scale	.62
Linear	1	auto	.62
Rbf	10	Scale	.795
Rbf	10	Auto	.72
Polynomial	10	Scale	.715
Polynomial	10	Auto	.71
Sigmoid	10	Scale	.27
Sigmoid	10	Auto	.695
Linear	10	Scale	.615
Linear	10	Auto	.615
Rbf	0.1	Scale	.76
Rbf	0.1	Auto	.69
Polynomial	0.1	Scale	.755
Polynomial	0.1	Auto	.415
Sigmoid	0.1	Scale	.42
Sigmoid	0.1	Auto	.46
Linear	0.1	Scale	.65
Linear	0.1	Auto	.65
Rbf	0.01	Scale	.42
Polynomial	0.01	Scale	.715
Sigmoid	0.01	Auto	.42
Linear	0.01	Auto	.67

Best accuracy obtained from validation set is 0.795 where parameters for SVM is kernel='rbf', C=10 and gamma='scale'.If we use this hyper parameter configuration on the whole train set and predict test set labels, we get 1.0 accuracy value on test set.

Task 4

Test accuracy using rbf kernel and C = 1 is 0.95.Accuracy is a useful performance metric but when it is used with balanced data.If we use it in an imbalanced data set like our sample data set, it becomes useless.Imagine we made no implementation properly just predict the majority class' label, we will get a high accuracy value but it does not mean our algorithm works well.

Confusion Matrix

Predicted \ Actual	Positive	Negative
Positive	**950**	**50**
Negative	**0**	**0**

After oversampling, test accuracy is 0.951.

Predicted \ Actual	Positive	Negative
Positive	**939**	**38**
Negative	**11**	**12**

After undersampling, test accuracy is 0.764.

Actual

Predicted	**Positive**	**Negative**
Positive	**728**	**14**
Negative	**222**	**36**

After class_weight parameter arranged as ‘balanced’, test accuracy is 0.936.

Actual

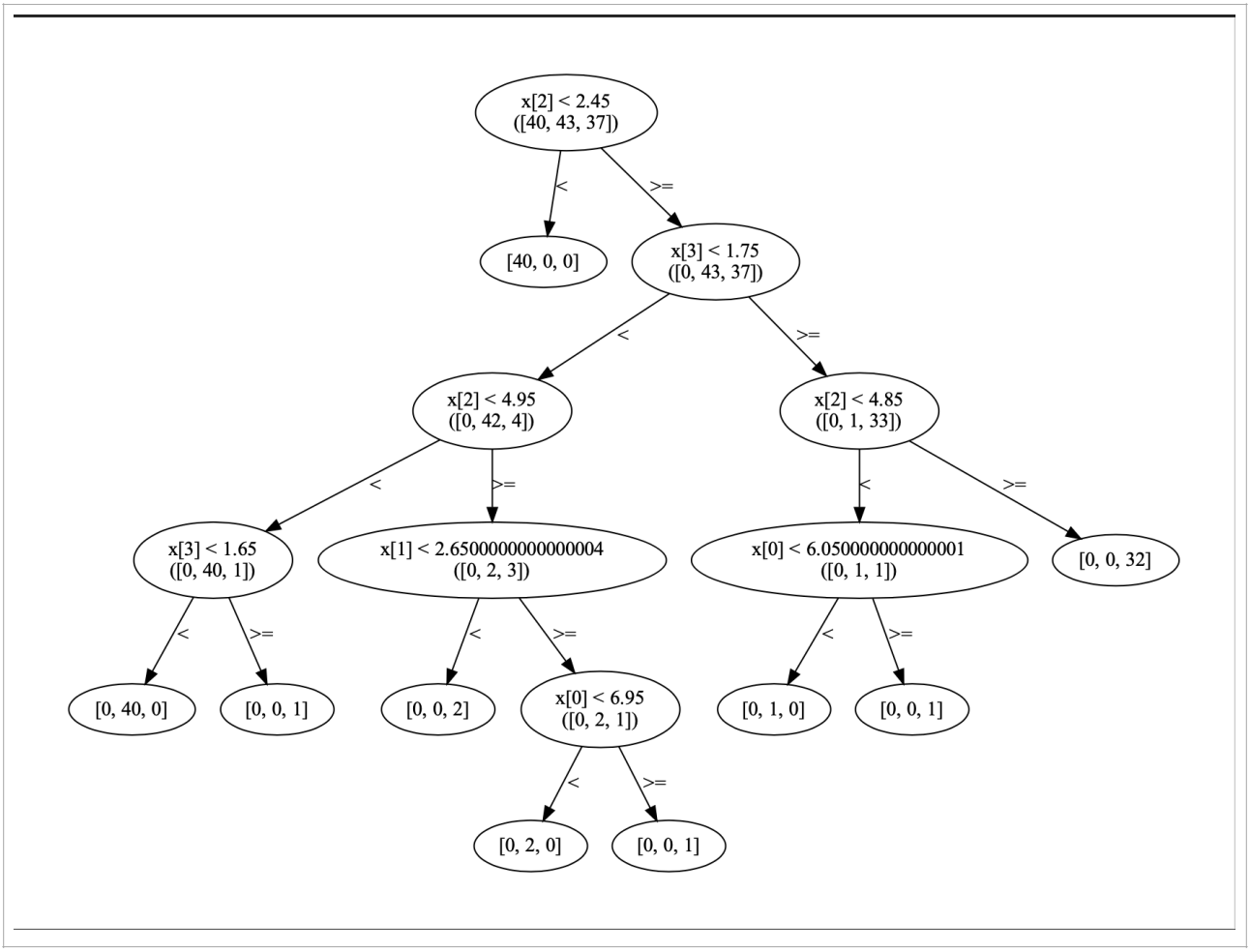
Predicted	**Positive**	**Negative**
Positive	**920**	**34**
Negative	**30**	**16**

Decision Trees

If we use train set and train labels to draw a decision tree and when information gain is used, tree visualizations look like:

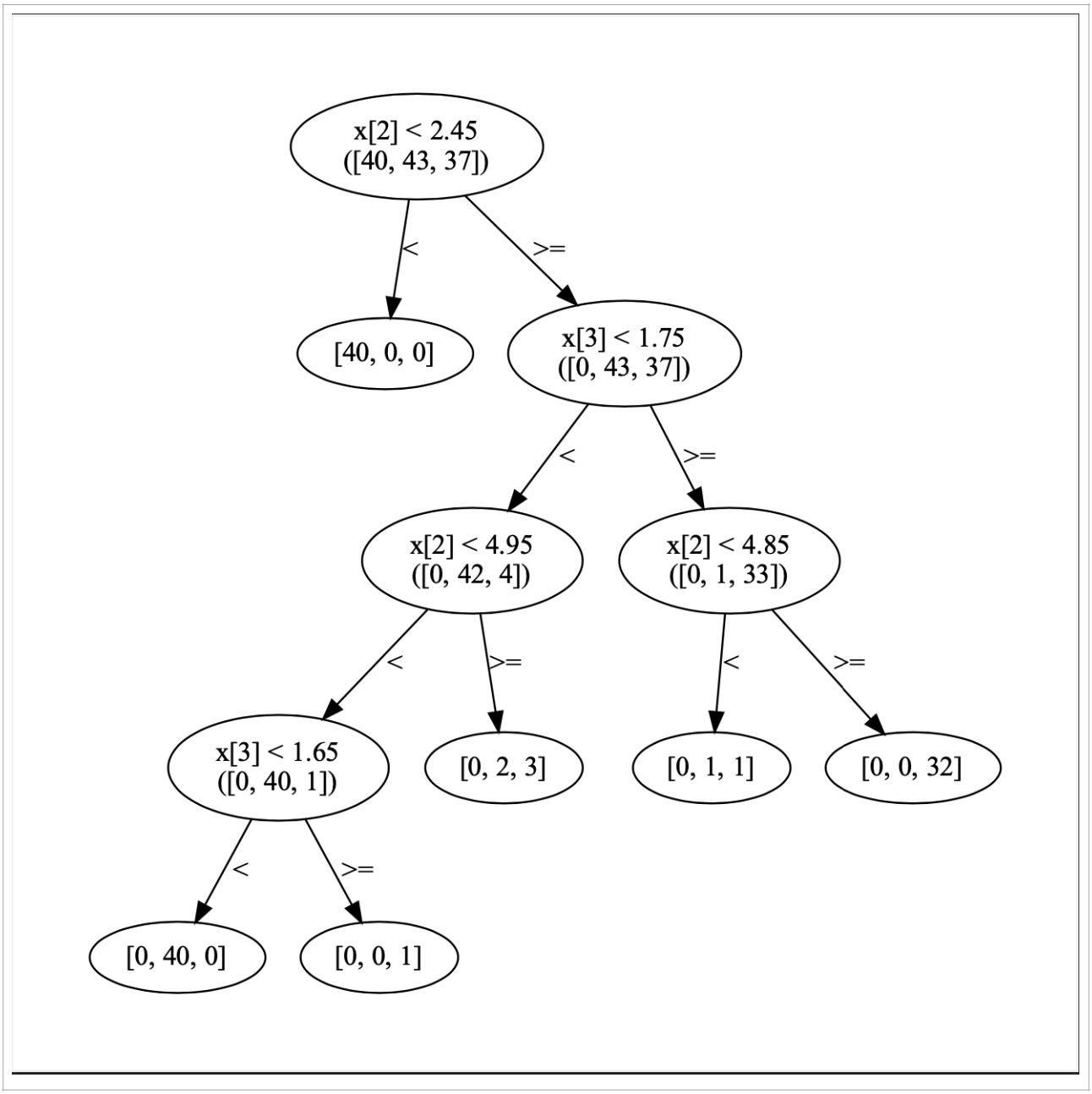
Without prepruning:

Test accuracy is 93.33 in percentage.



With prepruning:

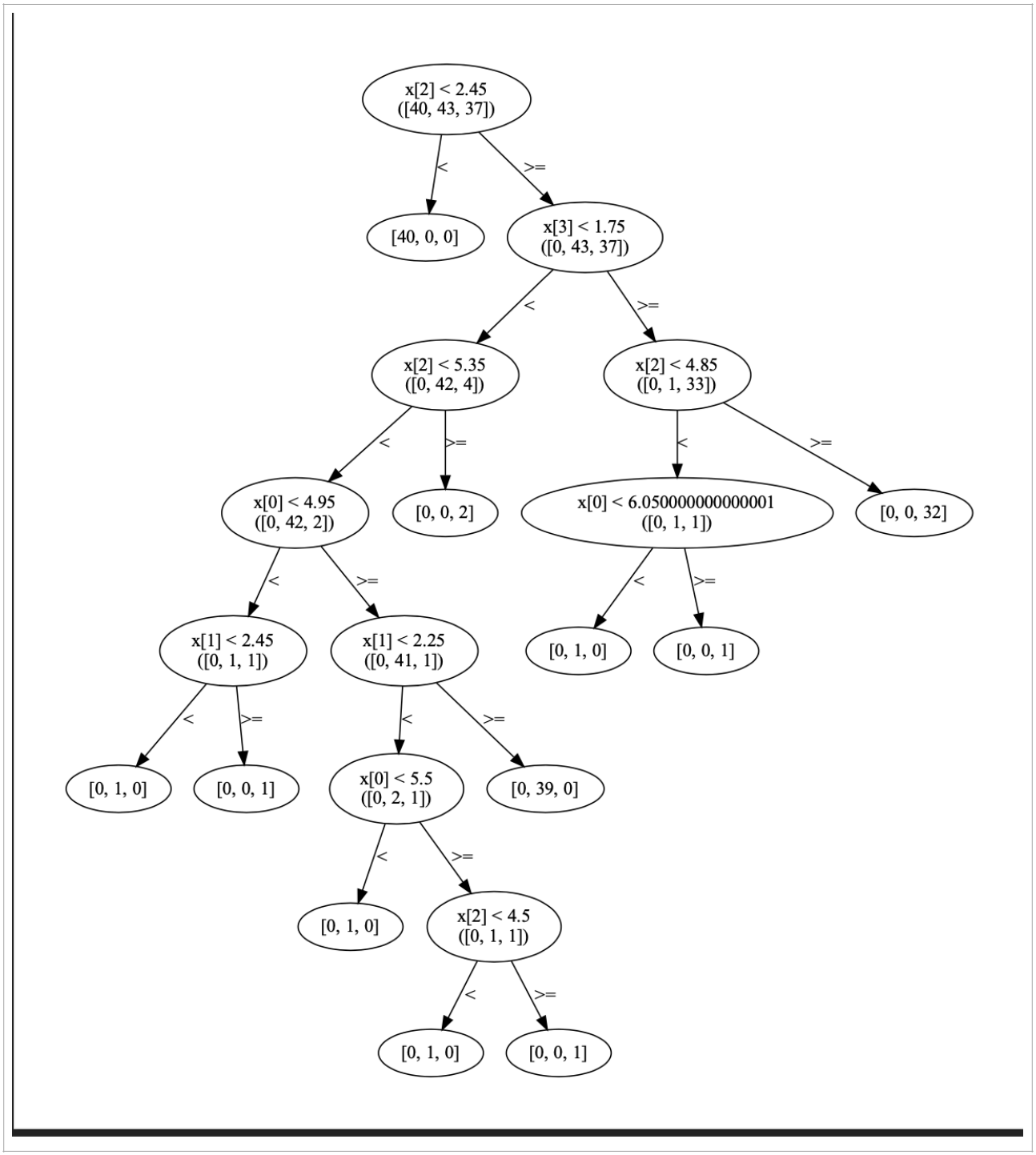
Test accuracy is 96.67 in percentage.



If we use train set and train labels to draw a decision tree and when average Gini index is used, tree visualizations look like:

Test accuracy is 90 in percentage.

Without prepruning:



With prepruning:

Test accuracy is 93.33 in percentage.