# Data Mining & Machine Learning F20DL

https://github.com/friedforfun/Data-Mining-Machine-Learning-CW

Group 4
Lewis Wilson, Sam Fay-Hunt, Kamil Szymczak, Chun Man

November 2020

# Contents

# 1 File management, data pre-processing, transformation and selection

## 1.1 File Management

To facilitate collaboration and reduce repetition we developed a Scripts module (See Figure A.2) containing the majority of our code for this coursework. These functions provide utility for loading data, pre-processing data, building models and other convenience tasks. We annotated all functions within the Scripts module with docstrings, and compiled them with Sphinx, a final version of the documentation can be found under '*docs.rar*' in the project root folder.

We used Jupyter notebooks for all the tasks, primarily as a testing workspace and a medium to present our final work.

## 1.2 Data pre-processing

### 1.2.1 Data transformation

**Downsampling**
We used local-mean downscaling (*Scripts/downsampling.py*) to try and expose low-level features (Appendix A.3). We frequently used downscaling throughout the project to reduce the image resolution by averaging 4 pixels into 1. We also used rescaling with aliasing to reduce the image to 12x12 pixels, to aid in visualising patterns in the pixel greyscale values (Appendix A.10 & A.11).

**Binning**
By implementing equal width binning we were able to greatly reduce the cardinality of the greyscale values from 256 to 8. This helped offload a considerable amount of computation when calculating the edges in the complex Bayes Networks, and offered a small accuracy uplift for the Naive Bayes classifier (see Section 2).

### 1.2.2 Data selection

**Balancing the class distribution**
Having observed frequent issues with overfitting due to a substantial imbalance in the class distributions (particularly with the binary classification labels) we utilised the sample method from the Pandas library along side the random_state argument to produce replicable datasets with a balanced number of each distinct class.

**Sampling data**
We primarily used the train_test_split function from the SKLearn library because it provides excellent utility for splitting apart the data with parameters to aid in discretizing, replicating, and resizing the data. We also made use of Pandas and Numpy shuffle methods when most convenient.

**Pixel Selection**
(Figure A.5) We produced pairwise correlations between each pixels value and the class label, then using this data we produced an ordered list of pixel indices for each label file. We used this data to produce heatmaps showing the importance of each pixel (See Appendix A.4), where the darkest pixels are deemed the most important for prediction and the lightest the least. We used the pixel ordering data to build and

score 2304 Naive Bayes classifiers for each class of labels, each plot on the x-axis is using all preceding pixels for classification see Figure in Appendix A.5.

## 2 Naive Bayesian Networks

We used the SKLearn library's Naive Bayes modules to build the Naive Bayes classifiers, we built 88 classifiers so we could compare and test the various pre-processing configurations we had developed. First we plotted the accuracy of the different configurations in bar charts to provide a high-level perspective on the accuracy, and then we built confusion matrices gain detailed insight into how the classifier's performance varied.

Observations:

1. Using no pre-processing technique with Naive Bayes had a significant negative impact on the quality of the predictions.
2. Downscaling on its own provides no observable benefit to the accuracy of Naive Bayes.
3. Naive Bayes performs much better when it is only trained to perform binary classifications
4. Naive Bayes has serious problems with overfitting when the distribution of classes in the dataset is poorly balanced.
5. Under the right conditions Naive Bayes can make excellent predictions, we observed prediction accuracy as high as 88% with our test data.

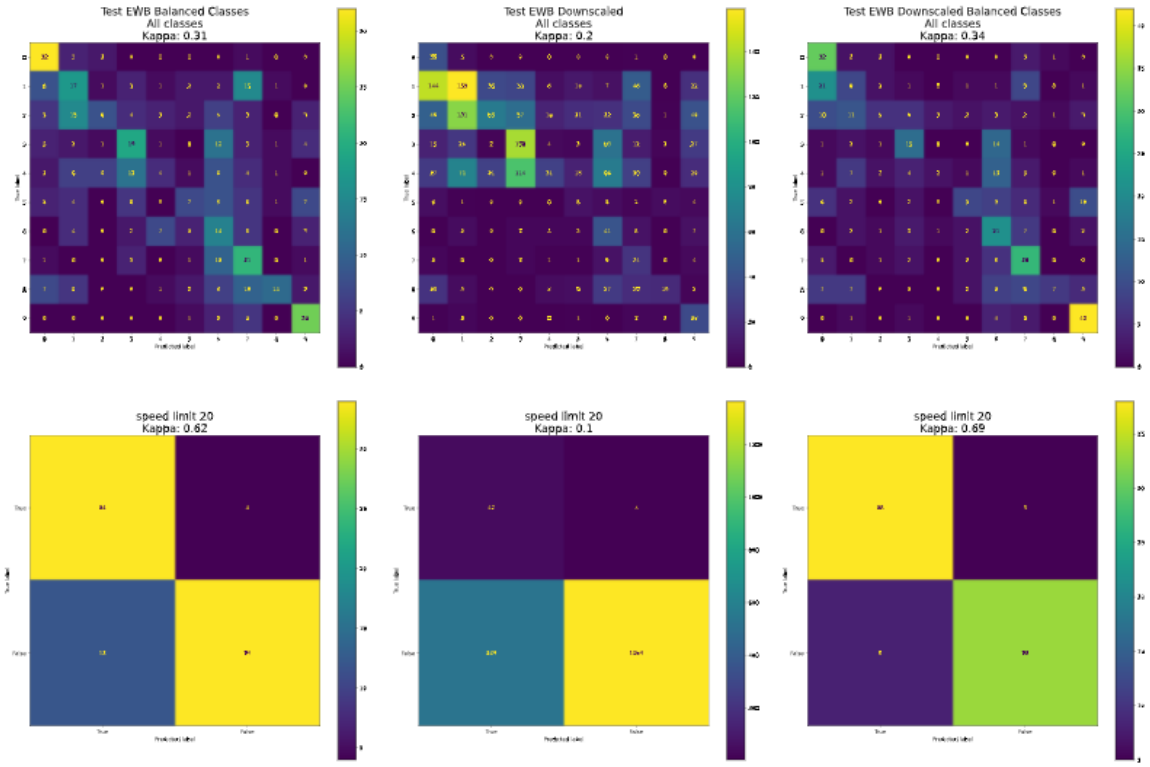



Figure 1: Naive Bayes confusion matrix showing Equal Width Binning

# 3 Complex Bayes nets

## 3.1 Building Bayes Networks

Bayes networks represent probabilistic directed acyclic graphs that define the relationships between conditional dependencies and random variables. A naive Bayesian network can be represented in a Bayes network where the node representing the probability distribution of the class is the only parent of all other nodes and no other edges exist in the network. By adding additional edges (so long as the graph remains acyclic) we can represent causal relations between random variables.

Using the pgmpy library we build Bayes Networks using K2 scoring and HillClimbing to infer the edges of the network. Next we calculated the model parameters using pgmpy's built-in MaximumLikelihood-Estimator, we also tried this with a BaysianEstimator using K2, and Dirichlet scoring mechanisms. Attempting to build the network both ways gave us solid insights into the problems that would have to be solved to produce a Bayes Network. We immediately encountered 2 computational complexity problems:

1. Learning the optimal edges.
2. Building all the model parameters

We handled the first problem by discretizing the greyscale values using equal-width binning(see section 1.2) and using the data acquired from Task 5 to select a small number of columns to build the networks. When using Weka to compute Bayesian networks we observed that Weka would perform extremely aggressive binning of the greyscale values often discretizing down to only 2 bins. This had a profound effect on the speed of learning the parameters and edges.

## 3.2 Algorithms & Data

We have noticed that useADTree parameter doesn't make a difference therefore we set it to True for all the experiments done in Weka.

Initializing as Naive Bayes gives the same accuracy and confusion matrix but as shown in the graph (Appendix A.21 & Appendix A.22) all pixels depend on the Label whereas with NB set to False, some pixels are not dependent and not connected as well as pixels are dependent on other pixels.

In the confusion matrix (Appendix Probability Contribution Table A.23) we can see pixel 1409 depends on the label, pixel 1362 and pixel 982. Due to using binning, Weka binned them into 2 binns one contains pixels between white and light grey and other bin has everything between grey and all the way to black. By knowing this bin distribution we can understand what impact combination of colors has on the outcome.

There wasn't a big improvement on the accuracy of confusion matrix between K2, TAN and Hill Climbing.But from insights into the tree visualizations the max number of parents set to 3 is better than 1 as more pixels depend on other pixels which are needed to register features correctly.

# 4 Clustering

## 4.1 Identifying clusters

We used the principle component analysis module from SKLearn to help plot the data as a scatter graph, this made it very clear we had a problem with how our data was distributed for clustering see Appendix A.9, this resulted in apparently meaningless clusters. To resolve this we tried many permutations of selection techniques, discretizations, and transformations. We considered that the data must be clustering on features that were otherwise transparent to us, as a result we tried looking for patterns in the clustering behaviour such as looking at how the clustering algorithm grouped data by high-level features such as the shape of the signs. We used KMeans clustering with the EM and Elkan algorithms to compare the resulting clusters, however they were mostly identical Elkan converged considerably faster then EM. Because we observed an issue with the desired class features not usefully dividing the data within the clustering dimensions we decided to use transfer learning to try and extract more useful feature vectors. We used Keras to download VGG16 with weights trained on imagenet, by removing the (top) fully connected layers past the convolutional and pooling layers we were able to produce feature vectors for each image. Using these feature vectors with principle component analysis we were finally able to produce some clearly divided clusters for the binary classification labels. We also noted that this resulted in clusters that were split by the shape of the signs (triangle or square).

Clustering by sign shape

(Figure A.12) We clustered the data into 2 clusters with the hope that it would differentiate the sign's shape, one cluster for triangular the other for rectangular signs. One cluster got more images of a particular shape than the other but the difference was very small. We performed clustering again using our best pixel selection algorithm (Figure A.13) but now with each image only containing the 250 best pixels, the clusters did a better job as they contained a lot more signs of one shape than the other. Thus we reaffirmed our knowledge from the task 4 heatmap stating that apart from digits on signs the shape is the second most important feature.

From our cluster graphs (Figure A.16 & Figure A.17) helped to visualize the connection between signs and clusters for the binary outcome datasets. We observed that triangular signs were indeed only in one cluster thus has 100% accuracy whereas the circular signs were distributed in both clusters, so we had high recall for triangular sign. In conclusion we observed that more intelligent feature extraction results in better clustering.

# Appendices

## A Appendix A

### A.1 Workload split

| Team member | Involvement |
|---|---|
| Lewis Wilson | I contributed throughout the project helping with the individual script files and notebooks. I created the visualisations for task 5 showing how the numbers of pixel affected the accuracy of classification. I displayed this in two ways showing for all 2304 pixels a line chart of the accuracy for every amount. Then displaying the accuracy results in a bar chart for the best pixel amount per class compared to the recommended 5, 10, 20-pixel amounts. I helped put together the report and proofread it prior to submission. |
| Chun Man | Created visualisations for average greyscale values for finding out about locality and correlations between pixels, contributed to the report. |
| Sam Fay-Hunt | Wrote the naive bayse gaussian scripts, organised Task 3 notebook for submission, wrote the scripts to display the confusion matrices, built the bayse networks, wrote Task 7 notebook, wrote downsampling.py, many of the scripts to plot the bar charts, most of the clustering scripts, half of the clustering Notebooks for tasks 9 & 11 (PCA and feature extraction), wrote a few assistance functions for Task 5, most of the doc strings, helped edit the report, and compiled Sphinx. |
| Kamil Szymczak | Contributed to the development of finding the most important pixels module and produced algorithms to generate heatmaps. Contributed to clustering as well as produced Bayesian Network Architectures in Weka and wrote a conversion script which produces weka file to only contain desired pixels. Contributed with report . |

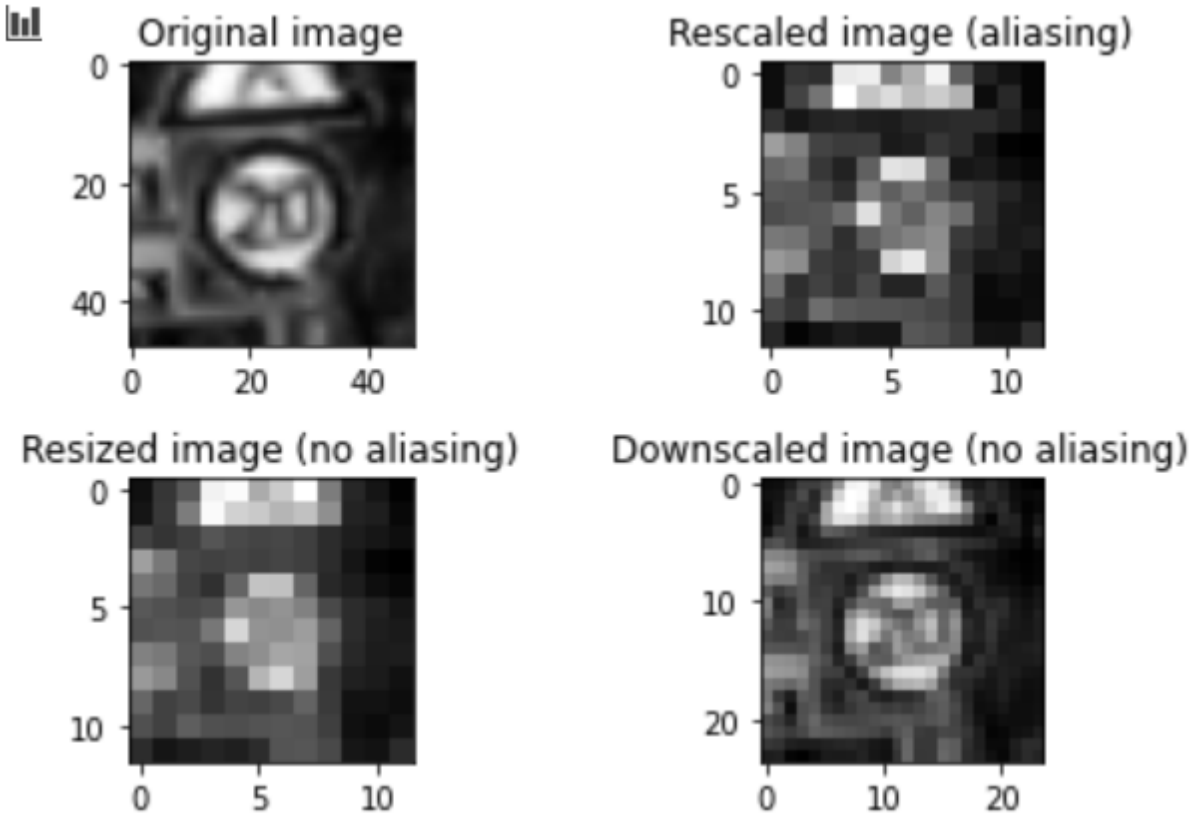
As a team we are happy with everyone's contributions to the project. All team members were punctional and showed up to all scheduled meetings. Sam took the lead as project manager throughout the project delgating the workload and providing support to others.

## A.2 Module Table

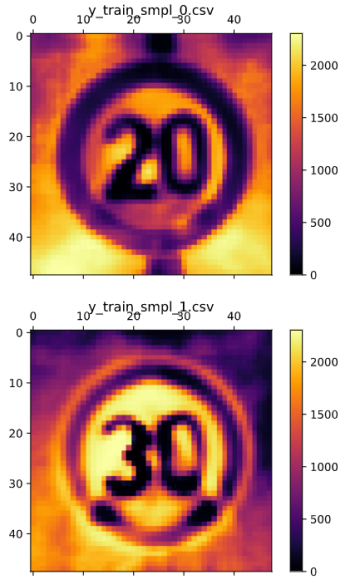
The following are located in the Scripts folder

| Module Name | Description |
|---|---|
| helperfn.py | Provides functions to load and transform with all datasets required. |
| downsample.py | Provides functions to downsample images |
| pixelFinder.py | Provides functions to find the most important pixels within a dataset of a chosen street sign. |
| bayseNet.py | Provides functions used for getting a score for a model by testing all test data against labels. |
| confusionMatrix.py | Provides functions for building and displaying confusion matrices as well as methods for calculating kappa values. |
| plotScripts.py | Provides functions for plotting data into graphs |
| wekaConversion.py | Provides functions to convert preprocessed data to be consumable by Weka |
| NaiveBayes/* | Contains modules for NB Gaussian and NB Categorical |
| clustering.py | Provides functions for getting information for clusters |

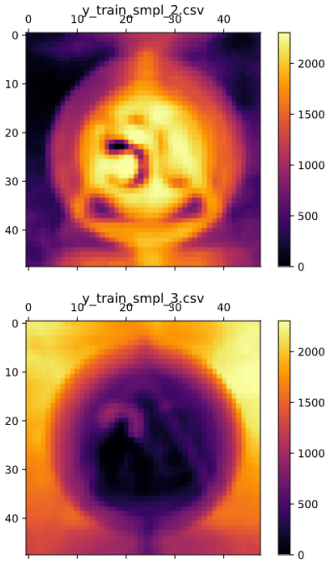
## A.3 Downsampling example



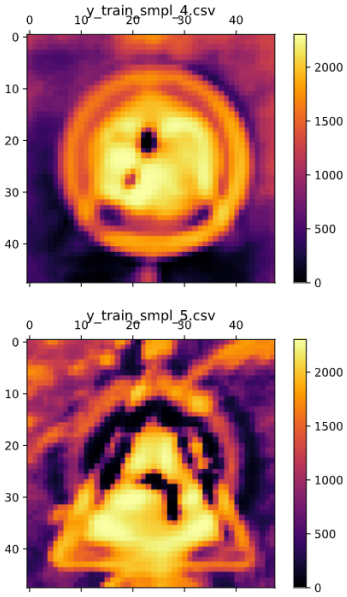
## A.4 Heatmaps



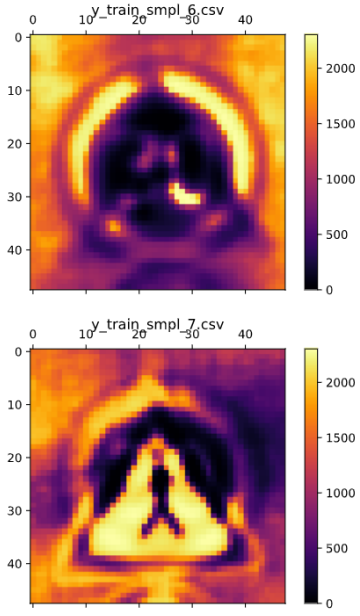

(a) 20 mph & 30 mph



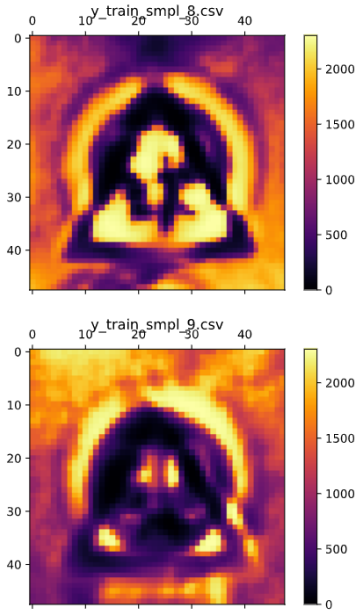

(b) 50 mph & 60 mph




(c) 70 mph & Left turn

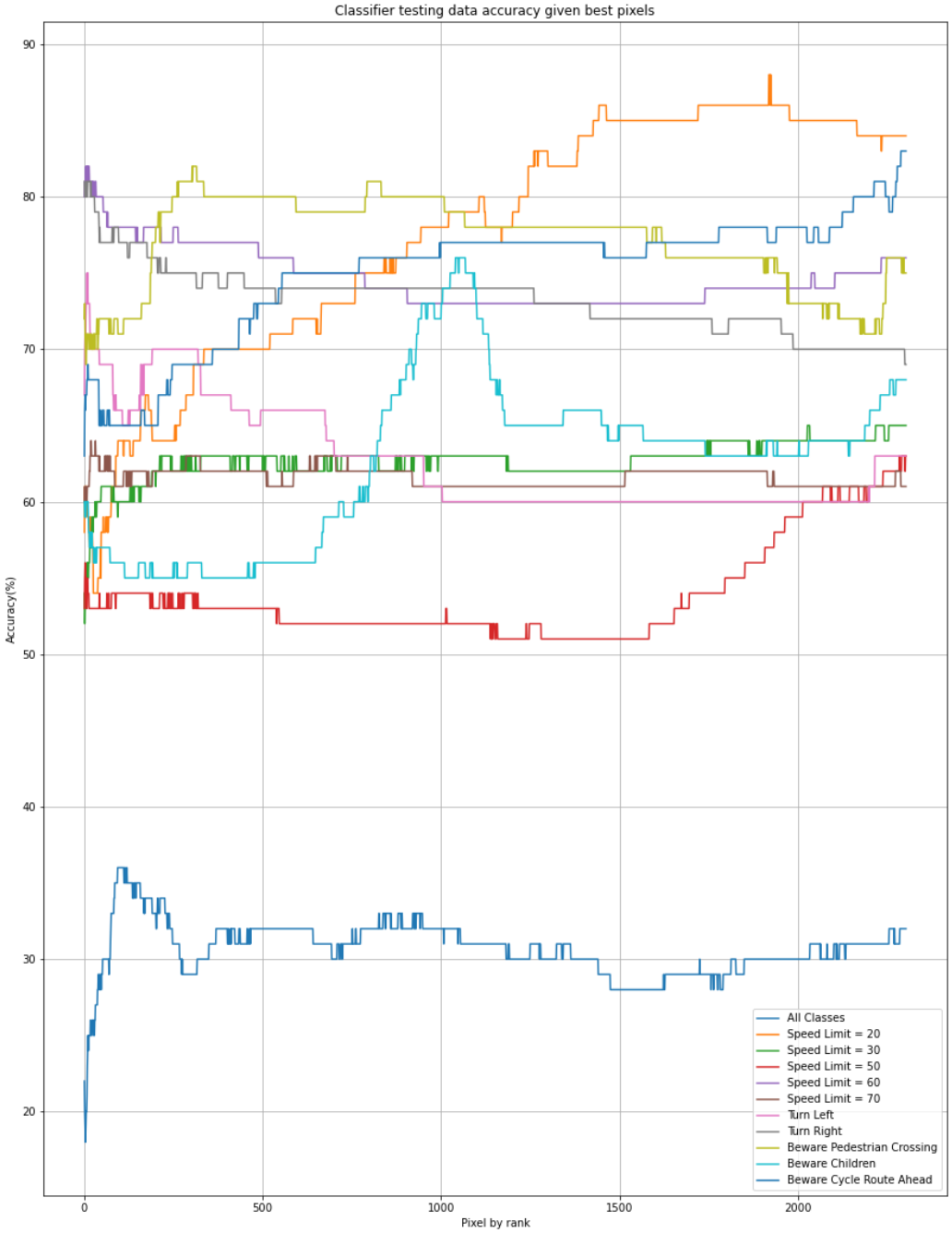



(d) Right turn & Beware pedestrian crossing

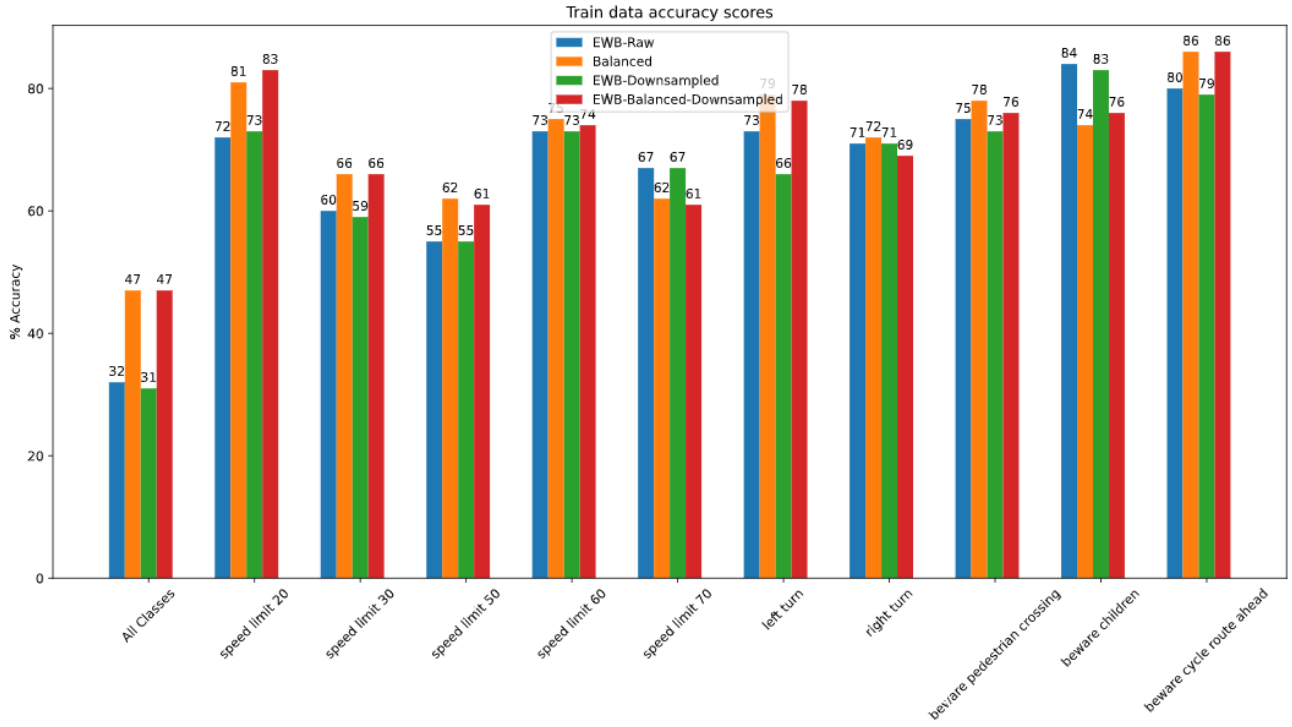



(e) Beware children & Beware cycle route ahead

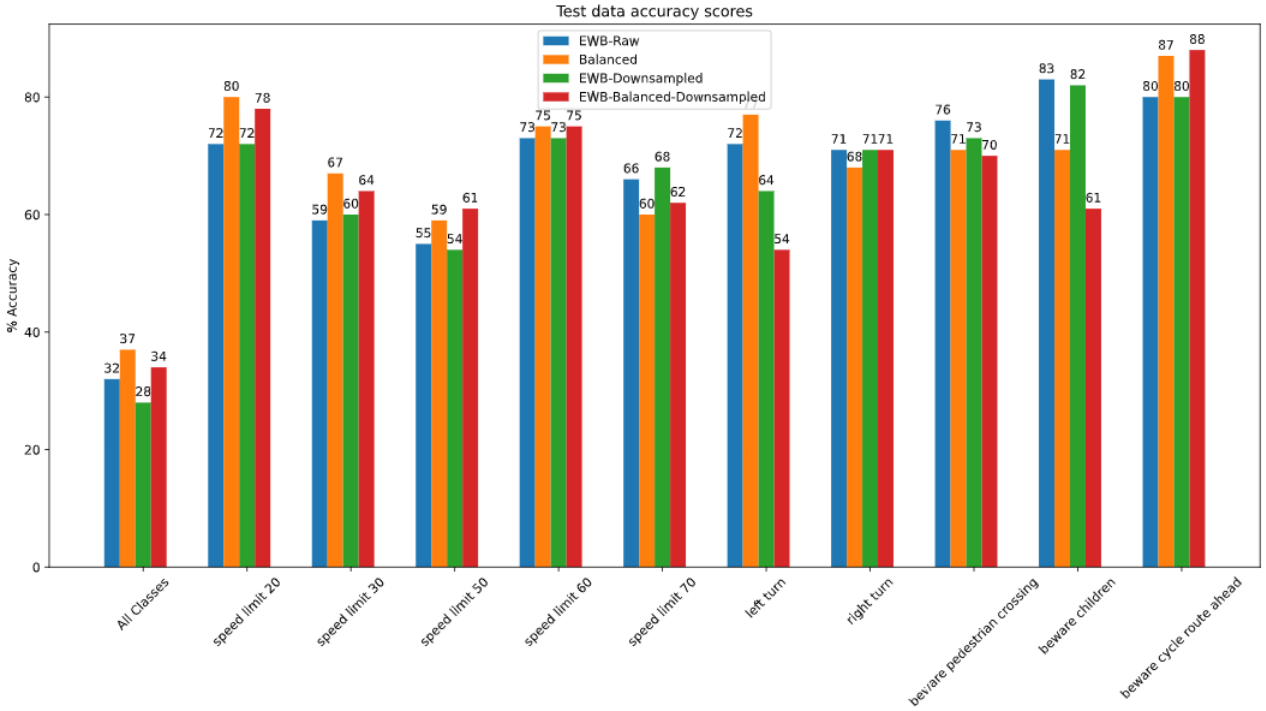
## A.5 Accuracy of classifiers built using the first n pixels for each class



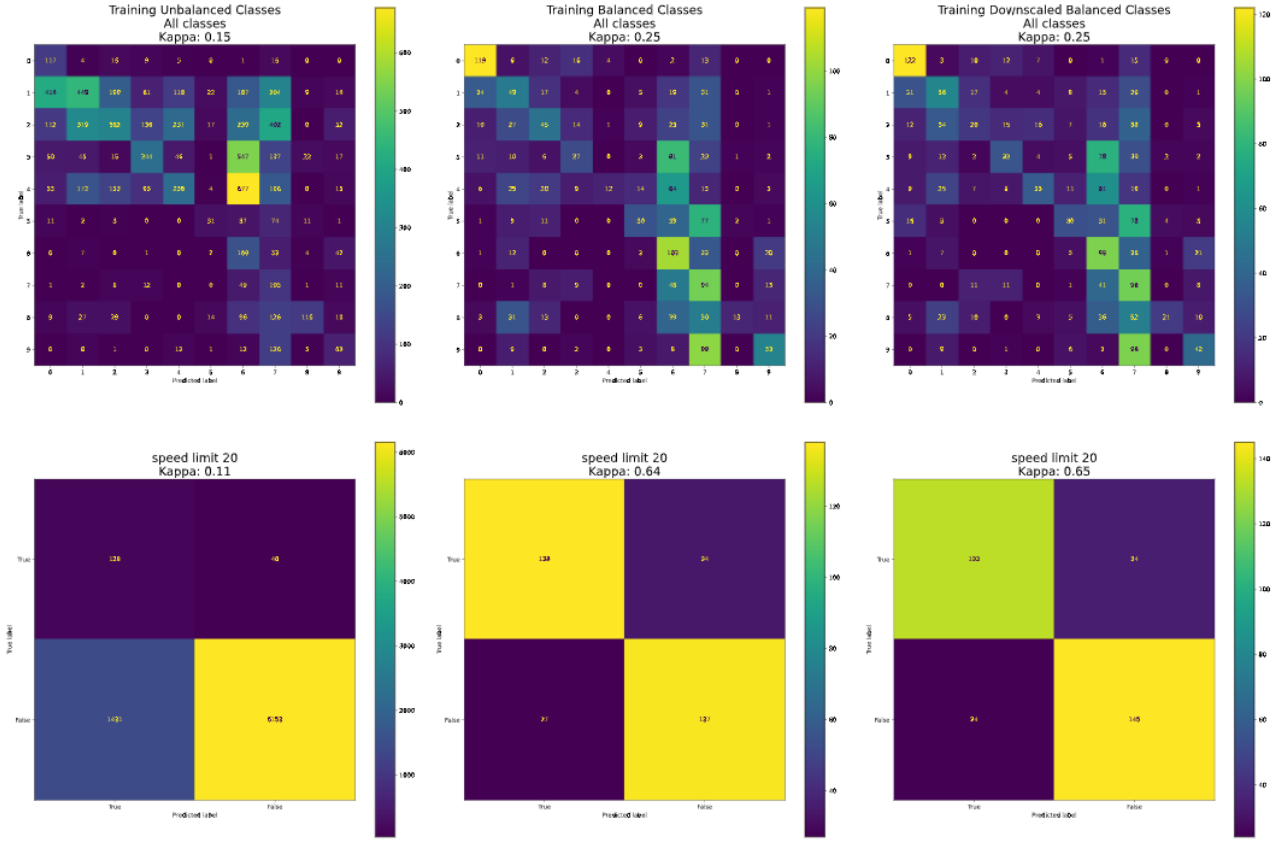
## A.6 Accuracy of Equal Width Binning on Train data



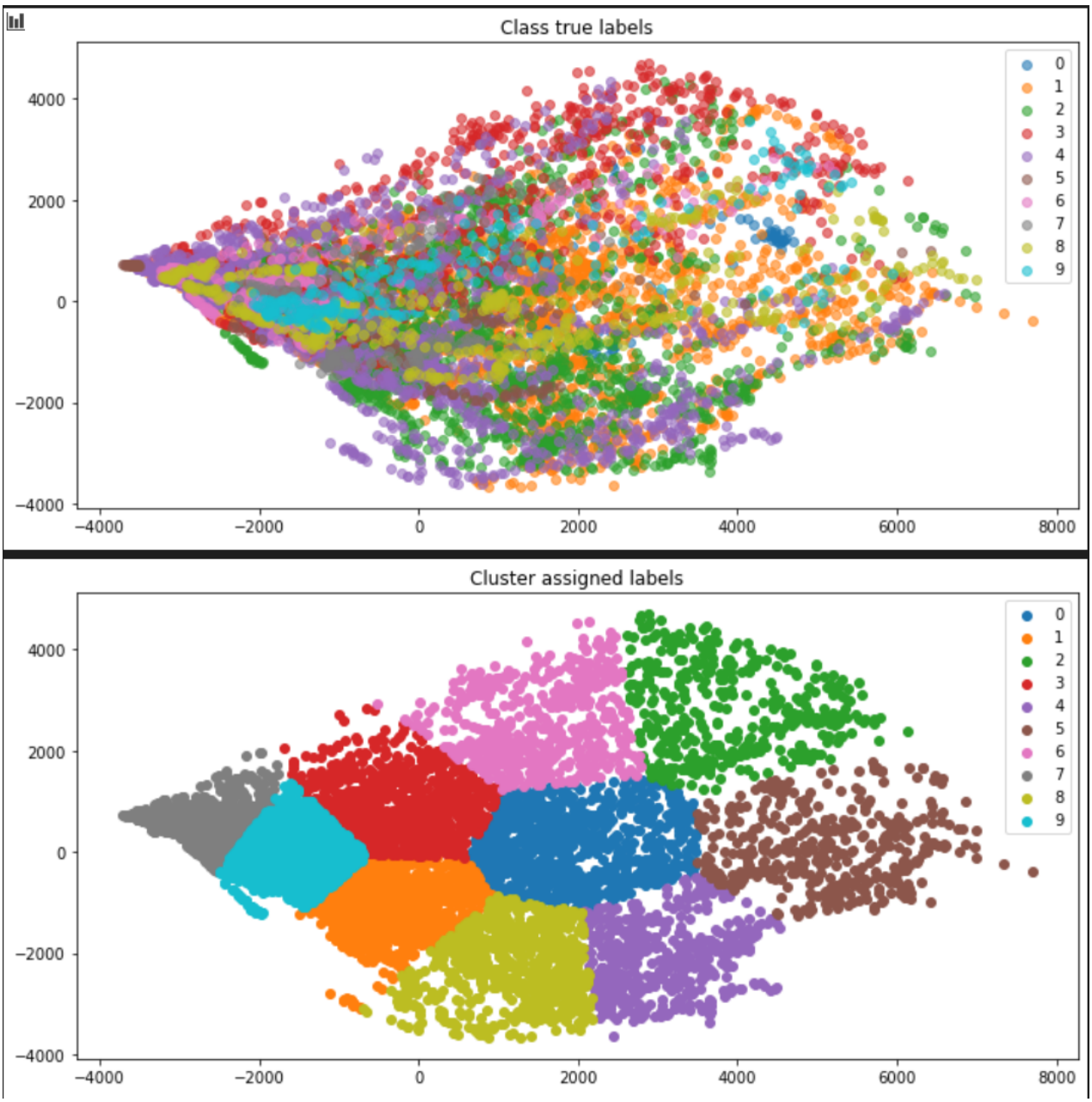

## A.7 Accuracy of Equal Width Binning on Test data



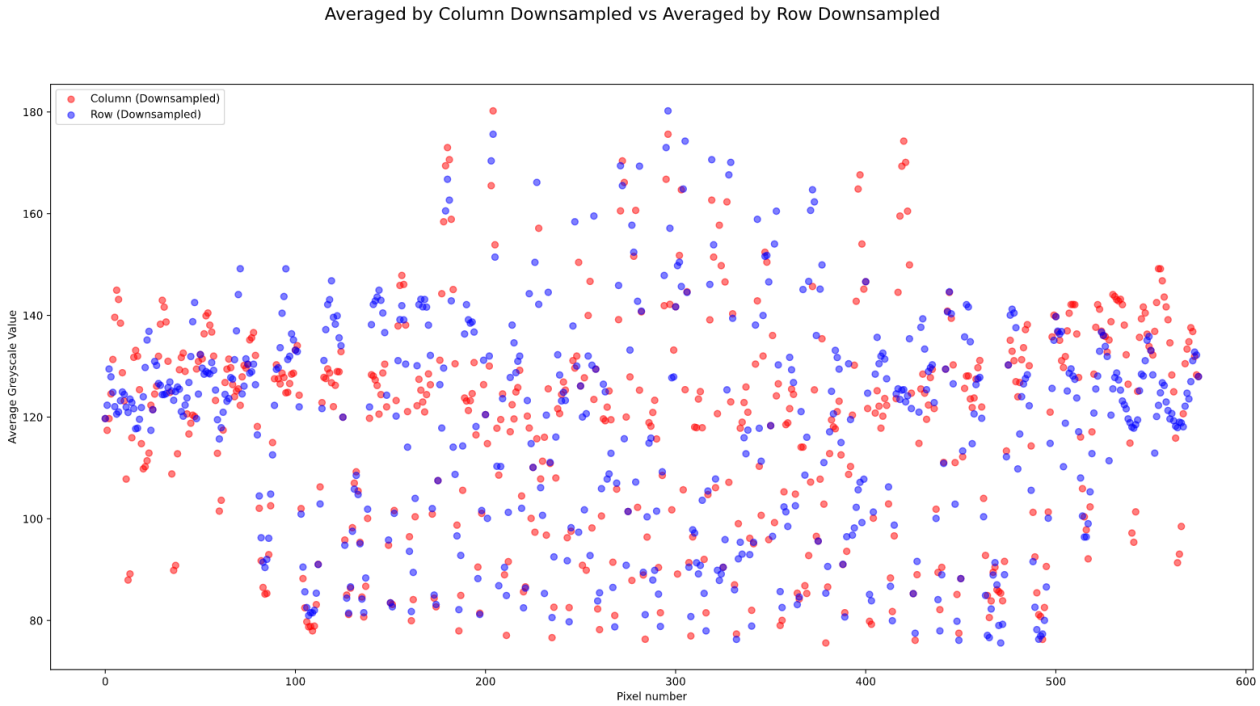
## A.8 Training confusion matrices for Naive Bayes



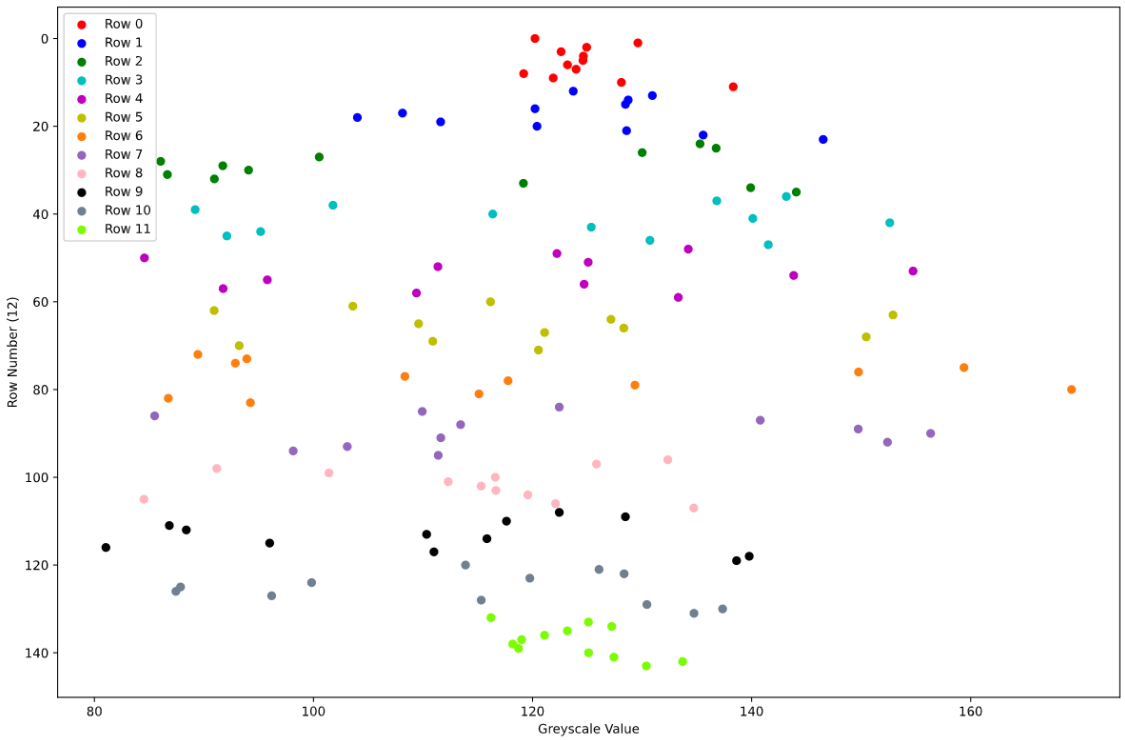
### A.9 No clusters of classes



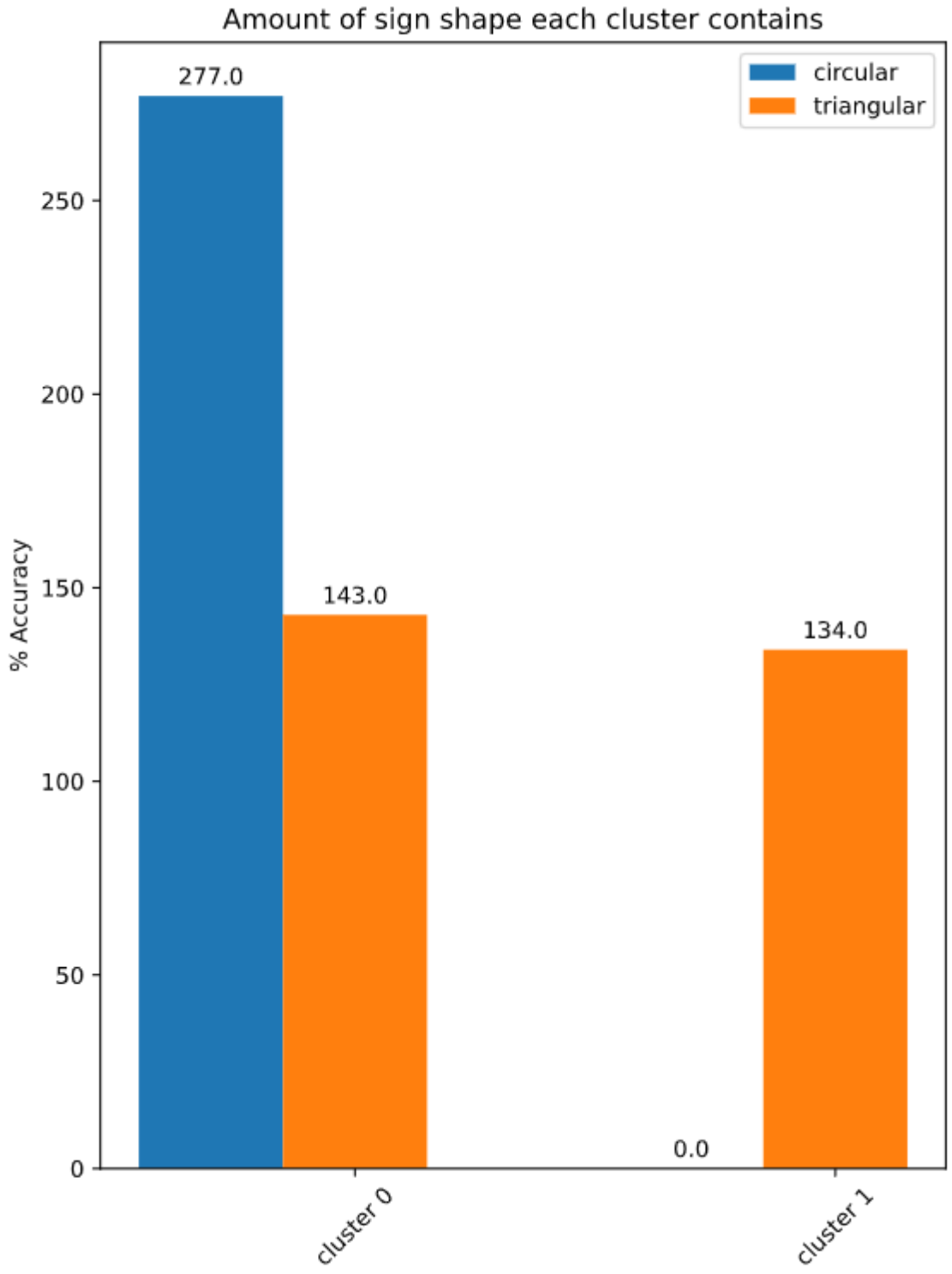
## A.10 Averaged by Column Downsampled vs Average by Row Downsampled



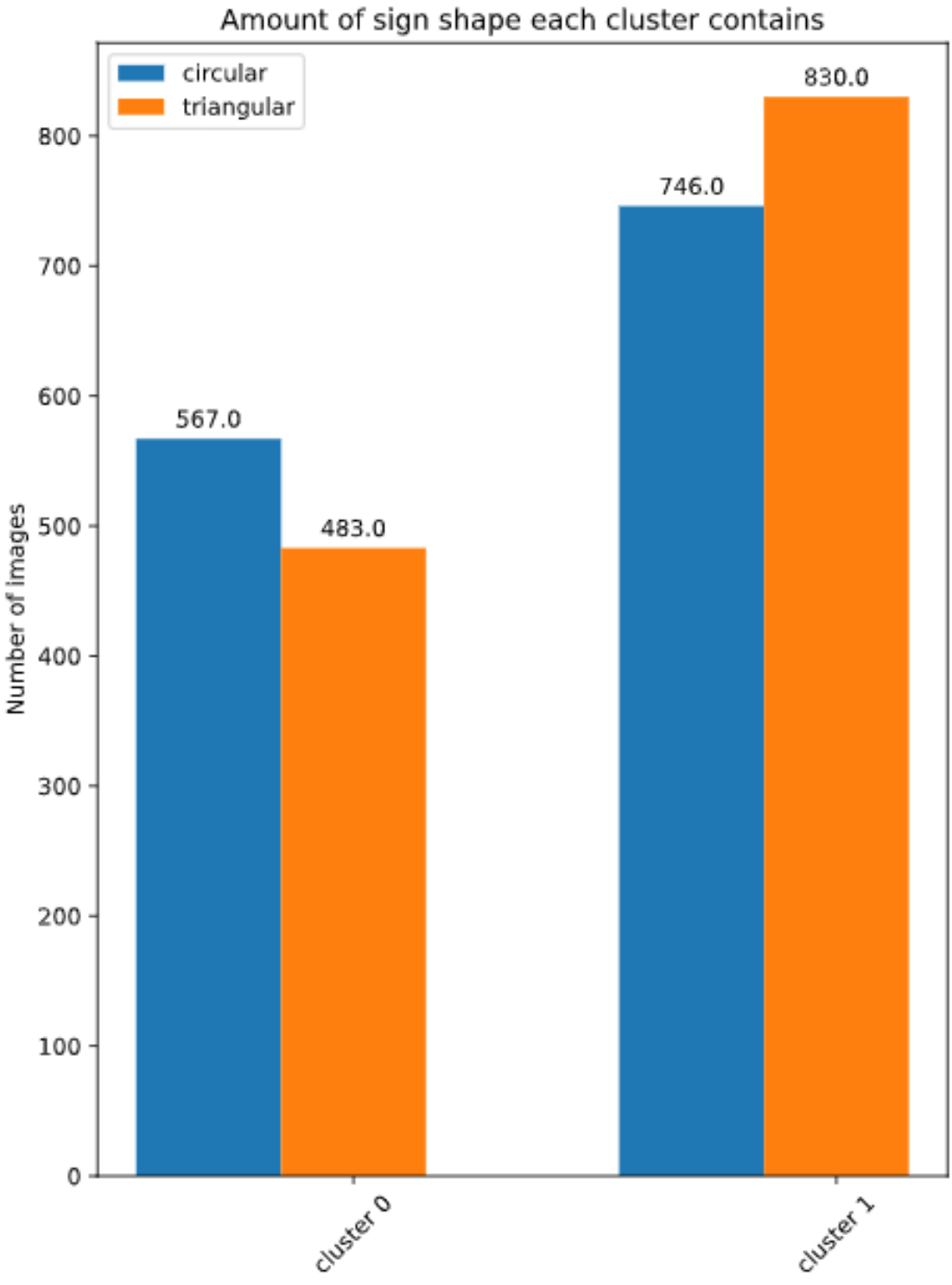

## A.11 12x12 Downsampled image



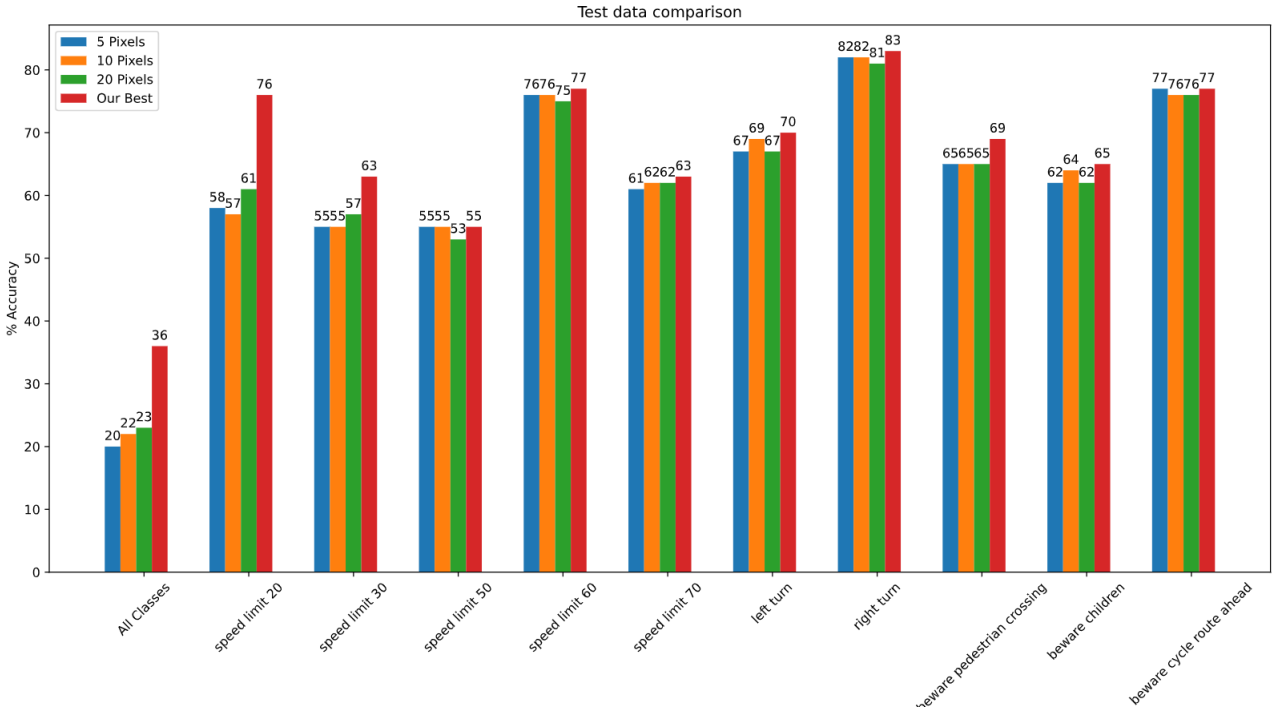
### A.12 Triangular and Circular sign clusters



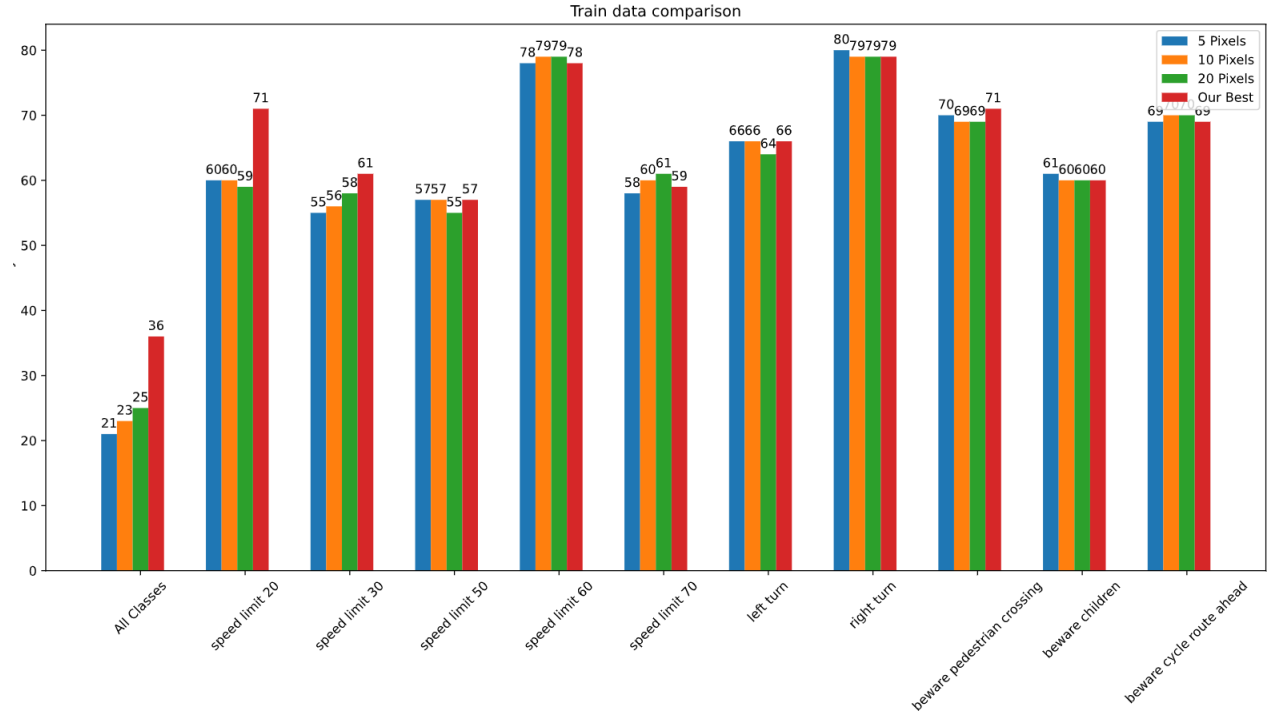
### A.13 Clustering by shape



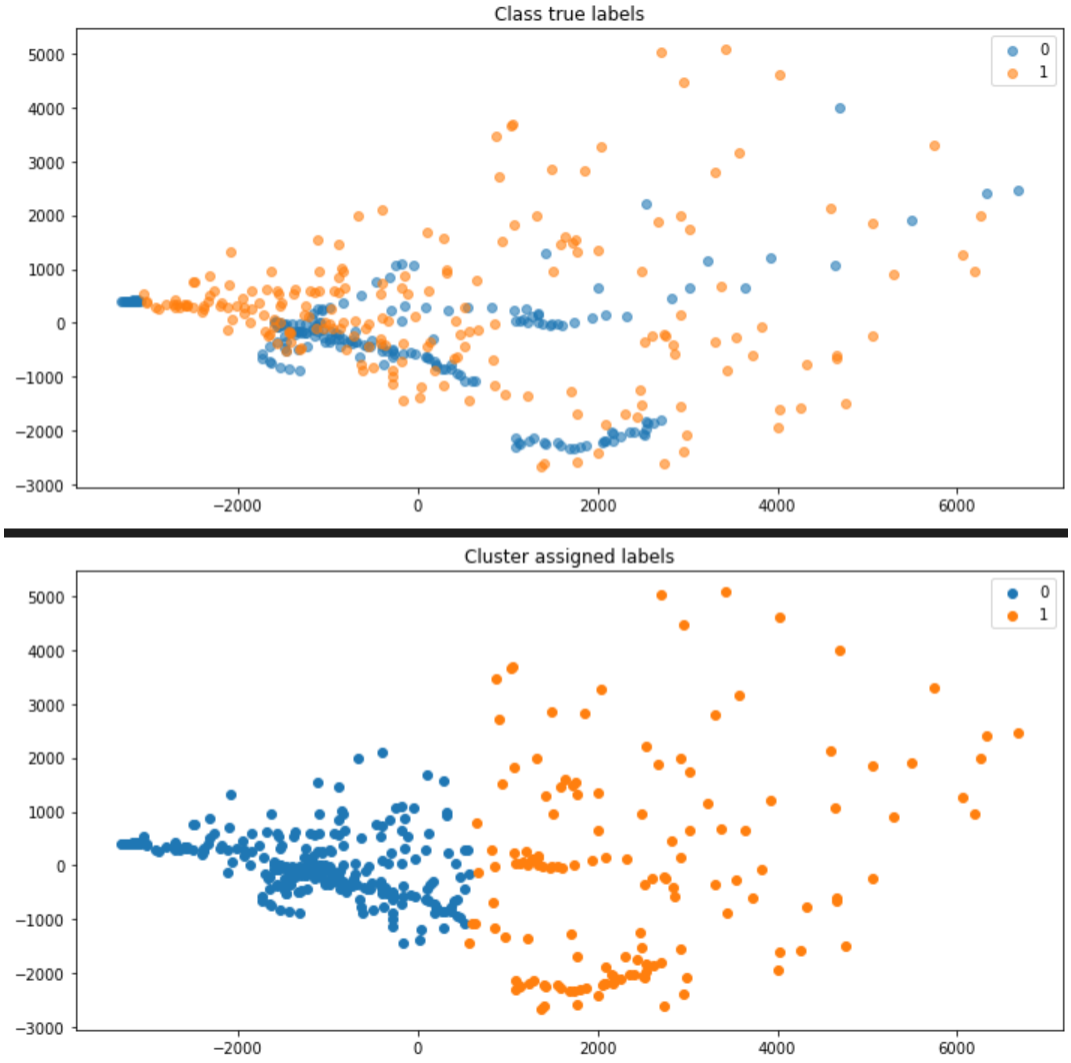
## A.14 Test data classification accuracy with the best selection of pixels from 500 pixels



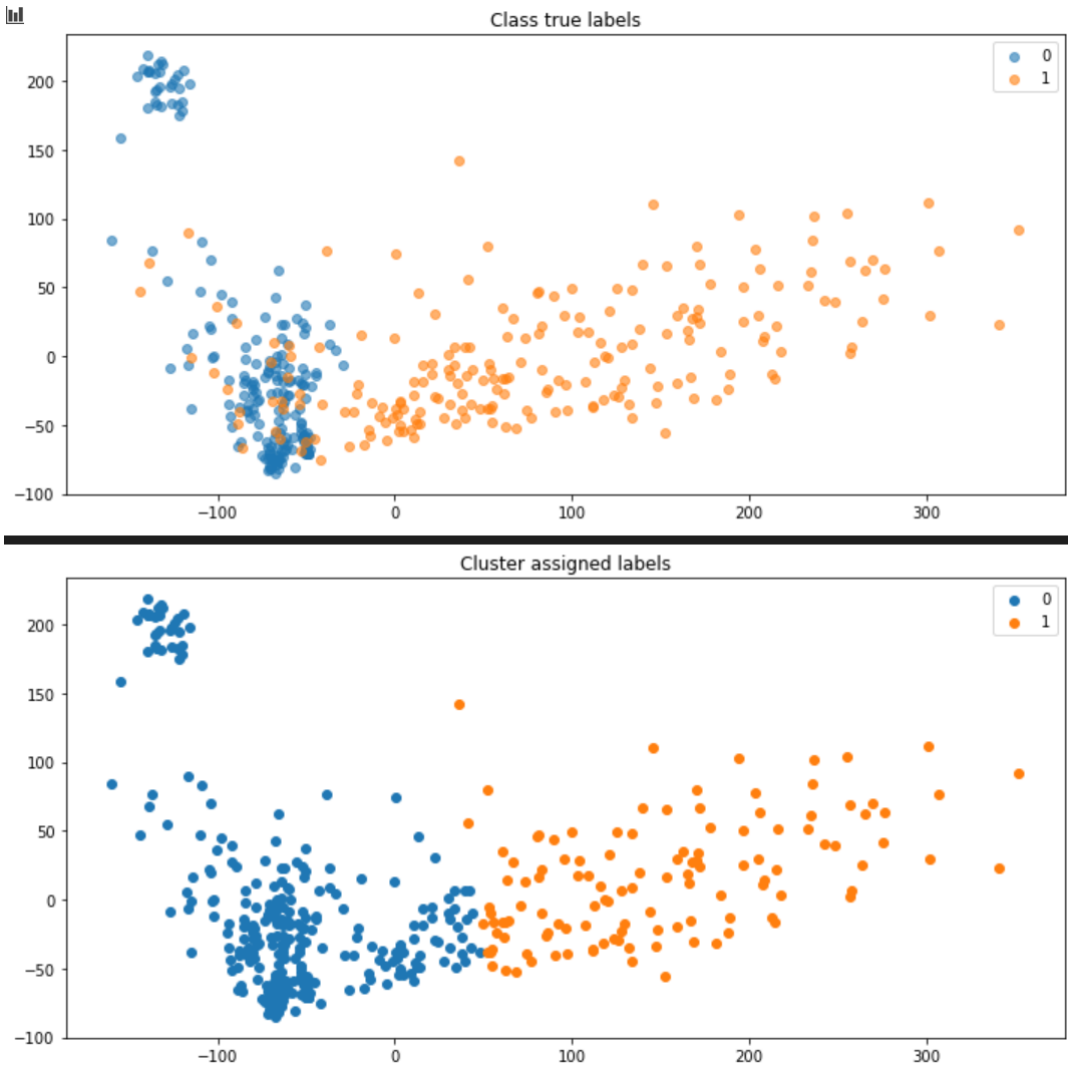

## A.15 Train data classification accuracy with the best selection of pixels from 500 pixels



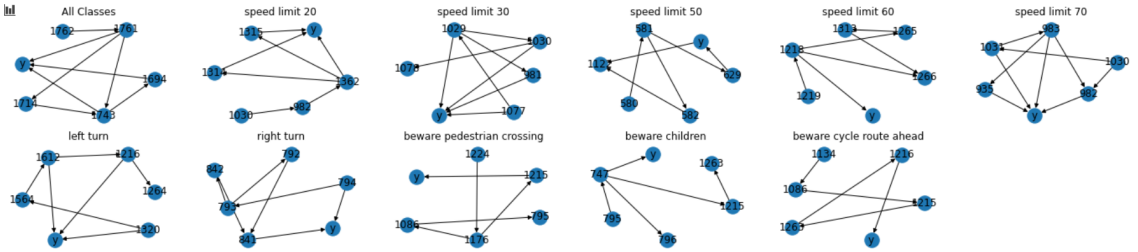
## A.16 Clustering Labels



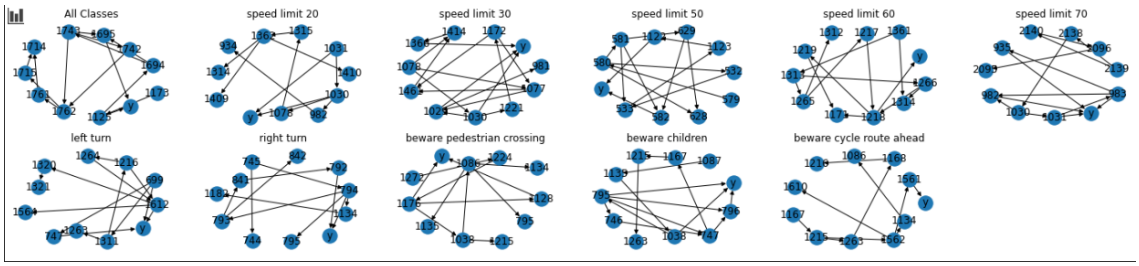
## A.17 Clustering Labels 250 best pixels



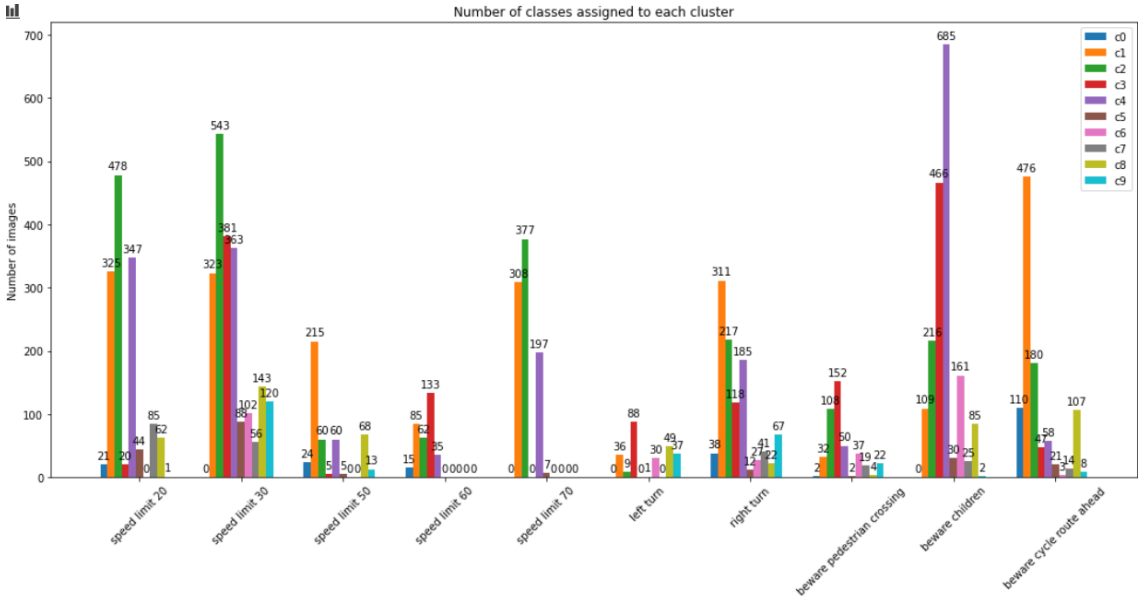
## A.18 5 Pixel Bayes Networks by class



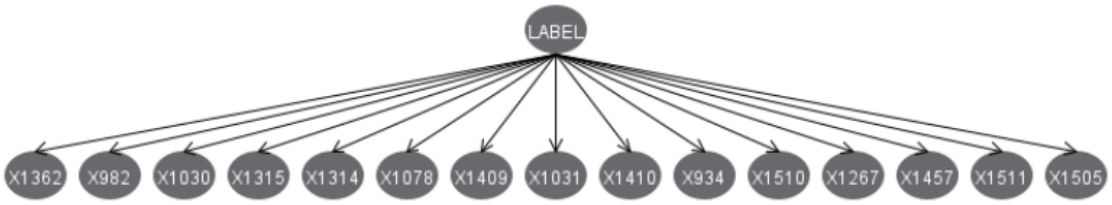

## A.19 K-means cluster with 10 clusters and 10 classes



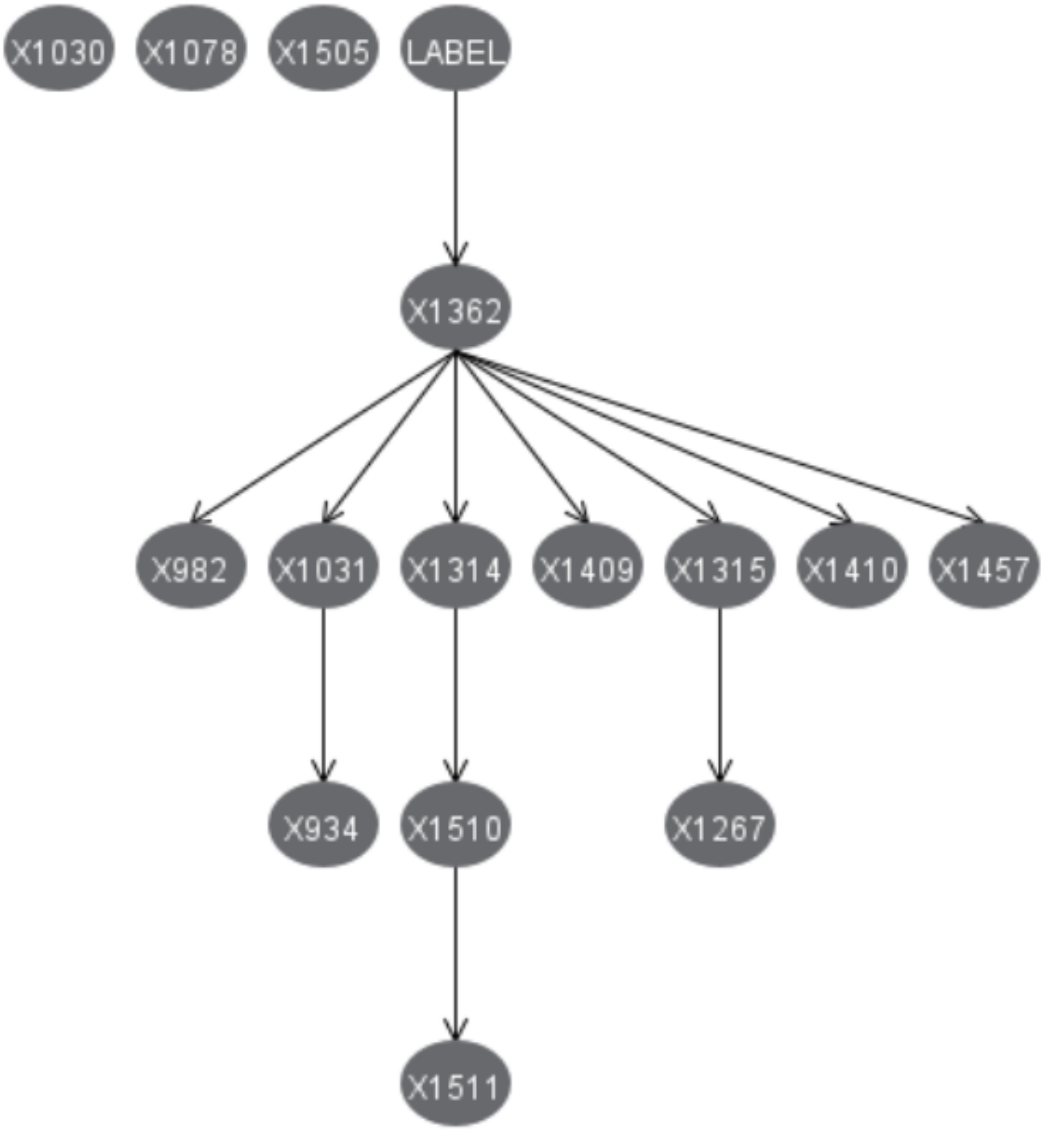
## A.20 K-means cluster with 10 clusters and 10 classes



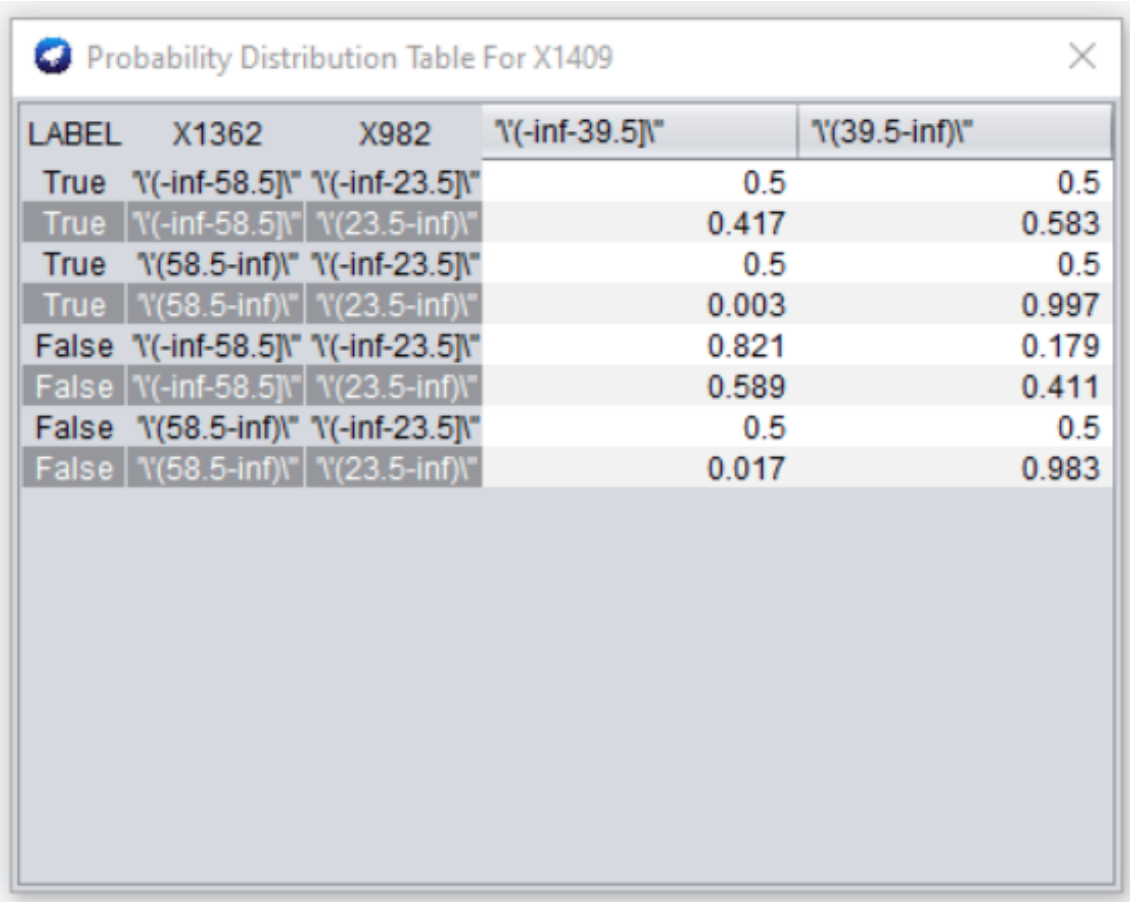
### A.21 K2, kParents=1, initAsNB=True, ADtree=True



### A.22 K2, kParents=1, initAsNB=False, ADtree=True



### A.23 Probability Distribution Table for X1409




| LABEL | X1362 | X982 | '\'(-inf-39.5]\'' | '\'(39.5-inf)\'' |
|---|---|---|---|---|
| True | '\'(-inf-58.5]\'' | '\'(-inf-23.5]\'' | 0.5 | 0.5 |
| True | '\'(-inf-58.5]\'' | '\'(23.5-inf)\'' | 0.417 | 0.583 |
| True | '\'(58.5-inf)\'' | '\'(-inf-23.5]\'' | 0.5 | 0.5 |
| True | '\'(58.5-inf)\'' | '\'(23.5-inf)\'' | 0.003 | 0.997 |
| False | '\'(-inf-58.5]\'' | '\'(-inf-23.5]\'' | 0.821 | 0.179 |
| False | '\'(-inf-58.5]\'' | '\'(23.5-inf)\'' | 0.589 | 0.411 |
| False | '\'(58.5-inf)\'' | '\'(-inf-23.5]\'' | 0.5 | 0.5 |
| False | '\'(58.5-inf)\'' | '\'(23.5-inf)\'' | 0.017 | 0.983 |