Data Mining & ML

Group 4

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.1) containing the majority of our code for this coursework. These functions provide utility for loading data, pre-processing data, building models and other convienience 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.2). We frequently used downscaling thoughout 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.9).

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 Bayse Networks, and offered a small accuracy uplift for the naive bayse 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 replicatable 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 convienient.

Pixel Selection
(Figure A.4) 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.3), 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 bayse classifiers for each class of labels, each plot on the x axis is using all preceeding pixels for classification see Figure in Appendix A.4.

2 Naive Bayesian Networks

We used the SKLearn librarys naive bayse modules to build the naive bayse 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 classifiers performance varied.
Observations:

1. Using no pre-processing technique with naive bayse 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 bayse
3. Naive bayse performs much better when it is only trained to perform binary classifications
4. Naive bayse has serious problems with overfitting when the distribution of classes in the dataset is poorly balanced.
5. Under the right condistion naive bayse can make excellent predictions, we observed prediction accuracy as high as 88% with our test data.

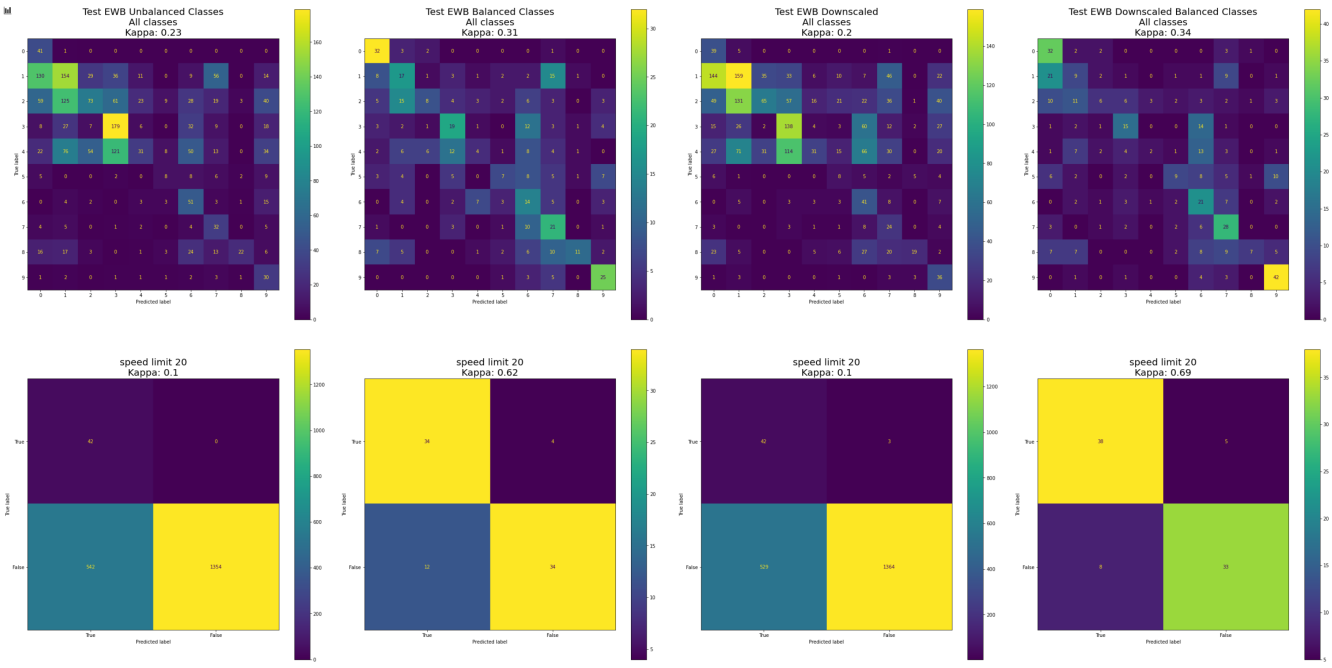


Figure 1: Naive bayse confusion matrix showing Equal Width Binning

IMAGE NEEDS RESIZING TO READ TEXT (Sam: maybe reduce it to 3 columns instead of 4?)

3 Complex Bayes nets

Describe & analyse the problem. Show all experiments complete with graphs and tables. Discuss produced software quality & discuss interesting properties of the data and algorithms

3.1 Building Bayes Networks

Bayes networks represent probablistic directed acyclic graphs that define the relationshps 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.

We decided to approach this task using both Weka and Python, with the intention of verifying our results against the other. 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 decided to use the pgmpy library to build our Bayes Networks in python, this immediately presented us with 2 computational complexity problems:

1. Building all the conditional probability factors.
2. Learning the optimal edges.

We handled the first problem by discretizing the greyscale values using equal width and frequency binning (see section 1.2.1) 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

3.3 Experimental Results

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.8, 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 clusting with the EM and Elkan algorithms...(talk about this more)
Becuase 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).

Figures needed (see task 9)

Appendices

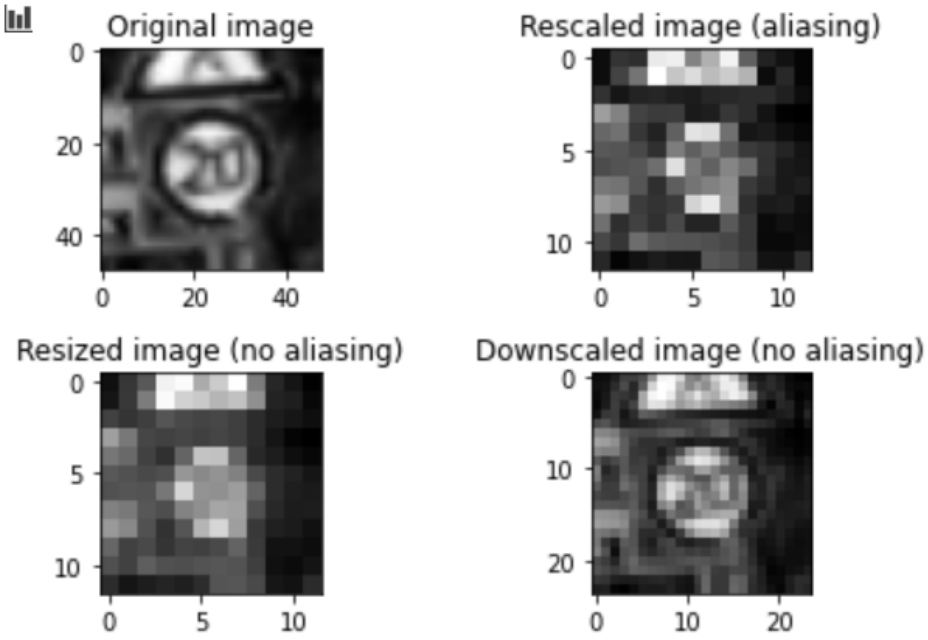
A Appendix A

A.1 Module Table

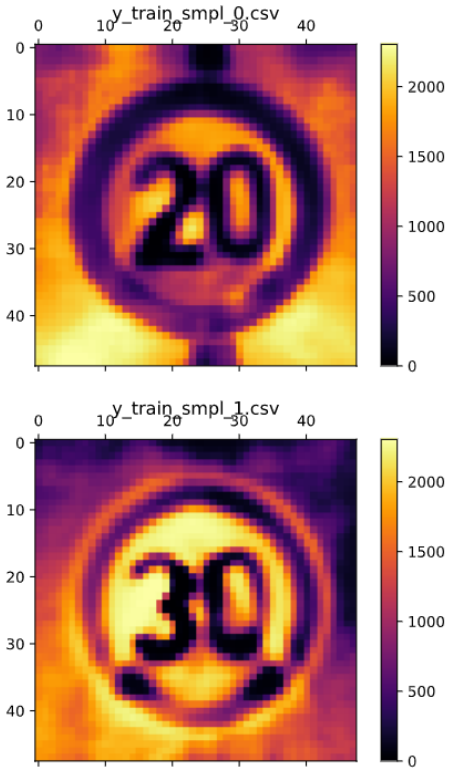
The following are located in the Scripts folder

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

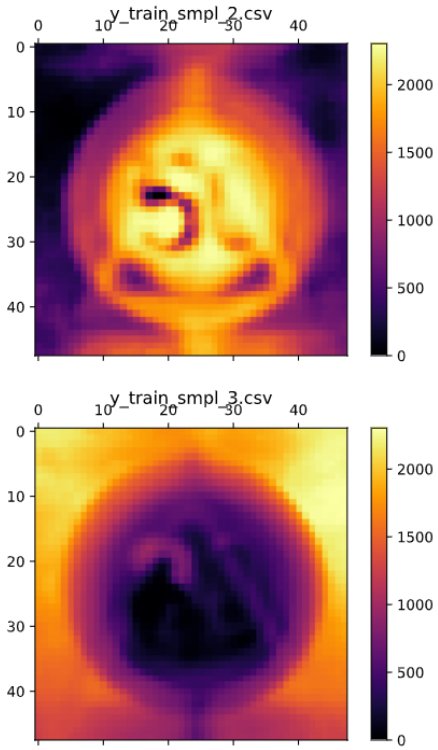
A.2 Downsampling example



A.3 Heatmaps

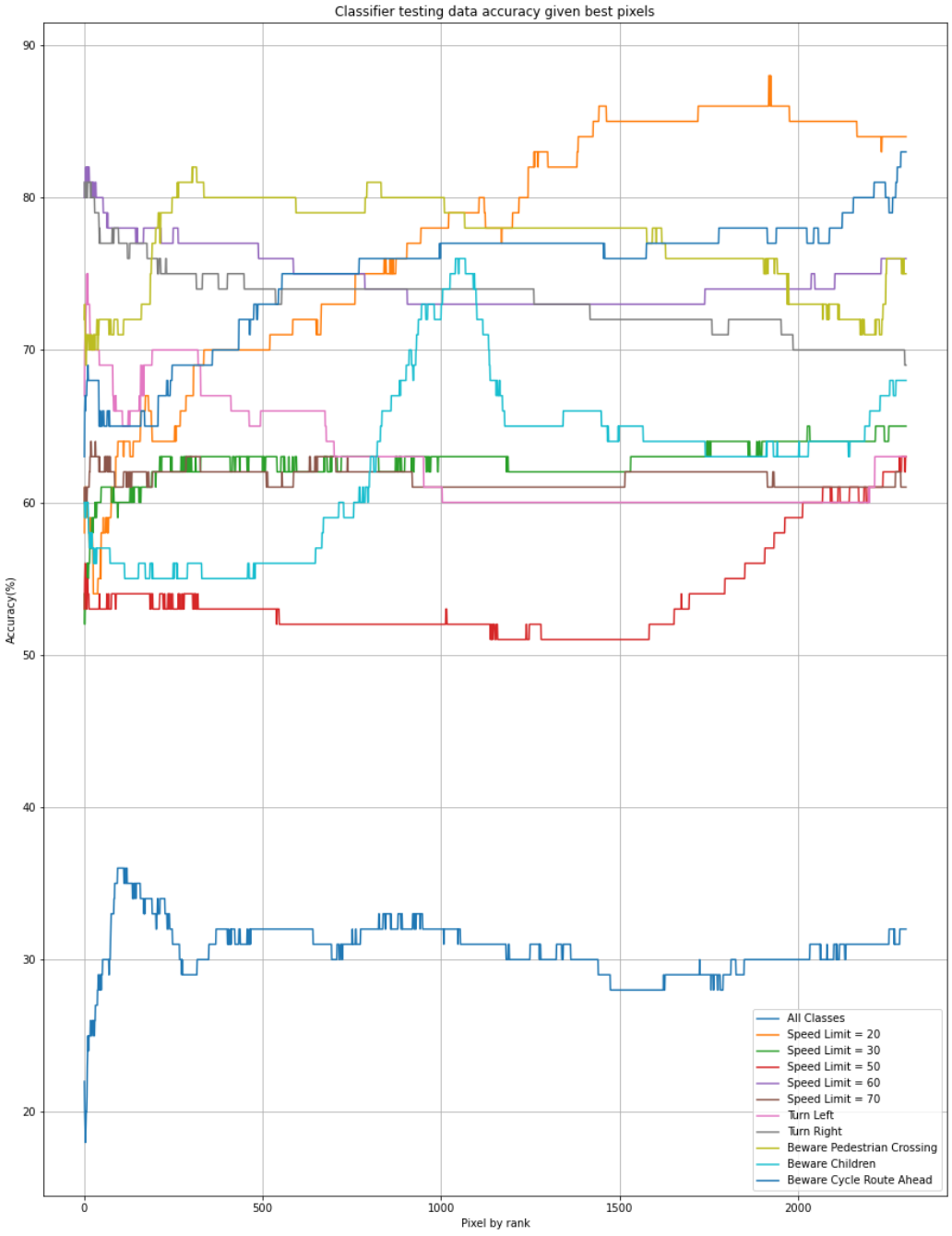


(a) 20mph & 60mph

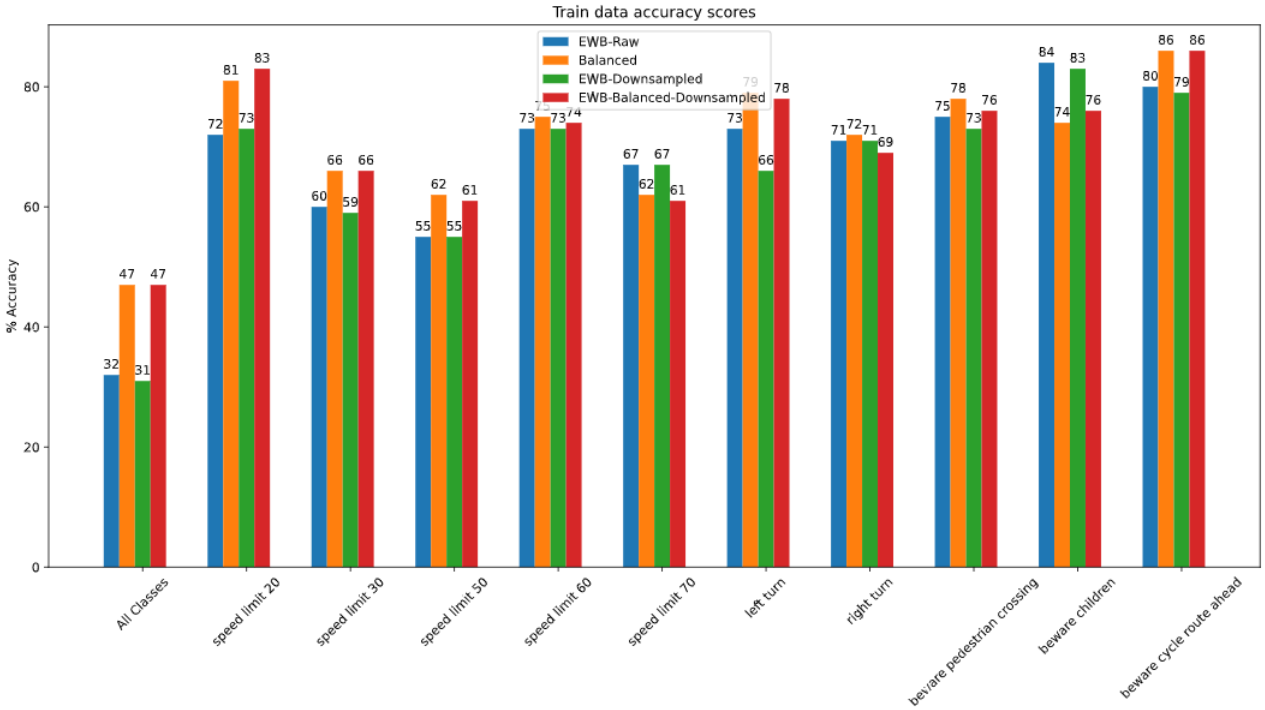


(b) 50mph & 60mph

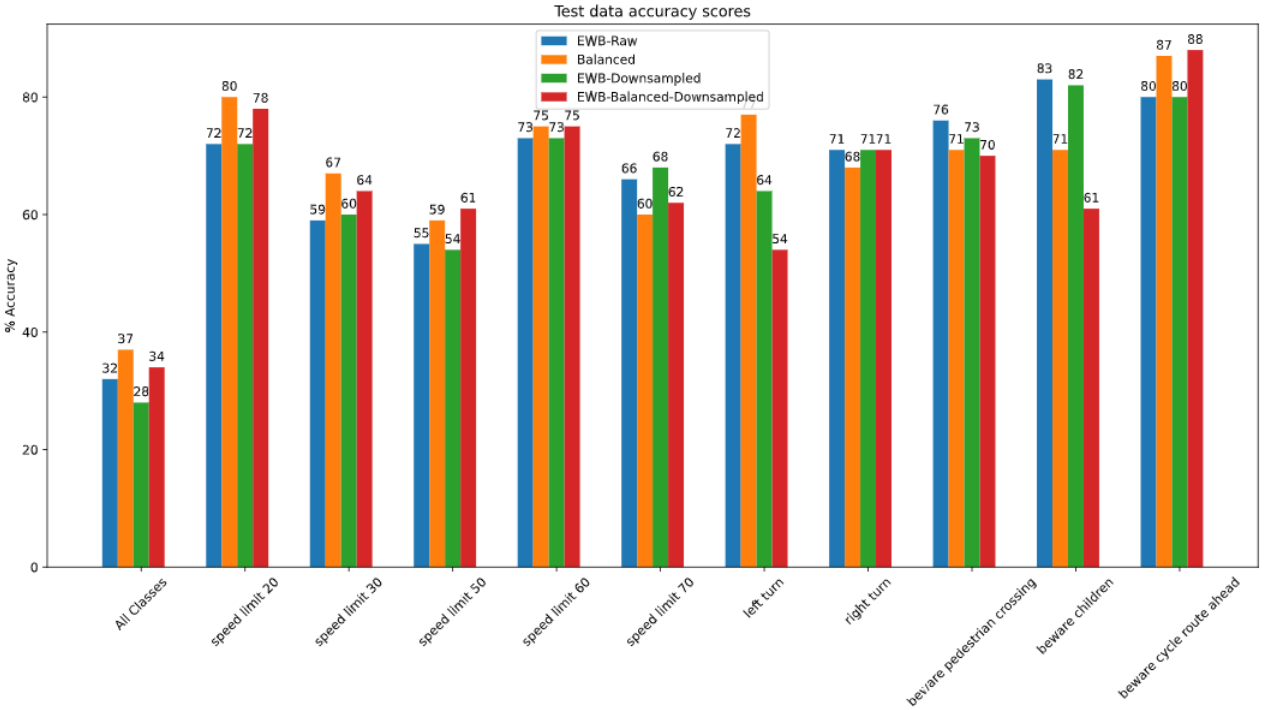
A.4 Accuracy of classifiers built using the first n pixels for each class



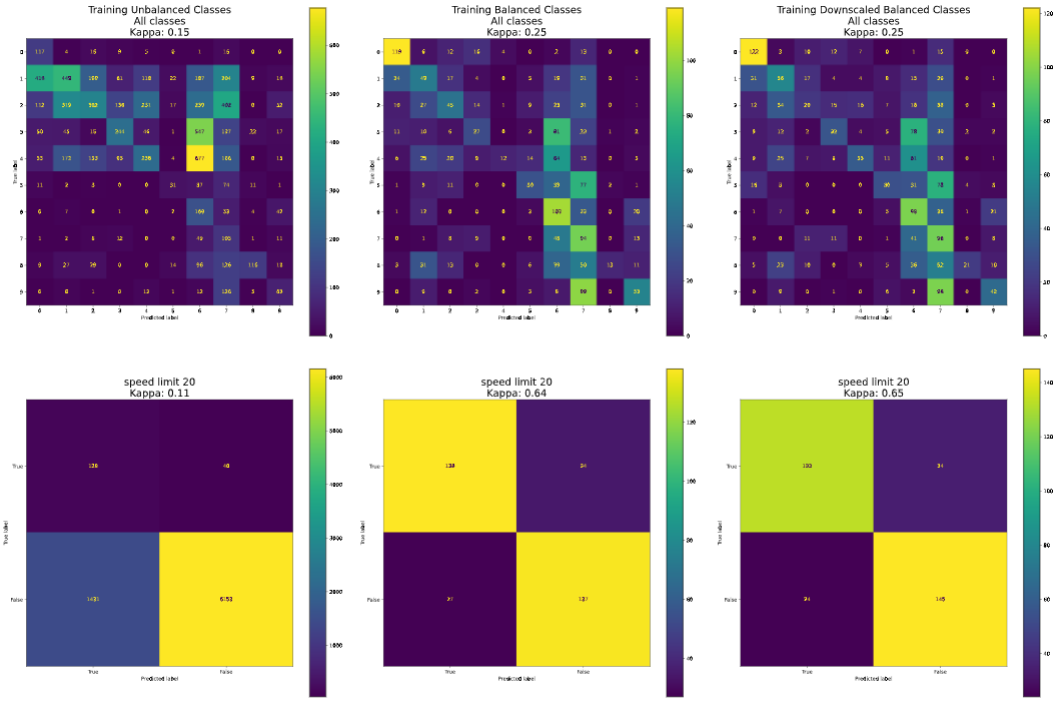
A.5 Accuracy of Equal Width Binning on Train data



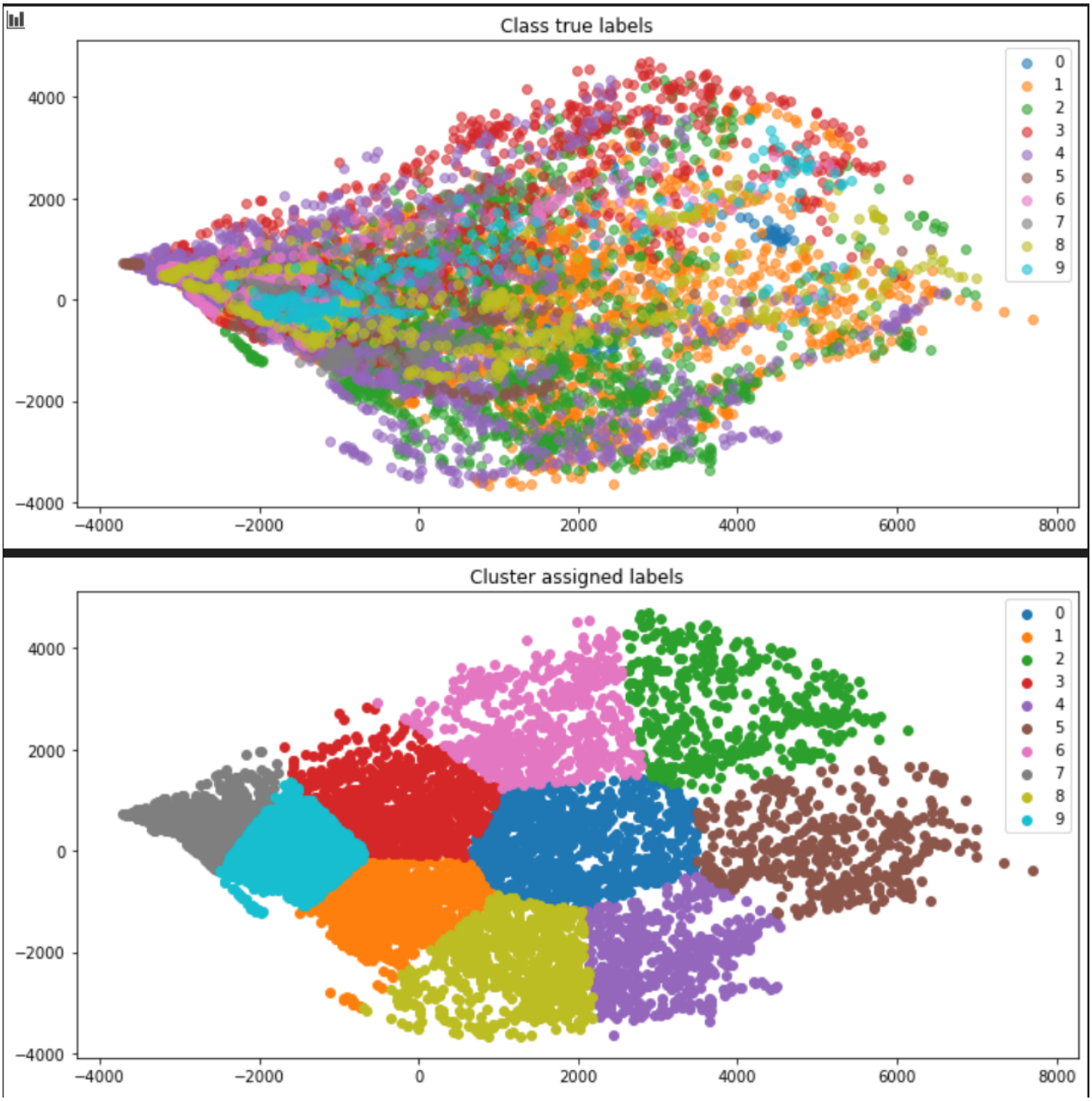
A.6 Accuracy of Equal Width Binning on Test data



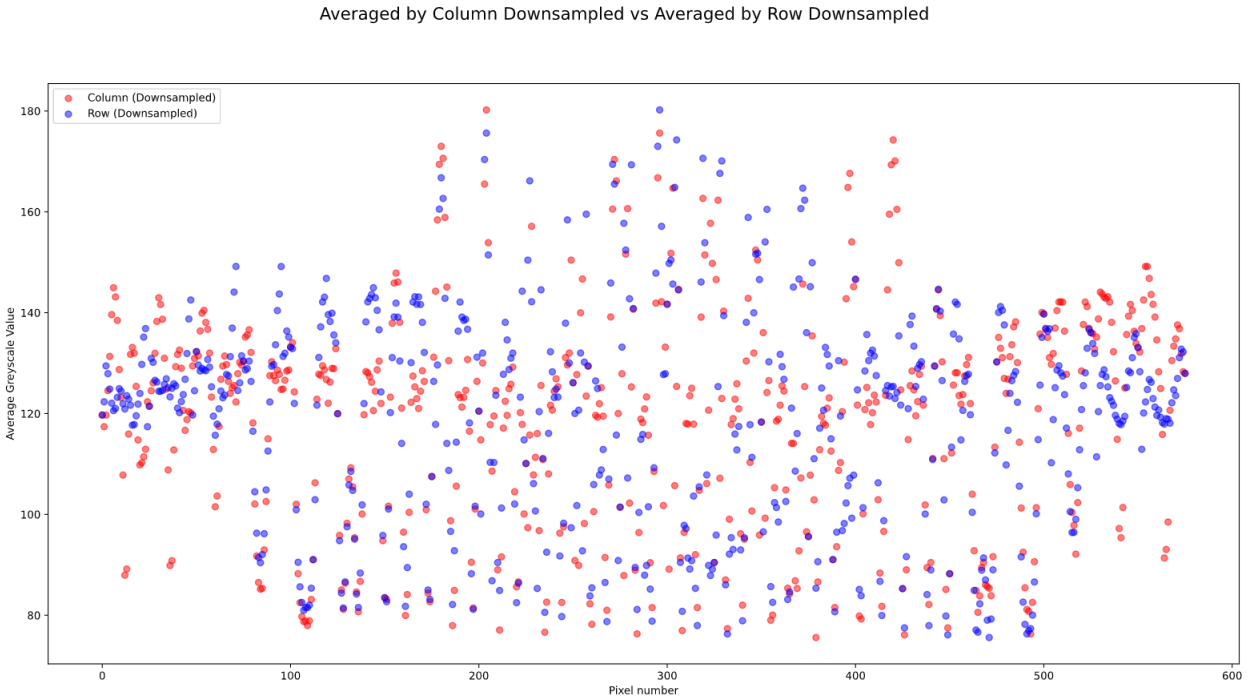
A.7 Training confusion matrices for Naive Bayes



A.8 No clusters of classes


Class true labels
Cluster assigned labels
0
1
2
3
4
5
6
7
8
9
4000
2000
0
−2000
−4000
−4000
−2000
0
2000
4000
6000
8000

A.9 Averaged by Column Downsampled vs Average by Row Downsampled



A.10 12x12 Downsampled image