Classification of NMNIST

Rajanish Rajagopalan, Swetha S George, Murugesan Nagarajan, Malini Chatterjee, Xinyi Wu, Yinghau Mo

Contents

Objective

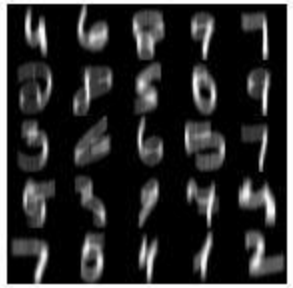
- In this project we built different classifiers to classify five numbers/classes from Noisy-MNIST(NMNIST) dataset.
- The aim of the project was to find the best classifier on this dataset based on the metrics obtained from Confusion Matrix

NMNIST Data Samples

- Data Source NMNIST_Data.zip Reference: http://www.numericinsight.com/DOWNLOADS.php
- Noise - Additive White Gaussian Noise (AWGN), Motion Blur, Combination of Reduced Contrast and AWGN
- AWGN is a basic noise model used to mimic the effect of many random processes https://en.wikipedia.org/wiki/Additive_white_Gaussian_noisehttps://en.wikipedia.org/wiki/Additive_white_Gaussian_noise

Sample images from the n-MNIST dataset:

n-MNIST with Additive White Gaussian Noise (AWGN)



n-MNIST with Motion Blur

n-MNIST with reduced contrast and AWGN

Training and Testing Data

- Each sample image is 28x28 and linearized as a vector of size 1x784. So, the training and test datasets are 2-d vectors of size mentioned below. Classes taken 5.
- The training dataset used is NMNIST(Noisy MNIST)
  - Classes: 1,2,4,7,8 (5 classes)
  - Dimensions of Features: 30658 X 784
  - Dimensions of Target: 30658 X 1
- The testing dataset used is test set of NMNIST:
  - Classes: 1,2,4,7,8 (5 classes)
  - Dimensions of Features: 5151X 784
  - Dimensions of Target: 5151X 1

Glimpse at training dataset

Glimpse at testing dataset

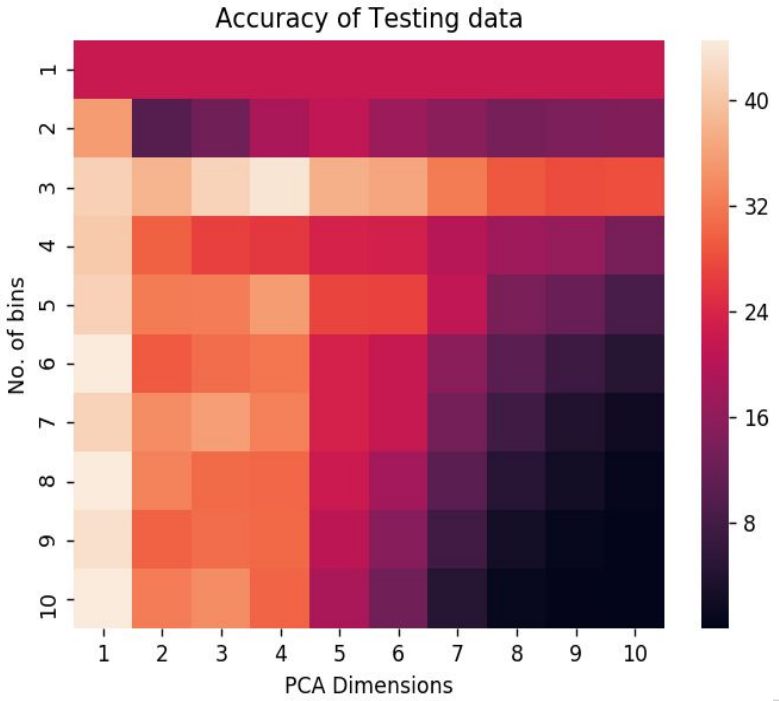
Histogram Classifier

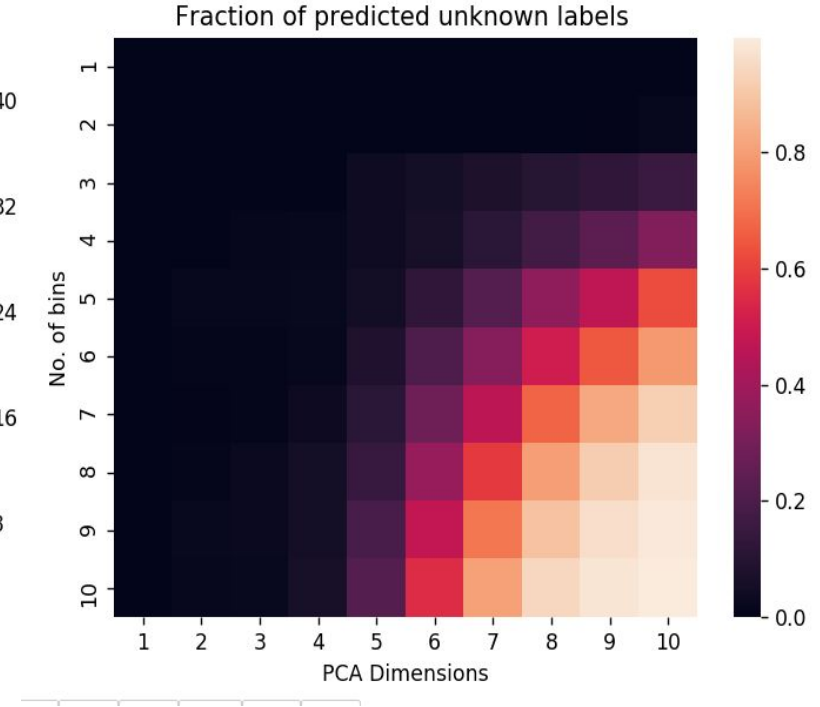
- Steps taken:
  - Reduced the dimension of dataset using PCA
  - Constructed Histogram function which can take varying no. of bins and PCA dimensions
    - Used python dict to dynamically allocate memory only for non-empty bins.
    - Avoids memory overflow errors when total number of bins grows.
  - Looked at model accuracy for varying bin sizes and number of PCA dimensions
  - Here the testing data and validation data for tuning model are the same

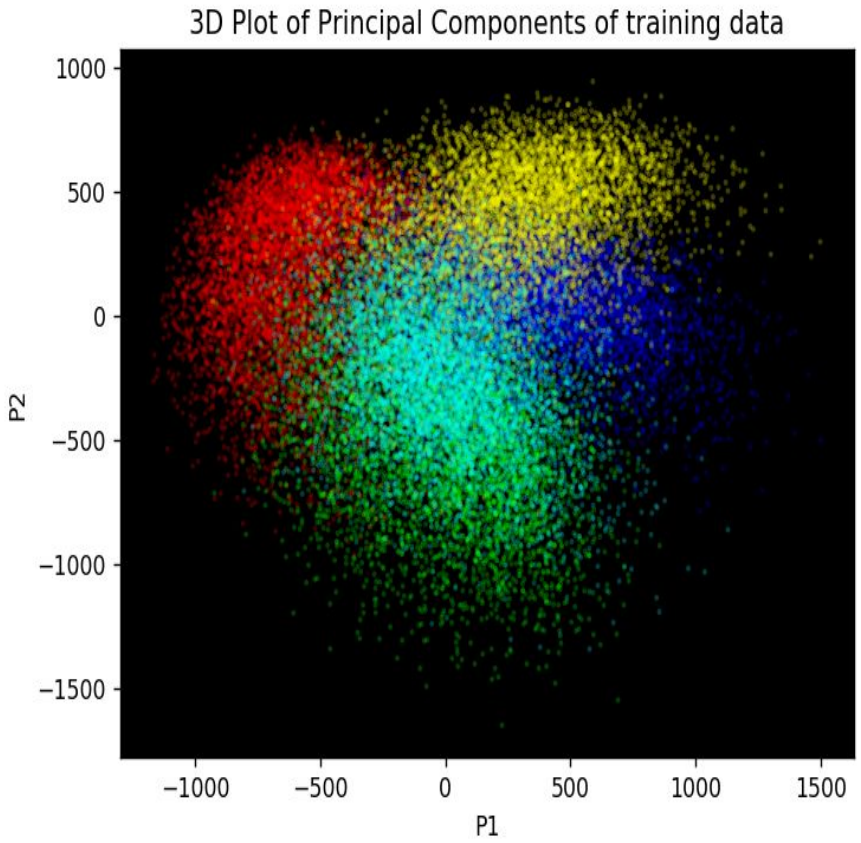
- Best accuracy: 44.47%
  - No. of bins > 2
  - No. of PCA dimensions = 1

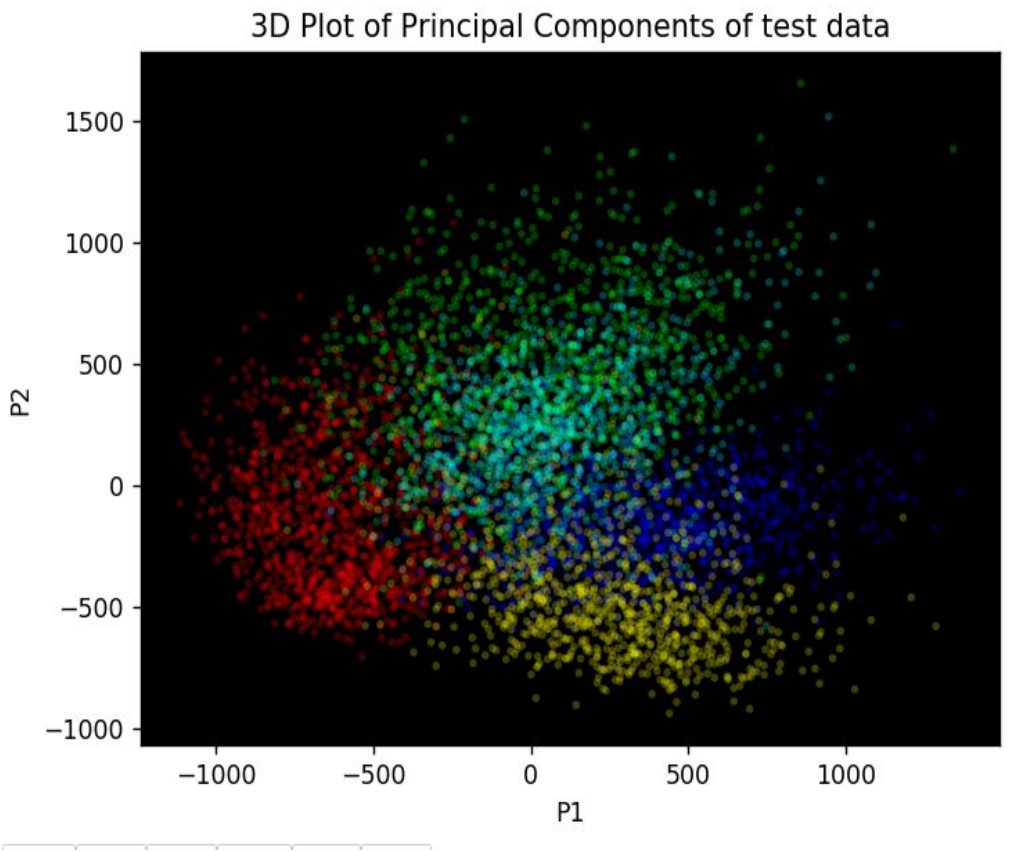
- Interestingly, increasing PCA dimensions beyond 1 does not seem to help!

- PCA dimensions > 4
  - Total bins increases exponentially (No. of bins ^ No. of PCA dimensions)
  - Increase in unknown predicted test labels thereby negatively impacting accuracy.





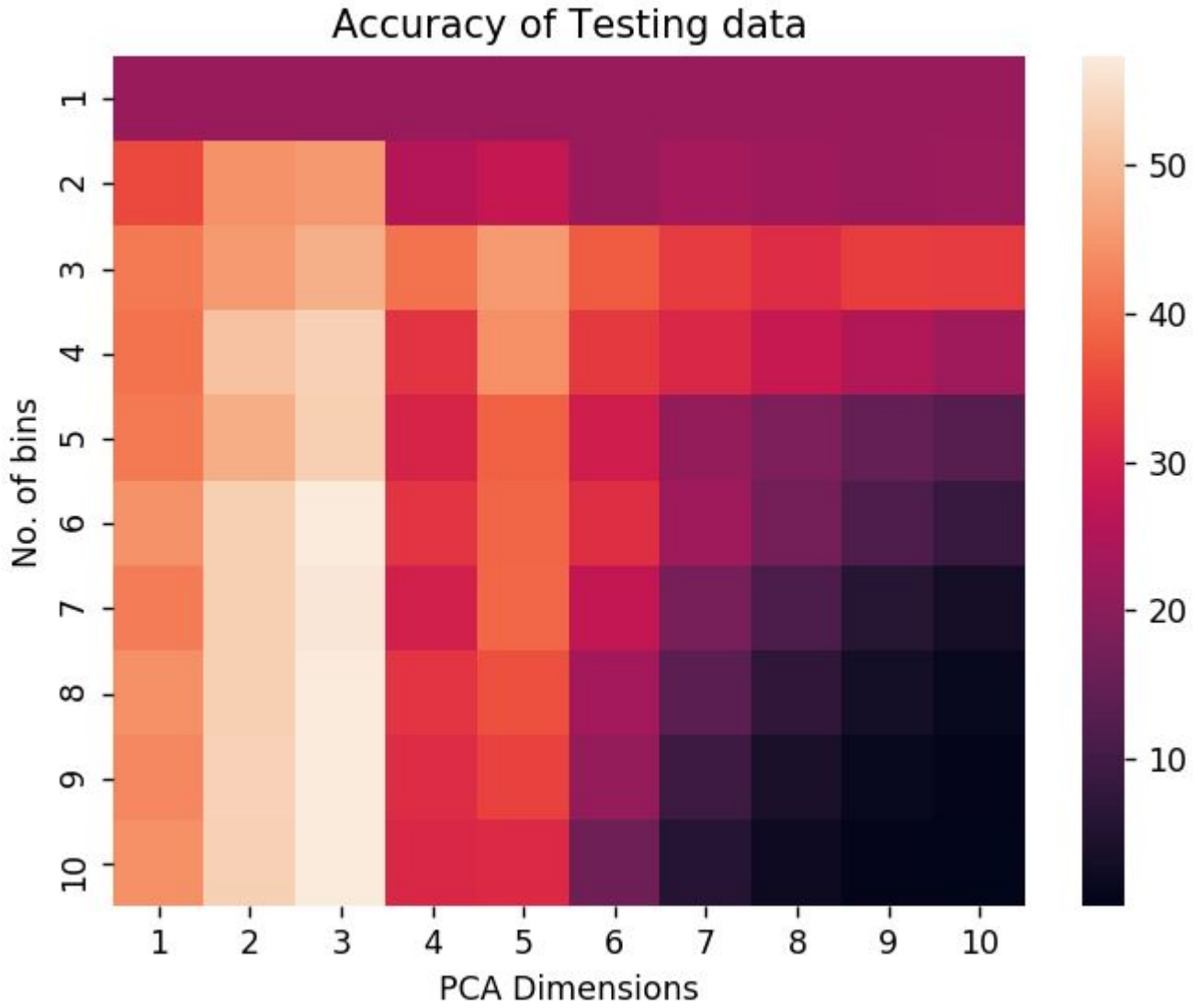




- Top 2 principal components for training (left) and test (right) data.
- Along 2nd principal axis, **green** and **yellow** labels are flipped between training and test data.
- This may be negatively impacting the accuracy when P2 is added.

After removing 2nd PCA component

- Best Accuracy: **57.33%**
  - No. of bins > 5
  - No. of PCA dims = 3
- Accuracy increased by 13%!!!
- The confusion matrix was calculated for No. of bins = 9 and No. of PCA dims = 3



Classifier Performance

Histogram Classifier on training data

	1	2	4	7	8
1	5716	657	29	22	318
2	642	3021	695	560	1040
4	68	1105	2989	1235	445
7	204	573	887	3221	1380
8	471	733	126	1020	3501

Accuracy: 60.17%

Class	1	2	4	7	8
PPV(%)	80.50	49.61	63.24	53.17	52.38

Histogram Classifier on test data

	1	2	4	7	8
1	971	95	1	4	64
2	127	421	208	95	181
4	13	176	436	210	147
7	44	94	123	518	249
8	44	76	25	219	610

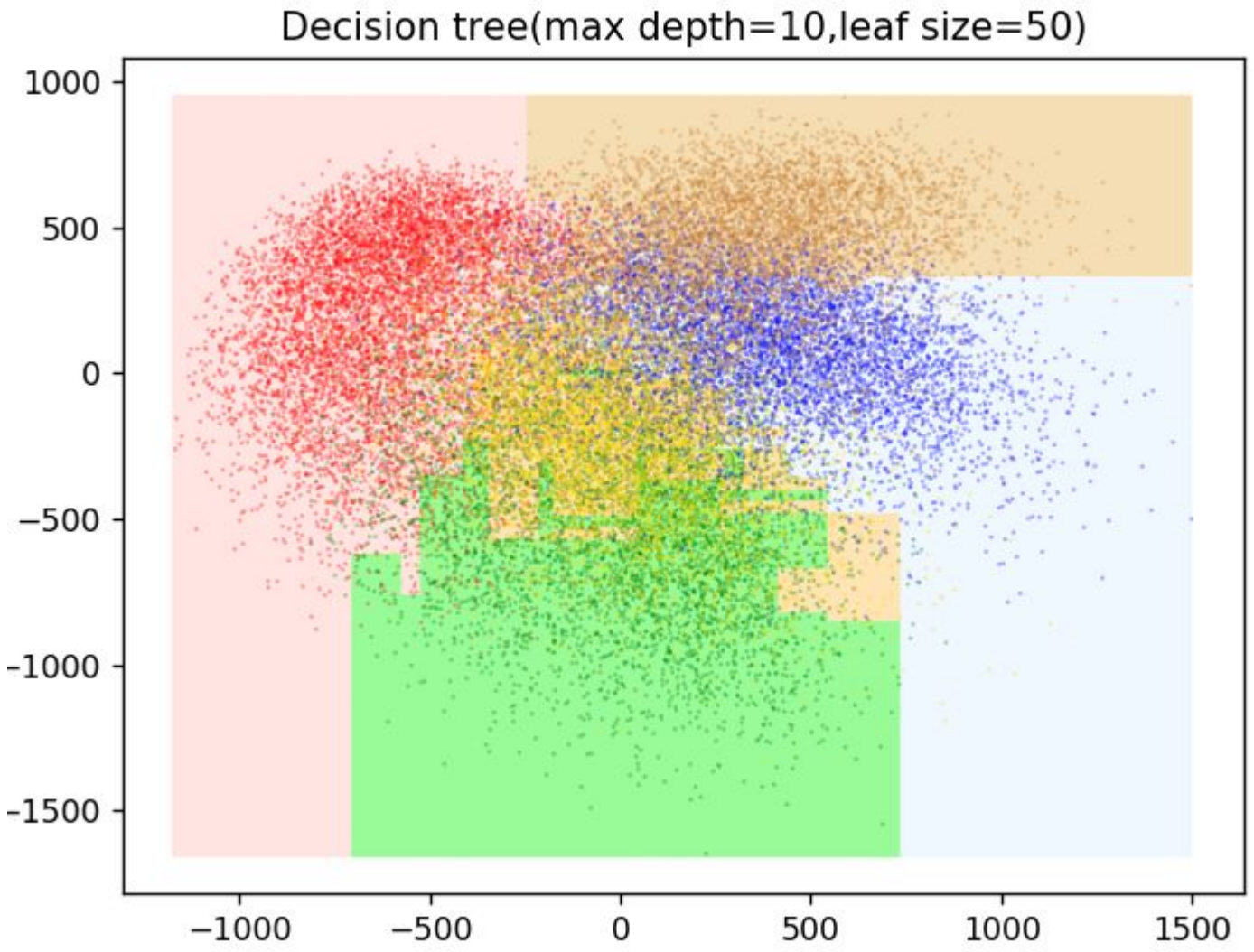
Accuracy: 57.33%

Class	1	2	4	7	8
PPV(%)	80.98	48.84	54.98	49.52	48.76

Decision Tree Classifier

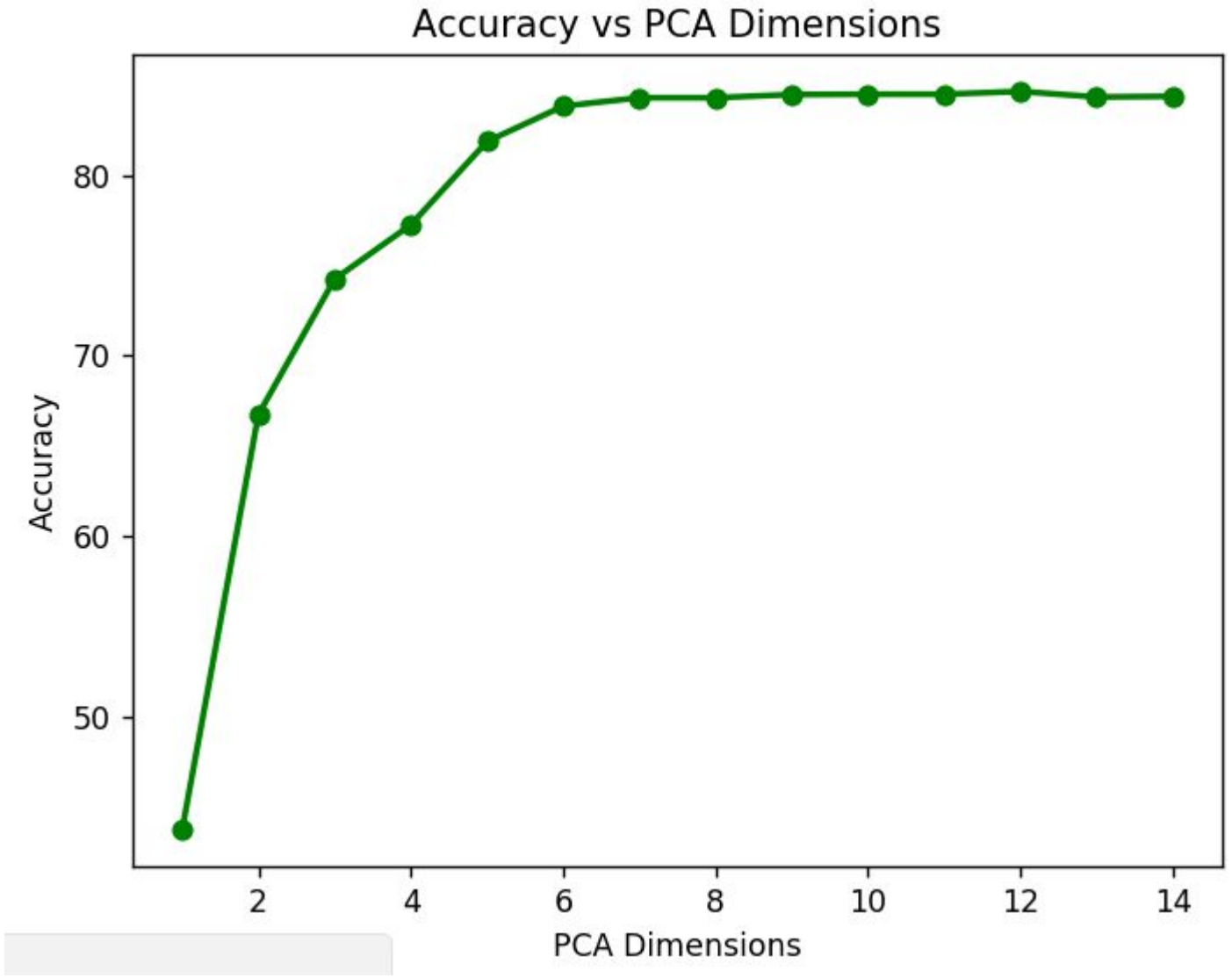
- Steps:
  - Reduced dimension of the dataset using PCA
  - Constructed decision tree classifier using sklearn.tree.DecisionTreeClassifier
  - Trained models across varying number of PCA components
  - Found the optimum classifier parameters for a fixed PCA dimension
  - After which observed accuracy change for varying PCA components on test data keeping the classifier parameters constant.
  - Here the testing data and validation data for tuning model are the same

Decision Tree boundary for model trained with 2 PCA components



Accuracy increases with PCA dimensions till around 7 and then remains roughly the same

So we chose PCA dimensions = 7 for this model



Classifier Performance

DecisionTreeClassifier on training data

	1	2	4	7	8
1	6239	145	101	136	121
2	205	4991	268	207	287
4	99	166	5096	275	206
7	149	187	352	5452	125
8	224	321	391	197	4718

Accuracy:86.42%

Class	1	2	4	7	8
PPV(%)	90.21	85.90	82.09	87.00	86.46

DecisionTreeClassifier on test data

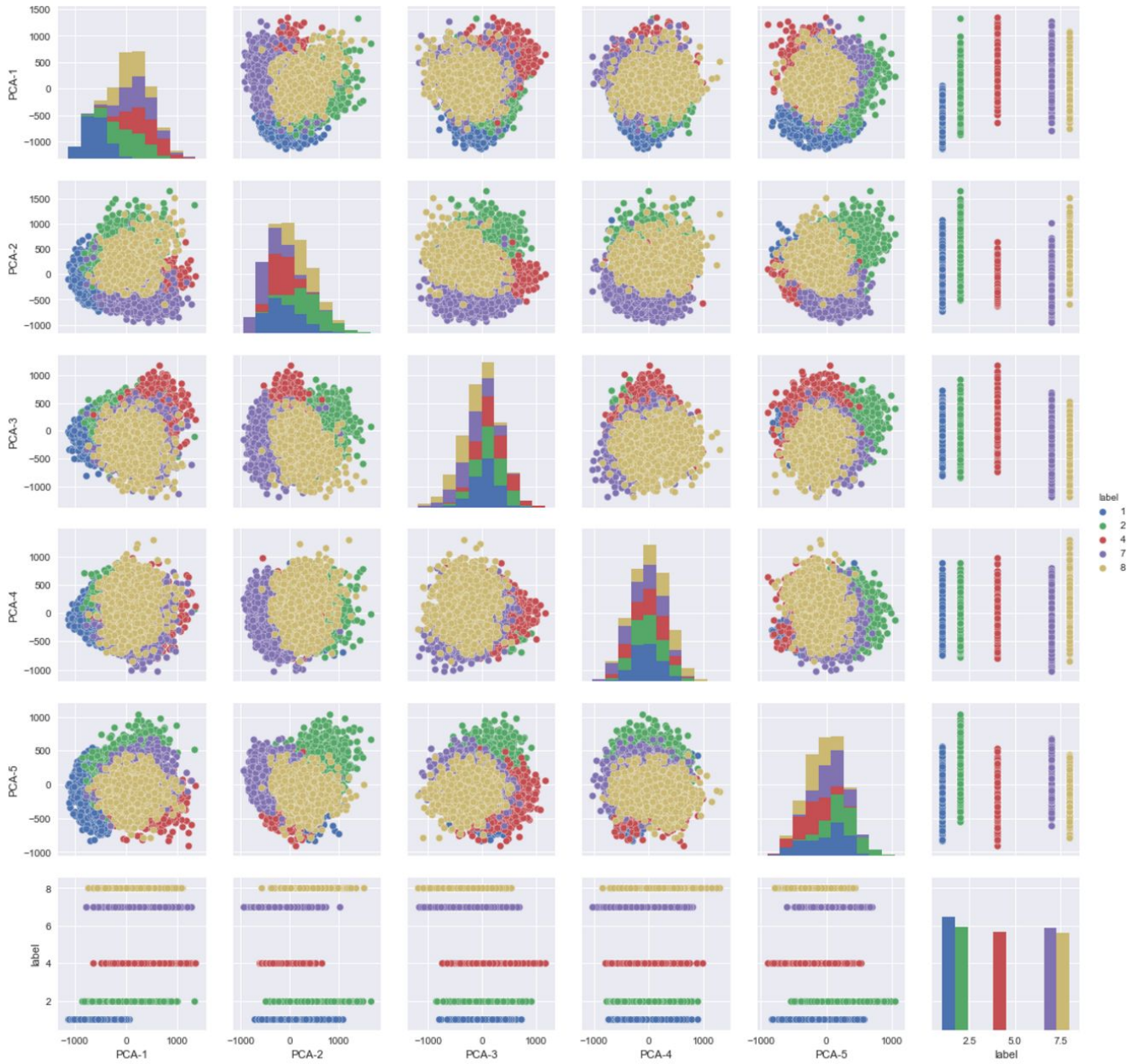
	1	2	4	7	8
1	1041	24	17	31	22
2	37	860	41	39	55
4	19	40	817	63	43
7	33	36	62	875	22
8	29	64	79	43	759

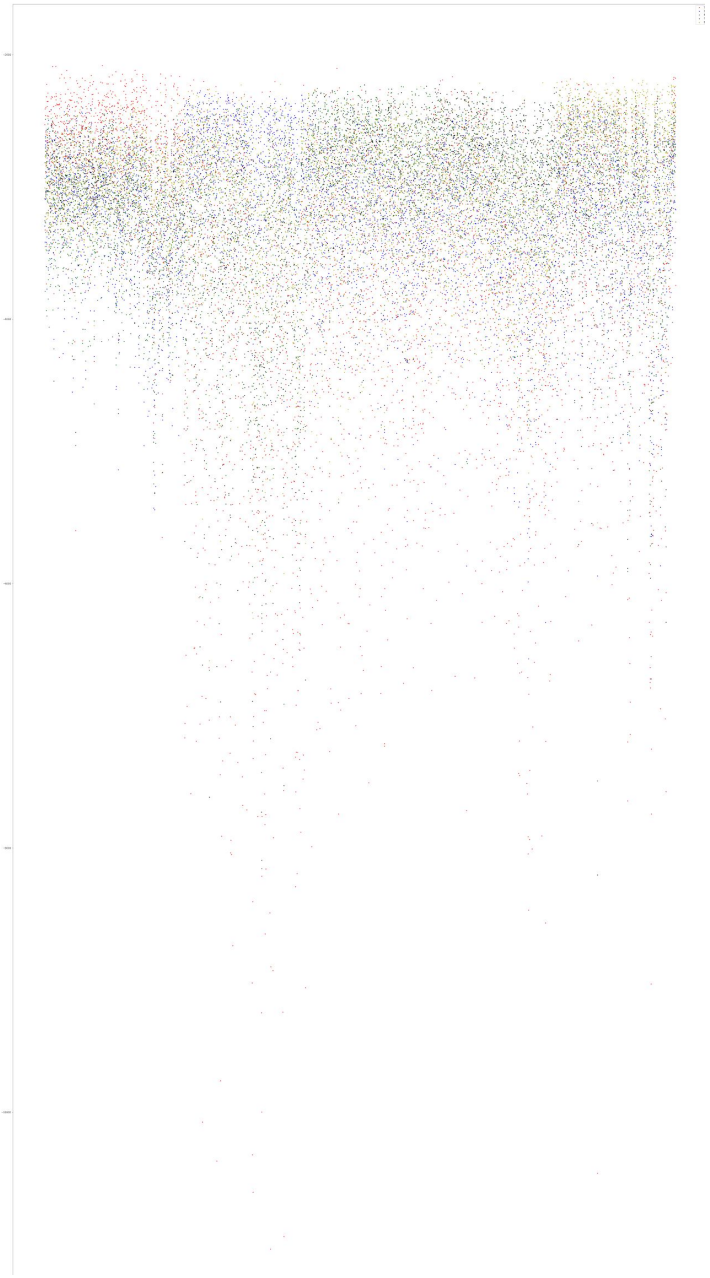
Accuracy: 84.49%

Class	1	2	4	7	8
PPV(%)	89.82	83.98	80.41	83.25	84.24

Bayesian Classifier

- Steps:
  - Reduced dimensions of original dataset for [1,2,4,7,8] using PCA, use 5 components.
  - Built model based on the processed training model, and adopted it onto the testing data.
  - After obtaining the final testing labels, calculate the accuracy and confusion matrix.
- Validation:
  - The classifier was then applied on test data as well as training data to predict class labels. Accuracy was calculated.
- To measure classifier performance confusion matrix was created for both training and test dataset.





Visualization of 5 Components for [1,2,4,7,8]

probability points of testing data of classes

Result: 75.364% accuracy for the Bayesian Model with 5 PCA

	1	2	4	7	8
1	824	60	10	36	205
2	32	729	29	20	222
4	31	53	756	75	67
7	33	16	113	806	60
8	40	60	50	57	767
PPV	0.8583	0.7941	0.7891	0.8109	0.5806

From the result , we can see that the PPV of number 8 is relatively lower than other numbers, Especially the Bayesian Classifier mixes up 8 with number 1 and number 2.

According to the results of Bayesian and histogram model, we can see that the accuracy of Bayes is lower than the Histogram's. However, we know that something wrong with Histogram model while there is an existence of vacant. Combining known knowledge, I think that this phenomenon may be caused by the characteristics of models, the histogram focus more on the frequency of data than the positions. What's more, the possibility of Bayesian model floats smoothly, which may cause the difference.

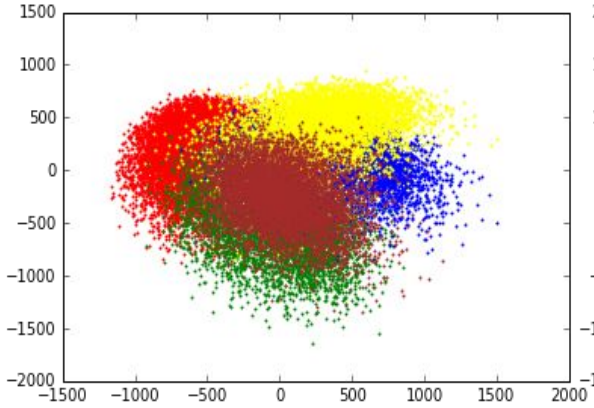
Linear Classifier

- Steps:
  - Reduced dimensions of original dataset for [1,2,4,7,8] using PCA, use 5 components
  - Built multiple linear models based on the processed training model, and adopted it onto the testing data.
  - After obtaining the final testing labels, calculate the accuracy and confusion matrix
- Validation:
  - The classifier was then applied on test data as well as training data to predict class labels. Accuracy was calculated.
- To measure classifier performance confusion matrix was created for both training and test dataset

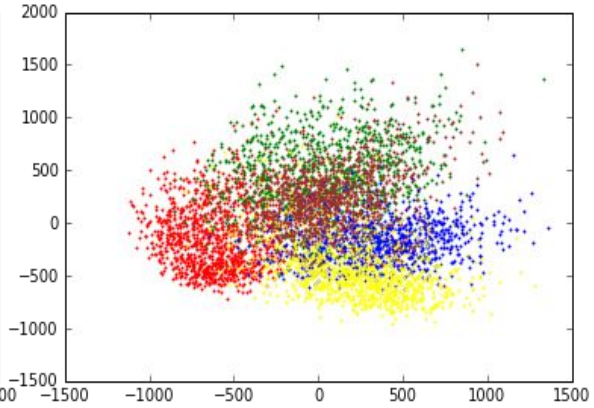
Linear Classifier Performance

PCA

Train Data



Test Data



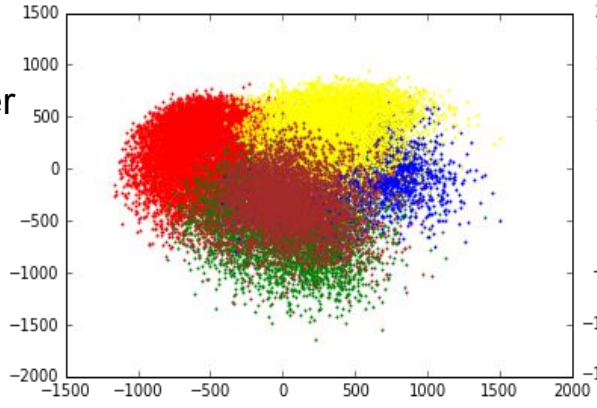
Train Data

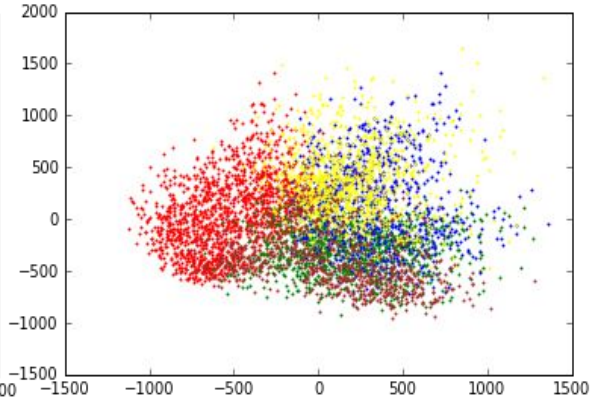
	1	2	4	7	8
1	6742	0	0	0	0
2	0	5958	0	0	0
4	0	0	5842	0	0
7	0	0	0	6265	0
8	0	0	0	0	5851
PPV	1	1	1	1	1
Acc%	100%	100%	100%	100%	100%

Test Data

	1	2	4	7	8
1	1135	0	0	0	0
2	0	1032	0	0	0
4	0	0	982	0	0
7	0	0	0	1028	0
8	0	0	0	0	974
PPV	1	1	1	1	1
Acc%	100%	100%	100%	100%	100%

Linear Classifier





	1	2	4	7	8
1	6327	88	85	98	144
2	575	4748	213	110	312
4	356	183	4317	744	242
7	511	139	772	4690	153
8	850	597	415	240	3749
PPV	0.73	0.82	0.74	0.79	0.81
Acc%	91%	93%	90%	91%	90%

91%

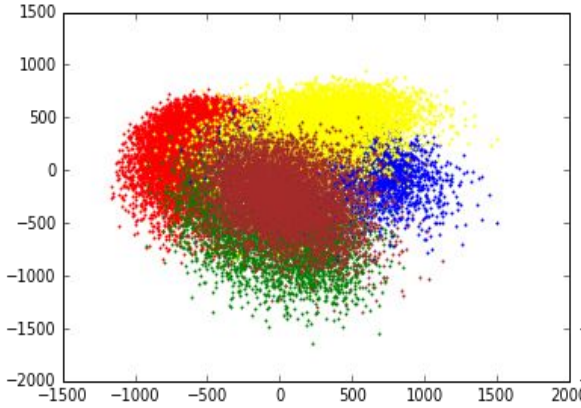
	1	2	4	7	8
1	898	64	2	16	155
2	324	28	468	160	52
4	15	624	210	111	22
7	64	271	133	101	459
8	134	101	31	601	107
PPV	0.62	0.02	0.25	0.10	0.13
Acc%	85%	60%	73%	65%	70%

70%

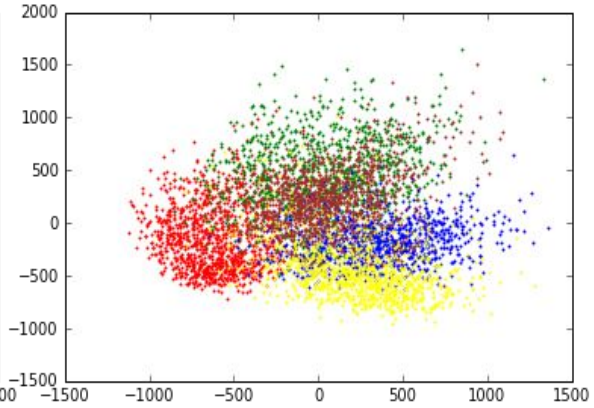
Linear Classifier Performance

PCA

Train Data



Test Data



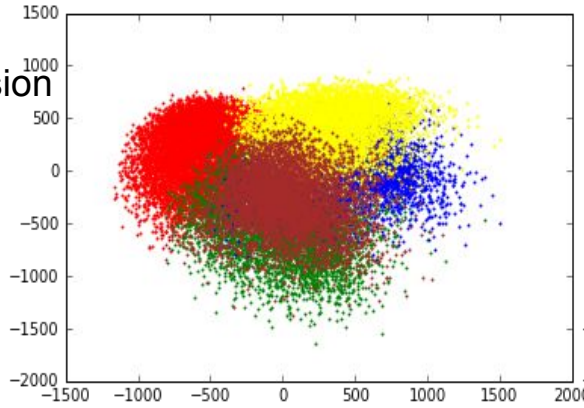
Train Data

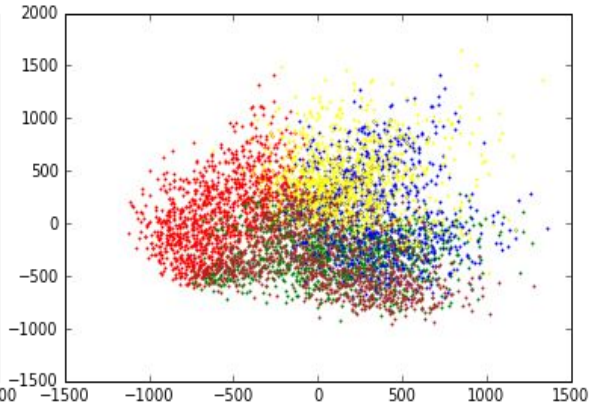
	1	2	4	7	8
1	6742	0	0	0	0
2	0	5958	0	0	0
4	0	0	5842	0	0
7	0	0	0	6265	0
8	0	0	0	0	5851
PPV	1	1	1	1	1
Acc%	100%	100%	100%	100%	100%

Test Data

	1	2	4	7	8
1	1135	0	0	0	0
2	0	1032	0	0	0
4	0	0	982	0	0
7	0	0	0	1028	0
8	0	0	0	0	974
PPV	1	1	1	1	1
Acc%	100%	100%	100%	100%	100%

Logistic Regression





	1	2	4	7	8
1	6055	167	133	174	213
2	335	4914	213	132	364
4	201	177	4474	666	324
7	337	134	666	4863	265
8	463	606	366	181	4235
PPV	0.82	0.82	0.76	0.81	0.78
Acc%	82%	82%	76%	81%	78%

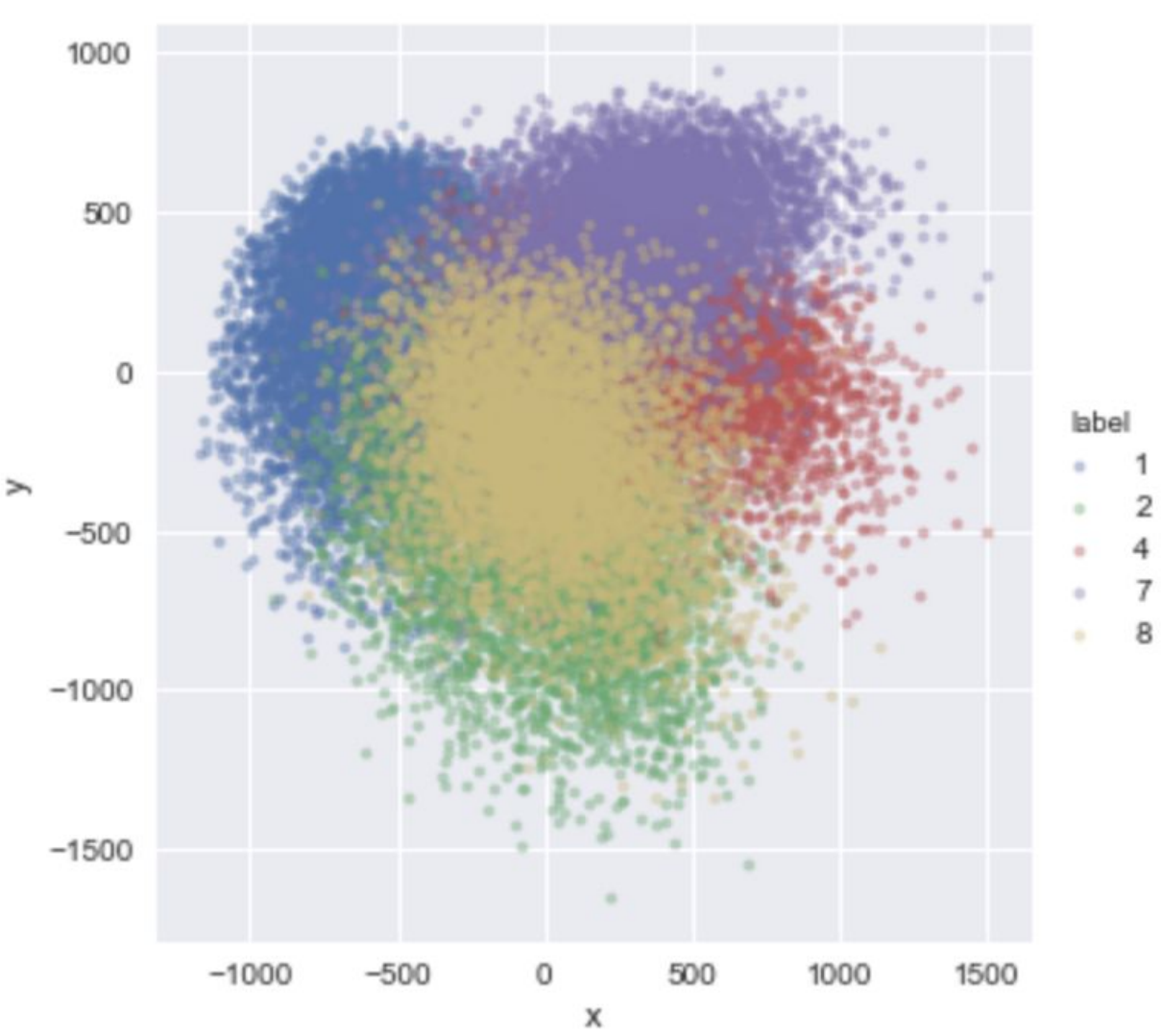
92%

	1	2	4	7	8
1	759	151	3	21	201
2	267	38	465	192	70
4	6	607	215	110	44
7	28	249	183	72	496
8	81	101	36	585	171
PPV	0.66	0.33	0.24	0.07	0.17
Acc%	66%	59%	72%	64%	68%

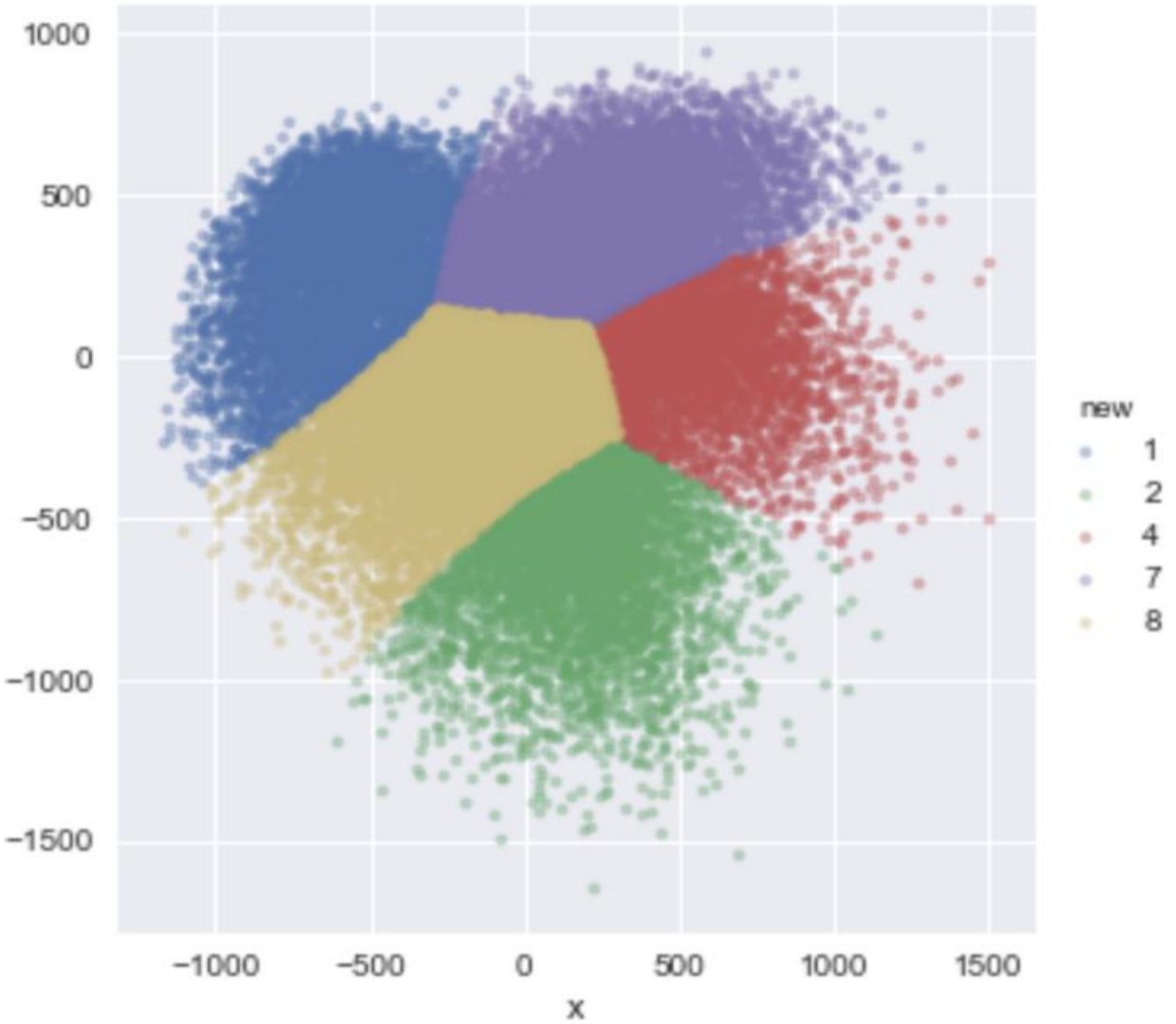
70%

Expectation Maximization

- Built an EM classifier using the first **five** principal components of P from PCA
- Accuracy on test data 76.58%
- Built another EM classifier using the first **two** principal components of P from PCA
- Accuracy on test data 62.30%
- It's reasonable that the accuracy went down after reducing the number of principal components used in prediction.
- We reduce it to two dimensions for better visualization. Built an EM classifier using the first **five** principal components of P from PCA



Plot based on the given labels



Plot based on the predicted labels

Visualization of the EM classifier using the first two principal components

Classifier Performance

EM Classifier using five principal components on training data

	1	2	4	7	8
1	6027	185	178	80	272
2	200	4909	224	64	561
4	90	87	3272	2095	298
7	224	106	1080	4469	386
8	250	270	281	250	4800

Accuracy:76.58%

Class	1	2	4	7	8
PPV(%)	89.39	82.39	56.01	71.33	82.04

EM Classifier using two principal components on training data

	1	2	4	7	8
1	5122	25	5	367	1223
2	181	2874	168	237	2498
4	67	151	3003	1801	820
7	280	47	528	4830	580
8	204	1649	369	358	3271

Accuracy: 62.30%

Class	1	2	4	7	8
PPV(%)	75.97	48.24	51.40	77.09	55.90

K-Nearest Neighbor Classifier

- Steps taken:
  - Reduced the dimension of the training dataset using PCA to 5 dimension.
  - Constructed the KNN classifier using the first 5 principal components.
  - Constructed the KNN classifier using the entire feature vector.
- Validation:
  - Predicted the class label for the training and test data with the first 5 principle components.
  - Predicted the class label for the training and test data with complete features.
- Observation
  - The accuracy of the KNN classifier is very low when the features are reduced to 5 dimension. The accuracy of the KNN classifier is more than 93% when the entire features are used.

KNN Classifier Performance

KNN Classifier using entire features on testing data

	1	2	4	7	8
1	1132	3	0	0	0
2	76	904	8	27	17
4	48	4	919	11	0
7	57	3	17	951	0
8	47	10	16	15	886

Accuracy:93.03%

Class	1	2	4	7	8
PPV(%)	83.23	97.83	95.72	94.72	98.11

KNN Classifier using five principal components on testing data

	1	2	4	7	8
1	817	153	0	17	148
2	217	41	447	239	88
4	3	468	335	90	86
7	23	180	132	75	618
8	104	61	59	554	196

Accuracy: 28.42%

Class	1	2	4	7	8
PPV(%)	70.18	4.54	34.42	7.69	17.25

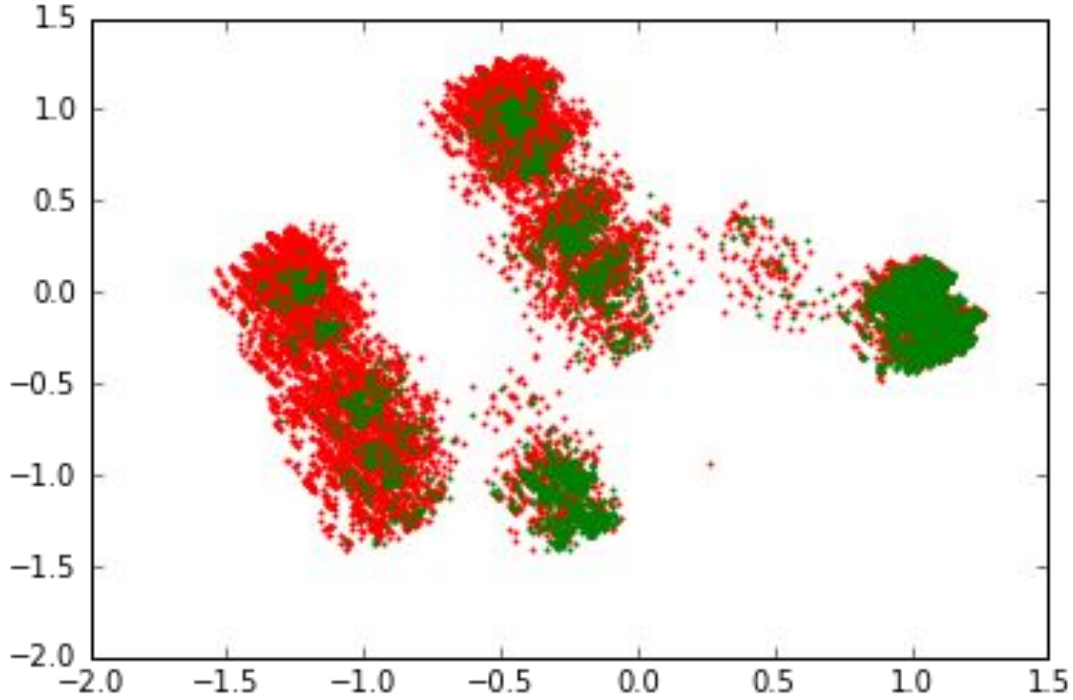
Conclusion

- We built and understood the working of different classifiers on the dataset

Classifier	Accuracy on test data(%)
Histogram	57.33
Decision Tree	84.49
Bayesian	74.36
Linear	70
Logistic Regression	70
Expectation Maximization	62.30
K-Nearest Neighbor	28.42

Appendix

- Use Census Data to classify salary range of citizens in to two classes (>$50k or <$50k).



Code for constructing and tuning Histogram

B - No: of BIns
D - No. of PCA dimensions
P - PCA components of training data
Ptest - PCA components of testing data
T - output target of training data
test_T - output target of testing data

```
def BuildHistogram(B,T,P,pmin,pmax):
    RC = (np.trunc(((B-1e-7) * (P - pmin)/(pmax - pmin)))).astype('int32');
    hist = {}
    for i,rc in enumerate(RC):
        key = tuple([x for x in rc])
        if key not in hist:
            hist[key] = {}
        counts = hist[key]
        label = T[i][0]
        if label not in counts:
            counts[label] = 1
        else:
            counts[label] += 1
    return hist
```

```
def HistogramTuning(B, D, P, T, Ptest, test_T):
    pmin = np.amin(np.vstack((P[:,:D],Ptest[:,:D])),axis = 0)
    pmax = np.amax(np.vstack((P[:,:D],Ptest[:,:D])),axis = 0)

    hist = BuildHistogram(B,T,P[:,:D],pmin,pmax)
    label_training = np.zeros((len(T),1)).astype('int8')
    for i,p in enumerate(P[:,:D]):
        label_training[i] = ApplyHistogramClassifier(p,B,pmin,pmax,hist)

    label_testing = np.zeros((len(test_T),1)).astype('int8')
    for i,p in enumerate(Ptest[:,:D]):
        label_testing[i] = ApplyHistogramClassifier(p,B,pmin,pmax,hist)

    unique, counts = np.unique(label_testing, return_counts=True)
    unknown_frac = 0.0
    if unique[0] == -1:
        unknown_frac = counts[0] / sum(counts)

    return (accuracy_score(label_training, T), accuracy_score(label_testing,test_T), unknown_frac)
```