CSE 802: Pattern Recognition & Analysis

Project Report

Abhinay Devapatla

Table of Contents

Introduction

This project is mainly meant to reinforce concepts covered in this course. The goal was to witness classification concepts such as curse of dimensionality and overfitting firsthand and see their effect on a dataset. Additionally, this project is meant to research classifiers not covered in this course and compare their effectiveness.

Description of Dataset

I found this dataset on Kaggle named the Mobile Price Classification problem [1]. The dataset was formulated as a classification problem for people to attempt, which is perfect for the scope of this project. According to the context, Bob has decided to start his own mobile company and wants to estimate the price of the phones. He collected sales data of various competitors' phones and wants to find a correlation between the phone's features and the phone's selling price. The estimated selling price has many ranges, and the goal is to develop a model to classify patterns to the appropriate price range.

The provided dataset is a file named "train.csv", I renamed it to "supervised.csv" for my own convenience. The csv file has 20 input features: battery power, bluetooth, clock speed, has dual sim card capability or not, front camera mega pixels, has 4G or not, internal memory in gigabytes, mobile depth in cm, weight of mobile phone, number of core processors, primary camera mega pixels, pixel resolution height, pixel resolution width, random access memory in megabytes, screen height of mobile in cm, screen width of mobile in cm, single charge battery lifespan, has 3G or not, has touch screen or not, has Wi-Fi or not. The output classes are split into 4 price ranges: 0 (low cost), 1 (medium cost), 2 (high cost), 3 (very high cost). There are 2000 patterns, and each pattern has already been assigned a label. Each output class has 500 patterns. There is no missing data, and the data is not normalized. To summarize, this classification problem is a 20-dimensional, 4-class problem.

Description of Analysis Conducted

The classifiers I used are Maximum Likelihood (MLE), Parzen Window Estimation, k-nearest neighbors (k-nn), Random Forest, Support Vector Machine, and Logistical Regression. I wrote custom code for MLE, Parzen Window Estimation, k-nn, Sequential Forward Search, Sequential Forward Floating Search, Principal Component Analysis, and Multiple Discriminant Analysis. I used Python libraries for Random Forest, Support Vector Machine, and Logistical Regression. For each classifier, I used 80% of the data for training and 20% for testing. I ensured the training set had the same amount of each class for the most accurate results.

MLE

I trained my model to estimate the mean and variance (the parameters) of a gaussian distribution. After developing the classifier, I ran Sequential Forward Search and Sequential Forward Floating Search to find the best feature sets. I chose a feature set that performed well and had parameters of value to compare against Principal Component Analysis and Multiple Discriminant Analysis. To ensure consistent results, I ran my results 3 times with shuffled data to avoid bias.

Parzen Window

I trained my model on 3 different Gaussian kernel heights: h = 0.1, h = 1, and h = 10. I decided to use all the features to see how accurate my classifier would be and compared it against Principal Component Analysis and Multiple Discriminant Analysis. Due to the computational complexity, I opted against running Sequential Forward Search or Sequential Forward Floating Search. To ensure consistent results, I ran my results 3 times for each height with shuffled data to avoid bias.

K-nn

I trained my model on 3 different k-values: k = 1, k = 5, and k = 10. I decided to use the feature set that MLE estimation deemed to be a good fit to see if MLE's results would work well on a different classifier. Additionally, I ran Principal Component Analysis and Multiple Discriminant Analysis. Due to the computational complexity, I opted against running Sequential Forward Search or Sequential Forward Floating Search. To ensure consistent results, I ran my results 3 times for each k-value with shuffled data to avoid bias.

Random Forest

I trained my model on 3 different n-sizes: n = 10, n = 100, n = 1000. I used the Python sklearn library to utilize the pre-built RandomForestClassifier. The RandomForestClassifier function inherently shuffles the data, so I ran it 3 times for each size n to avoid bias. While Random Forest does not inherently do feature selection, it does create feature subsets, so I opted to not use Sequential Forward Search and Sequential Floating Forward Search. I ran Principal Component Analysis but was unable to run Multiple Discriminant Analysis, since the imported class could not support the imaginary value results from Multiple Discriminant Analysis.

Support Vector Machine

I trained my model on 3 types of svm: linear, rbf, and poly. I used the Python sklearn library to utilize the pre-built SVC. The SVC function inherently shuffles the data, so I ran it 3 times for each size n to avoid bias. I once again used all the features in the dataset to see how well the classifier would perform. I opted to not use Sequential Forward Search and Sequential Floating Forward Search due to computational complexity. I ran Principal Component Analysis but was unable to run Multiple Discriminant Analysis, since the imported class could not support the imaginary value results from Multiple Discriminant Analysis.

Logistical Regression

I trained my model on 3 different maximum iteration values: max_iter = 1000, max_iter = 10000, and max_iter = 100000. I used the Python sklearn library to utilize the pre-built LogisticRegression. The LogisticRegression function inherently shuffles the data, so I ran it 3 times for each size n to avoid bias. I once again used all the features in the dataset to see how well the classifier would perform. I opted to not use Sequential Forward Search and Sequential Floating Forward Search due to computational complexity. I ran Principal Component Analysis but was unable to run Multiple Discriminant Analysis, since the imported class could not support the imaginary value results from Multiple Discriminant Analysis.

Presentation of Results

Figures 1 and 2, below, show the effect PCA and MDA had on the dataset during one of the iterations of the classifiers.

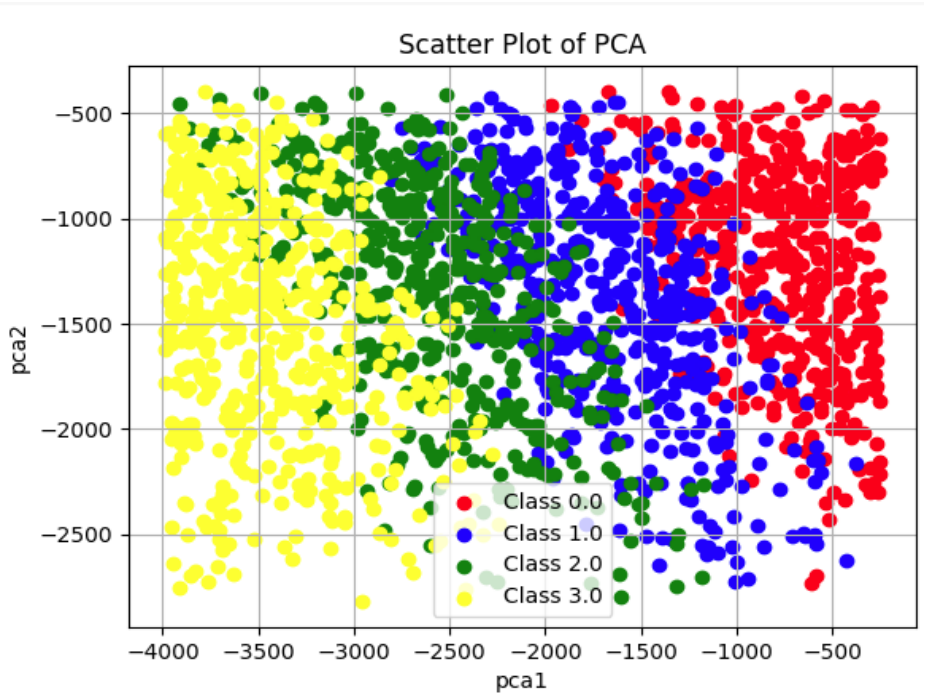


Figure 1: PCA

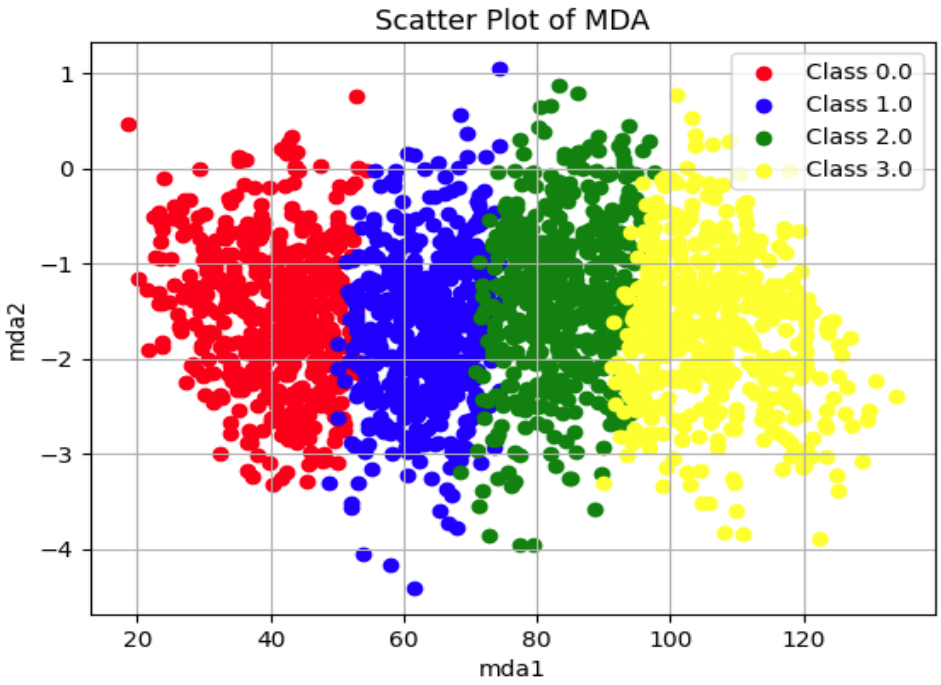


Figure 2: MDA

MLE Results

Table 1, below, gives the average error rate and variance for MLE. The following sections show the data at each iteration. The feature set referred to as "chosen feature set" are the following features: {'battery_power', 'clock_speed', 'dual_sim', 'int_memory', 'mobile_wt', 'px_height', 'px_width', 'ram', 'wifi', 'price_range'}.

	Error Rate Mean	Error Rate Variance
Chosen feature set	0.059666667	0.00083333333
PCA features	0.23766667	0.00018633333
MDA features	0.062333333	0.00017633333

Table 1: MLE Results

MLE Iteration #1

Figures 3 and 4, below, show the accuracy results of SFS and SFS for every feature set of length "k", with k being a value ranging from 1 to 20. **Figures 5-7**, below, show the confusion matrices and error rates for the feature set I chose, the PCA features, and the MDA features.

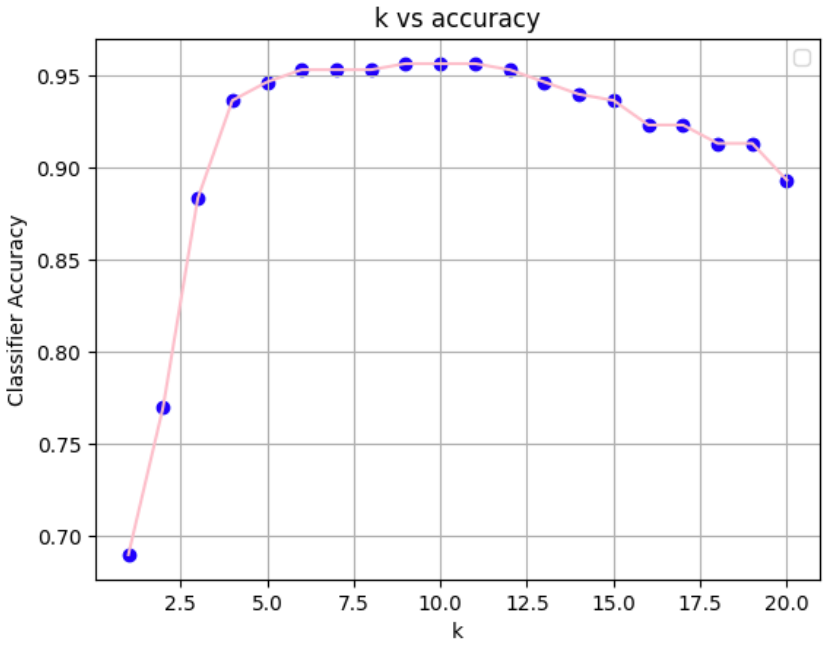


Figure 3: MLE SFS

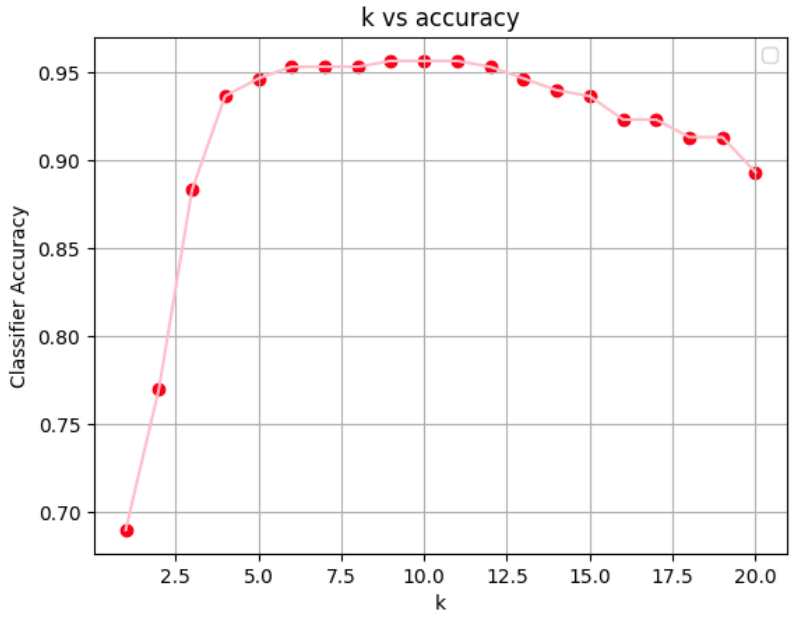


Figure 4: MLE SFFS

```
(array([[96,  4,  0,  0],
        [ 4, 95,  1,  0],
        [ 0,  4, 96,  0],
        [ 0,  0,  3, 97]]),
 0.043333333333333335)
```

Figure 5: MLE on my chosen feature set

```
(array([[81, 19,  0,  0],
        [ 6, 76, 18,  0],
        [ 0, 17, 68, 15],
        [ 0,  0, 19, 81]]),
 0.25)
```

Figure 6: MLE on PCA features

```
(array([[95,  5,  0,  0],
        [ 3, 94,  3,  0],
        [ 0,  7, 90,  3],
        [ 0,  0,  3, 97]]),
 0.07)
```

Figure 7: MLE on MDA features

MLE Iteration #2

Figures 8 and 9, below, show the accuracy results of SFS and SFS for every feature set of length "k", with k being a value ranging from 1 to 20. **Figures 10-12**, below, show the confusion matrices and error rates for the feature set I chose, the PCA features, and the MDA features.

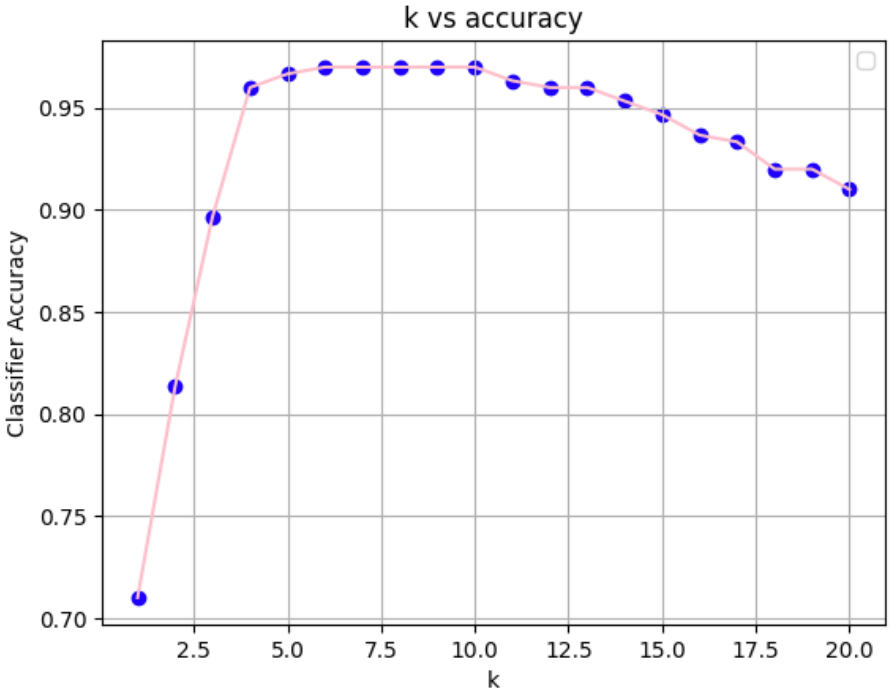


Figure 8: MLE SFS

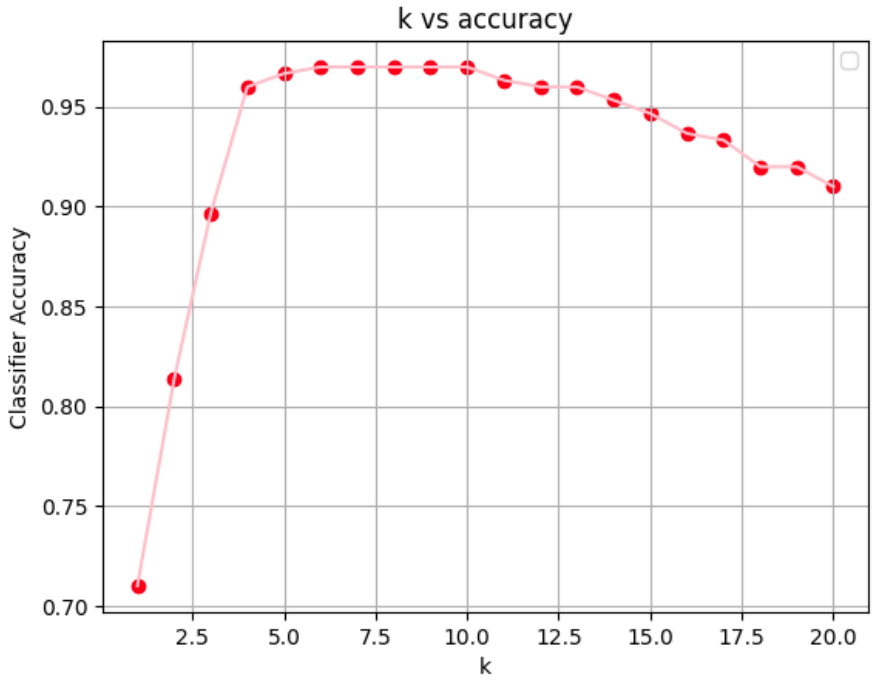


Figure 9: MLE SFFS

```
(array([[98,  2,  0,  0],
        [ 1, 95,  4,  0],
        [ 0,  3, 94,  3],
        [ 0,  0,  4, 96]]),
 0.043333333333333335)
```

Figure 10: MLE on my chosen feature set

```
(array([[87, 13,  0,  0],
        [ 9, 76, 15,  0],
        [ 0, 14, 70, 16],
        [ 0,  0, 14, 86]]),
 0.22333333333333333)
```

Figure 11: MLE PCA features

```
(array([[98,  2,  0,  0],
        [ 1, 97,  2,  0],
        [ 0,  4, 91,  5],
        [ 0,  0,  2, 98]]),
 0.04666666666666667)
```

Figure 12: MLE MDA features

MLE Iteration #3

Figures 13 and 14, below, show the accuracy results of SFS and SFS for every feature set of length "k", with k being a value ranging from 1 to 20. **Figures 15-17**, below, show the confusion matrices and error rates for the feature set I chose, the PCA features, and the MDA features.

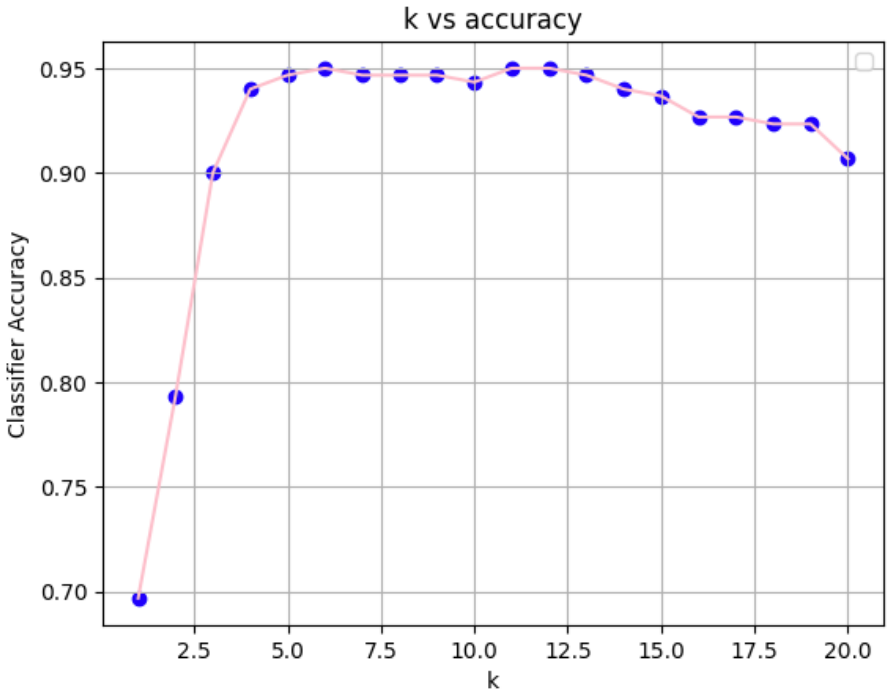


Figure 13: MLE SFS

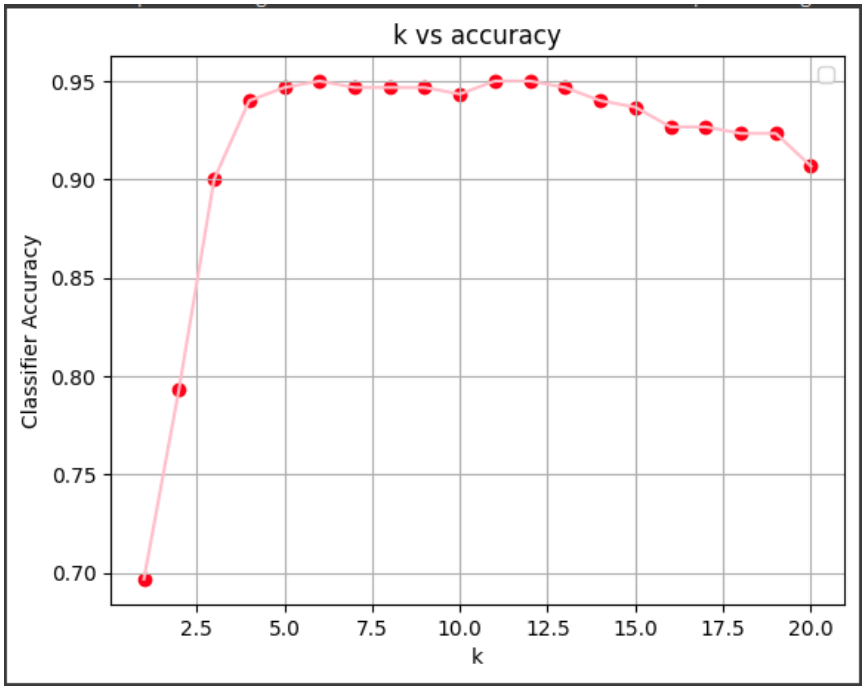


Figure 14: MLE SFFS

```
(array([[95,  5,  0,  0],
        [ 6, 90,  4,  0],
        [ 0,  4, 87,  9],
        [ 0,  0,  5, 95]]),
 0.09333333333333334)
```

Figure 15: MLE on my chosen features

```
(array([[89, 11,  0,  0],
        [13, 79,  8,  0],
        [ 0, 25, 60, 15],
        [ 0,  0, 13, 87]]),
 0.24)
```

Figure 16: MLE on PCA features

```
(array([[100,   0,   0,   0],
        [  6,  92,   2,   0],
        [  0,   8,  87,   5],
        [  0,   0,   4,  96]]),
 0.07)
```

Figure 17: MLE on MDA features

Parzen Window Results

The following sections show the data collected at each window size.

Window h = 0.1

Table 2, below, shows the average error rate and variance for h = 0.1. **Figures 18-20**, below, show the confusion matrices and error rates for a feature set containing all the features, PCA features, and MDA features for h = 0.1 over 3 iterations.

	Error Rate Mean	Error Rate Variance
All features	0.67	0
PCA features	0.67	0
MDA features	0.073666667	0.00043333333

Table 2: h = 0.1 Results

```
(array([[100,   0,   0,   0],
        [100,   0,   0,   0],
        [100,   0,   0,   0],
        [100,   0,   0,   0]]),
 0.6666666666666666)
```

```
(array([[100,   0,   0,   0],
        [100,   0,   0,   0],
        [100,   0,   0,   0],
        [100,   0,   0,   0]]),
 0.6666666666666666)
```

```
(array([[100,   0,   0,   0],
        [100,   0,   0,   0],
        [100,   0,   0,   0],
        [100,   0,   0,   0]]),
 0.6666666666666666)
```

Figure 18: All features

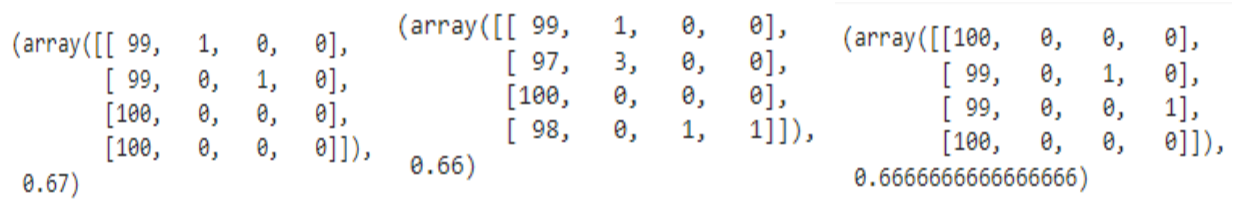


Figure 19: PCA features

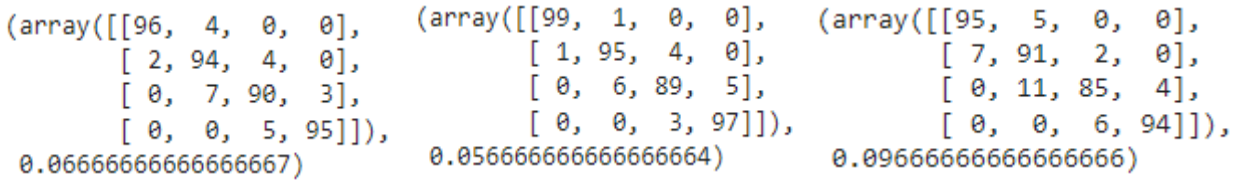


Figure 20: MDA features

Window h = 1

Table 3, below, shows the average error rate and variance for h = 1. **Figures 21-23**, below, show the confusion matrices and error rates for a feature set containing all the features, PCA features, and MDA features for h = 1 over 3 iterations.

	Error Rate Mean	Error Rate Variance
All features	0.67	0
PCA features	0.41666667	3.3333333E-5
MDA features	0.051	0.000112

Table 3: h = 1 Results

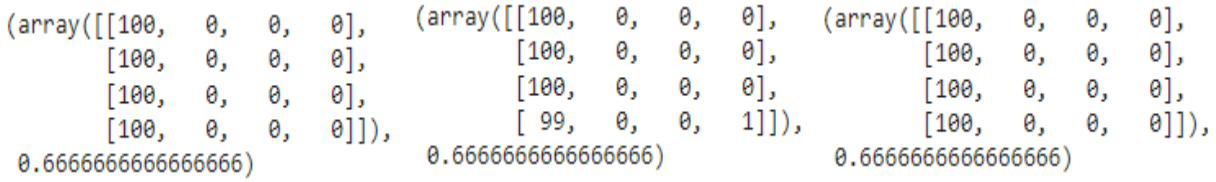


Figure 21: All features

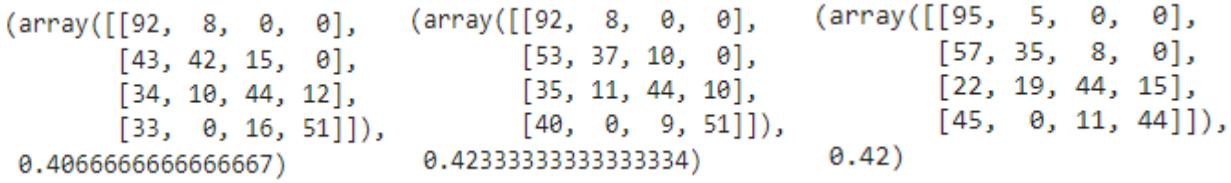


Figure 22: PCA features

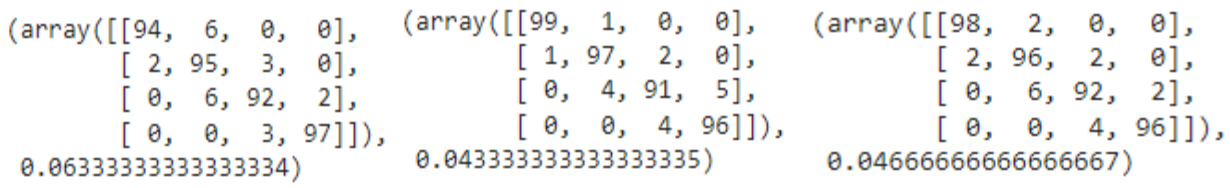


Figure 23: MDA features

Window h = 10

Table 4, below, shows the average error rate and variance for h = 10. **Figures 24-26**, below, show the confusion matrices and error rates for a feature set containing all the features, PCA features, and MDA features for h = 10 over 3 iterations.

	Error Rate Mean	Error Rate Variance
All features	0.11766667	0.0011053333
PCA features	0.26866667	0.00049633333
MDA features	0.052333333	0.00024633333

Table 3: h = 10 Results

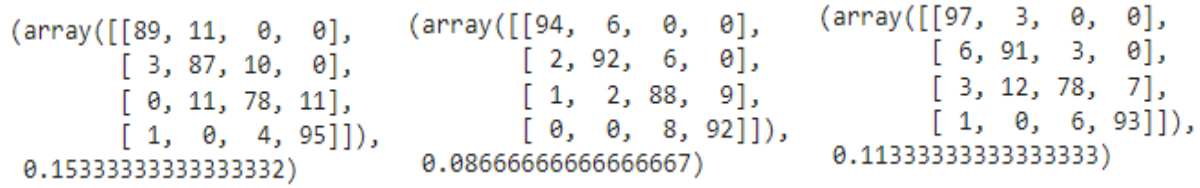


Figure 24: All features

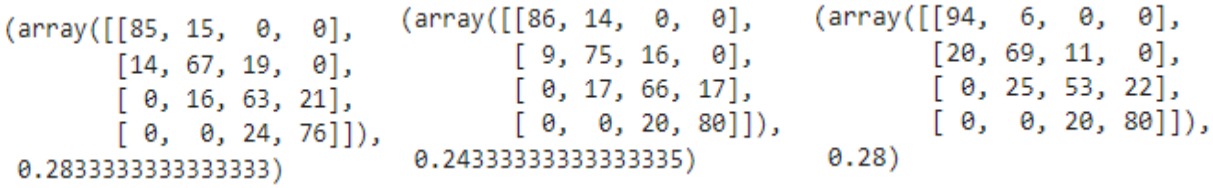


Figure 25: PCA features

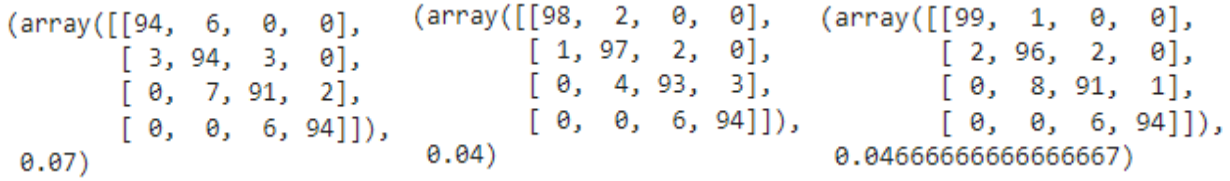


Figure 26: MDA features

K-nn Results

The following sections show the data collected at each k-value. The feature set referred to as "chosen feature set" are the following features: {'battery_power', 'clock_speed', 'dual_sim', 'int_memory', 'mobile_wt', 'px_height', 'px_width', 'ram', 'wifi', 'price_range'}.

K-nn: k = 1

Table 5, below, shows the average error rate and variance for k = 1. **Figures 27-29**, below, show the confusion matrices and error rates for a feature set containing all the features, PCA features, and MDA features for k = 1 over 3 iterations.

	Error Rate Mean	Error Rate Variance
Chosen feature set	0.113	0.0013
PCA features	0.27666667	0.00042233333
MDA features	0.078666667	0.00036633333

Table 5: k = 1 Results

```
(array([[89, 11,  0,  0],
        [ 3, 87, 10,  0],
        [ 0, 11, 78, 11],
        [ 0,  0,  4, 96]]),
 0.15333333333333332)
```

```
(array([[94,  6,  0,  0],
        [ 2, 92,  6,  0],
        [ 0,  2, 89,  9],
        [ 0,  0,  9, 91]]),
 0.08333333333333333)
```

```
(array([[97,  3,  0,  0],
        [ 6, 91,  3,  0],
        [ 0, 12, 81,  7],
        [ 0,  0,  6, 94]]),
 0.10333333333333333)
```

Figure 27: Chosen feature set

```
(array([[85, 15,  0,  0],
        [14, 65, 21,  0],
        [ 0, 16, 63, 21],
        [ 0,  0, 24, 76]]),
 0.29)
```

```
(array([[85, 15,  0,  0],
        [ 9, 73, 18,  0],
        [ 0, 17, 66, 17],
        [ 0,  0, 20, 80]]),
 0.25333333333333335)
```

```
(array([[94,  6,  0,  0],
        [20, 68, 12,  0],
        [ 0, 26, 52, 22],
        [ 0,  0, 20, 80]]),
 0.2866666666666667)
```

Figure 28: PCA features

```
(array([[94,  6,  0,  0],
        [ 2, 94,  4,  0],
        [ 0,  7, 90,  3],
        [ 0,  0,  5, 95]]),
 0.07333333333333333)
```

```
(array([[99,  1,  0,  0],
        [ 1, 93,  6,  0],
        [ 0,  6, 89,  5],
        [ 0,  0,  3, 97]]),
 0.06333333333333334)
```

```
(array([[94,  6,  0,  0],
        [ 7, 91,  2,  0],
        [ 0, 11, 85,  4],
        [ 0,  0,  6, 94]]),
 0.1)
```

Figure 29: MDA features

K-nn: k = 5

Table 6, below, shows the average error rate and variance for k = 5. **Figures 30-32**, below, show the confusion matrices and error rates for a feature set containing all the features, PCA features, and MDA features for k = 5 over 3 iterations.

	Error Rate Mean	Error Rate Variance
Chosen feature set	0.075666667	0.0019063333
PCA features	0.26133333	0.00019633333
MDA features	0.053	0.0001

Table 6: k = 5 Results

```
(array([[90, 10,  0,  0],
        [ 3, 93,  4,  0],
        [ 0, 11, 82,  7],
        [ 0,  0,  5, 95]]),
 0.11666666666666667)
```

```
(array([[95,  5,  0,  0],
        [ 3, 91,  6,  0],
        [ 0,  3, 90,  7],
        [ 0,  0,  7, 93]]),
 0.08)
```

```
(array([[100,   0,   0,   0],
        [  6,  91,   3,   0],
        [  0,  12,  84,   4],
        [  0,   0,   4,  96]]),
 0.08333333333333333)
```

Figure 30: Chosen feature set

```
(array([[85, 15,  0,  0],
        [12, 71, 17,  0],
        [ 0, 19, 67, 14],
        [ 0,  0, 18, 82]]),
 0.25666666666666665)
```

```
(array([[87, 13,  0,  0],
        [12, 72, 16,  0],
        [ 0, 17, 66, 17],
        [ 0,  0, 13, 87]]),
 0.25)
```

```
(array([[91,  9,  0,  0],
        [19, 74,  7,  0],
        [ 0, 27, 52, 21],
        [ 0,  0, 13, 87]]),
 0.27666666666666667)
```

Figure 31: PCA features

```
(array([[96,  4,  0,  0],
        [ 2, 94,  4,  0],
        [ 0,  4, 94,  2],
        [ 0,  0,  3, 97]]),
 0.05333333333333334)
```

```
(array([[99,  1,  0,  0],
        [ 1, 97,  2,  0],
        [ 0,  4, 91,  5],
        [ 0,  0,  3, 97]]),
 0.04333333333333335)
```

```
(array([[97,  3,  0,  0],
        [ 3, 93,  4,  0],
        [ 0,  6, 91,  3],
        [ 0,  0,  4, 96]]),
 0.06333333333333334)
```

Figure 32: MDA Features

K-nn: k = 10

Table 7, below, shows the average error rate and variance for k = 1. **Figures 33-35**, below, show the confusion matrices and error rates for a feature set containing all the features, PCA features, and MDA features for k = 1 over 3 iterations.

	Error Rate Mean	Error Rate Variance
Chosen feature set	0.086333333	0.00023333333
PCA features	0.25433333	5.3333333E-6
MDA features	0.055666667	6.5333333E-5

Table 7: k = 10 Results

```
(array([[93,  7,  0,  0],
        [ 2, 91,  7,  0],
        [ 0,  7, 85,  8],
        [ 0,  0,  2, 98]]),
 0.10333333333333333)
```

```
(array([[96,  4,  0,  0],
        [ 1, 93,  6,  0],
        [ 0,  3, 89,  8],
        [ 0,  0,  4, 96]]),
 0.07333333333333333)
```

```
(array([[100,   0,   0,   0],
        [  6,  91,   3,   0],
        [  0,  10,  84,   6],
        [  0,   0,   2,  98]]),
 0.08333333333333333)
```

Figure 33: Chosen feature set

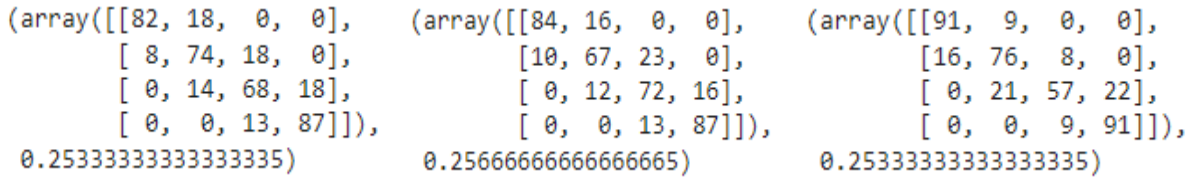


Figure 34: PCA features

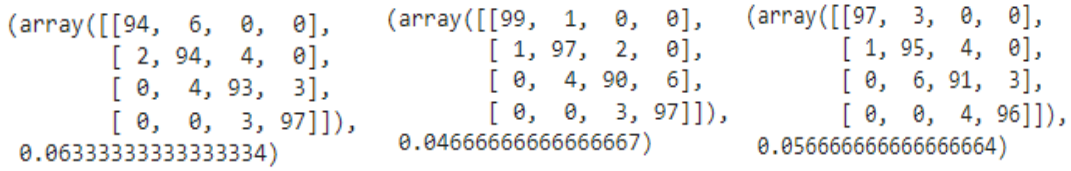


Figure 35: MDA features

Random Forest Results

The following sections show the data collected at each n value.

Random Forest: n = 10

Table 8, below, shows the average error rate and variance for n = 10. **Figures 36 and 37**, below, show the confusion matrices and ACCURACY rates for a feature set containing all the features and PCA features for n = 10 over 3 iterations.

	Error Rate Mean	**Error Rate Variance**
All features	0.1925	0.00011875
PCA features	0.2175	8.125E-5

Table 8: n = 10

```
(array([[89, 11,  0,  0],
        [10, 82,  7,  1],
        [ 1, 26, 65,  8],
        [ 1,  1, 13, 85]]),
 0.8025,
 ram              0.473064
 battery_power    0.068451
 px_width         0.060590
 px_height        0.056683
 mobile_wt        0.039680
 int_memory       0.033593
 talk_time        0.033364
 clock_speed      0.032385
 pc               0.031041
 m_dep            0.029494
 sc_h             0.027680
 sc_w             0.027000
 fc               0.024249
 n_cores          0.022194
 four_g           0.007973
 touch_screen     0.007408
 dual_sim         0.006642
 wifi             0.006364
 blue             0.006126
 three_g          0.006018
 dtype: float64)
```

```
(array([[96,  4,  0,  0],
        [ 7, 70, 23,  0],
        [ 1, 19, 70, 10],
        [ 0,  0, 16, 84]]),
 0.8,
 ram              0.472297
 battery_power    0.073566
 px_height        0.059548
 px_width         0.058653
 mobile_wt        0.040876
 pc               0.034294
 int_memory       0.033585
 talk_time        0.030515
 clock_speed      0.029475
 sc_w             0.029090
 sc_h             0.026985
 fc               0.024047
 n_cores          0.023878
 m_dep            0.022922
 dual_sim         0.009507
 wifi             0.006744
 touch_screen     0.006469
 four_g           0.006095
 three_g          0.005867
 blue             0.005584
 dtype: float64)
```

```
(array([[92,  8,  0,  0],
        [ 9, 76, 15,  0],
        [ 0, 21, 70,  9],
        [ 0,  0, 10, 90]]),
 0.82,
 ram              0.451381
 battery_power    0.076533
 px_height        0.065707
 px_width         0.059971
 talk_time        0.040490
 mobile_wt        0.040221
 int_memory       0.036220
 pc               0.032748
 sc_w             0.031309
 sc_h             0.028907
 clock_speed      0.028599
 m_dep            0.024524
 fc               0.024218
 n_cores          0.019868
 touch_screen     0.008163
 dual_sim         0.007572
 four_g           0.006665
 three_g          0.005814
 blue             0.005653
 wifi             0.005434
 dtype: float64)
```

Figure 36: All features

```
(array([[93,  7,  0,  0],
        [12, 69, 19,  0],
        [ 0, 19, 73,  8],
        [ 0,  0, 21, 79]]),
 0.785,
 pca1    0.745756
 pca2    0.254244
 dtype: float64)
```

```
(array([[87, 13,  0,  0],
        [13, 77, 10,  0],
        [ 0, 22, 65, 13],
        [ 0,  0, 13, 87]]),
 0.79,
 pca1    0.758277
 pca2    0.241723
 dtype: float64)
```

```
(array([[85, 15,  0,  0],
        [13, 76, 11,  0],
        [ 0, 18, 60, 22],
        [ 0,  0, 12, 88]]),
 0.7725,
 pca1    0.765826
 pca2    0.234174
 dtype: float64)
```

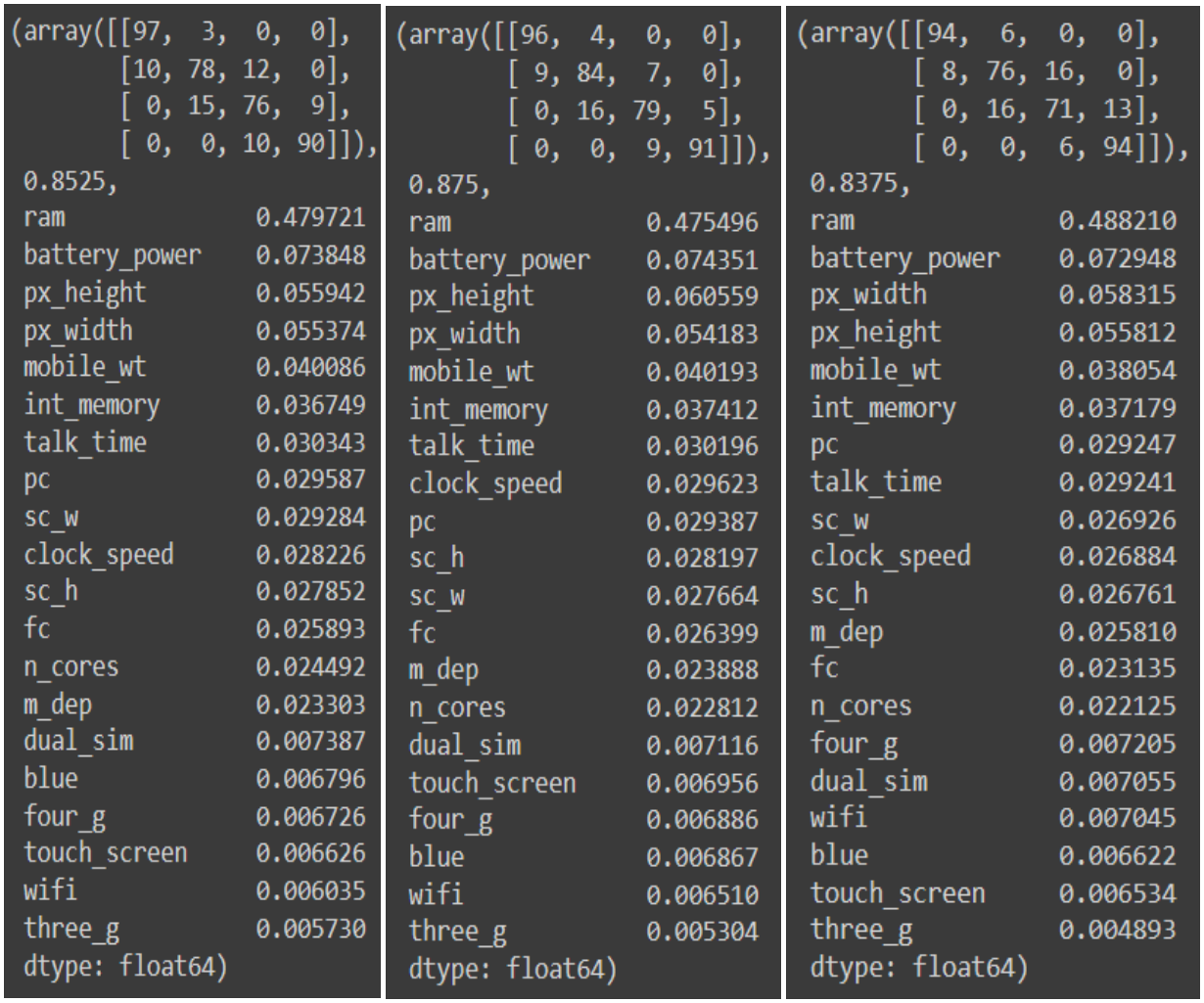
Figure 37: PCA features

Random Forest: n = 100

Table 9, below, shows the average error rate and variance for n = 100. **Figures 38 and 39**, below, show the confusion matrices and ACCURACY rates for a feature set containing all the features and PCA features for n = 100 over 3 iterations.

	Error Rate Mean	Error Rate Variance
All features	0.145	0.00035625
PCA features	0.21666667	0.00065833333

Table 9: n = 100



```
(array([[97,  3,  0,  0],
        [10, 78, 12,  0],
        [ 0, 15, 76,  9],
        [ 0,  0, 10, 90]]),
 0.8525,
 ram              0.479721
 battery_power    0.073848
 px_height        0.055942
 px_width         0.055374
 mobile_wt        0.040086
 int_memory       0.036749
 talk_time        0.030343
 pc               0.029587
 sc_w             0.029284
 clock_speed      0.028226
 sc_h             0.027852
 fc               0.025893
 n_cores          0.024492
 m_dep            0.023303
 dual_sim         0.007387
 blue             0.006796
 four_g           0.006726
 touch_screen     0.006626
 wifi             0.006035
 three_g          0.005730
 dtype: float64)
```

```
(array([[96,  4,  0,  0],
        [ 9, 84,  7,  0],
        [ 0, 16, 79,  5],
        [ 0,  0,  9, 91]]),
 0.875,
 ram              0.475496
 battery_power    0.074351
 px_height        0.060559
 px_width         0.054183
 mobile_wt        0.040193
 int_memory       0.037412
 talk_time        0.030196
 clock_speed      0.029623
 pc               0.029387
 sc_h             0.028197
 sc_w             0.027664
 fc               0.026399
 m_dep            0.023888
 n_cores          0.022812
 dual_sim         0.007116
 touch_screen     0.006956
 four_g           0.006886
 blue             0.006867
 wifi             0.006510
 three_g          0.005304
 dtype: float64)
```

```
(array([[94,  6,  0,  0],
        [ 8, 76, 16,  0],
        [ 0, 16, 71, 13],
        [ 0,  0,  6, 94]]),
 0.8375,
 ram              0.488210
 battery_power    0.072948
 px_width         0.058315
 px_height        0.055812
 mobile_wt        0.038054
 int_memory       0.037179
 pc               0.029247
 talk_time        0.029241
 sc_w             0.026926
 clock_speed      0.026884
 sc_h             0.026761
 m_dep            0.025810
 fc               0.023135
 n_cores          0.022125
 four_g           0.007205
 dual_sim         0.007055
 wifi             0.007045
 blue             0.006622
 touch_screen     0.006534
 three_g          0.004893
 dtype: float64)
```

Figure 38: All features

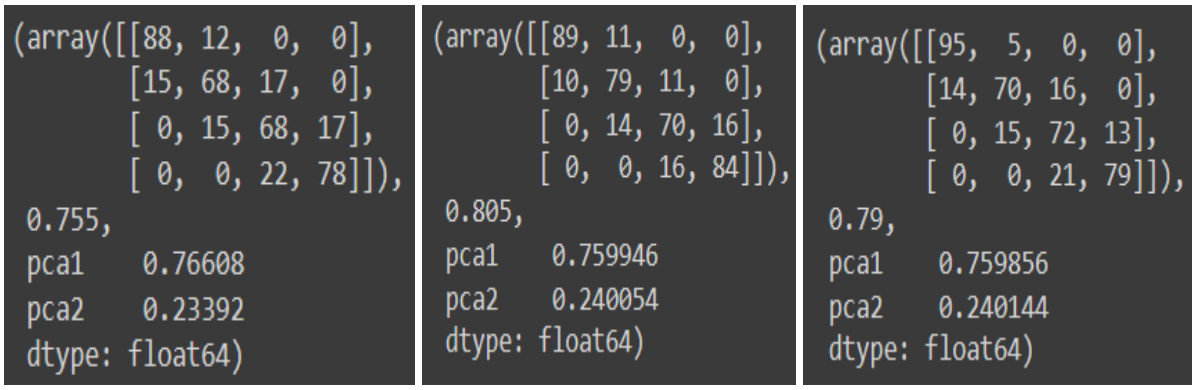


Figure 39: PCA features

Random Forest: n = 1000

Table 10, below, shows the average error rate and variance for n = 1000. **Figures 40 and 41**, below, show the confusion matrices and ACCURACY rates for a feature set containing all the features and PCA features for n = 1000 over 3 iterations.

	Error Rate Mean	Error Rate Variance
All features	0.1325	0.00030625
PCA features	0.20416667	0.0014145833

Table 10: n = 1000

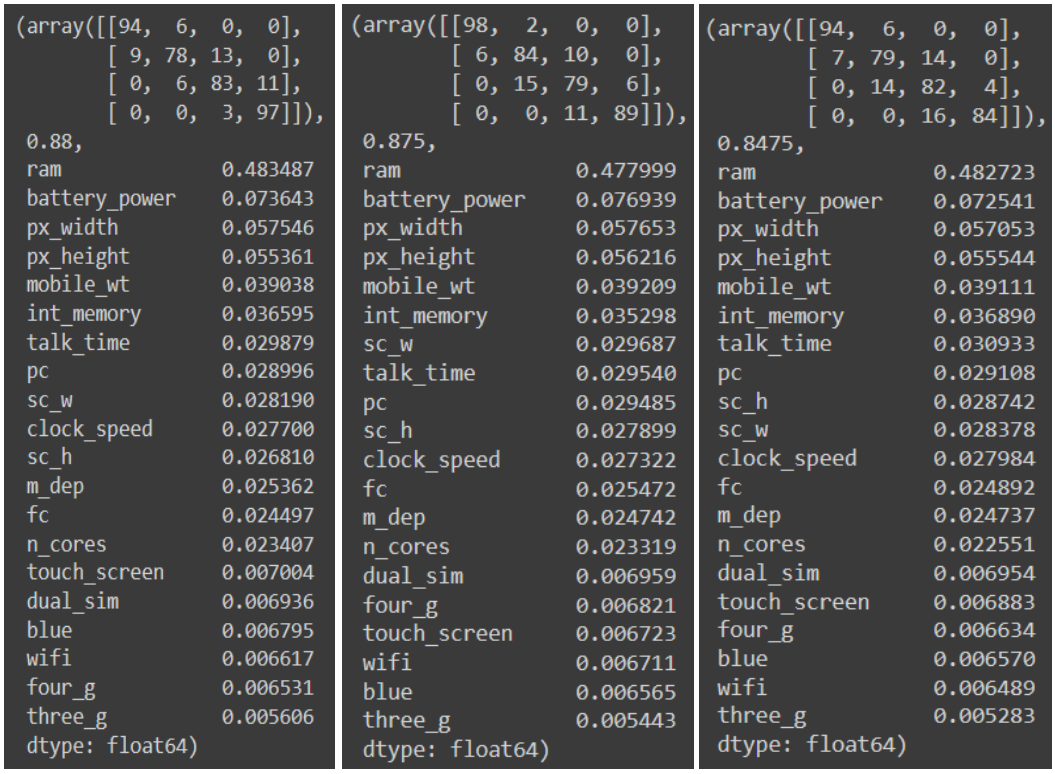


Figure 40: All features

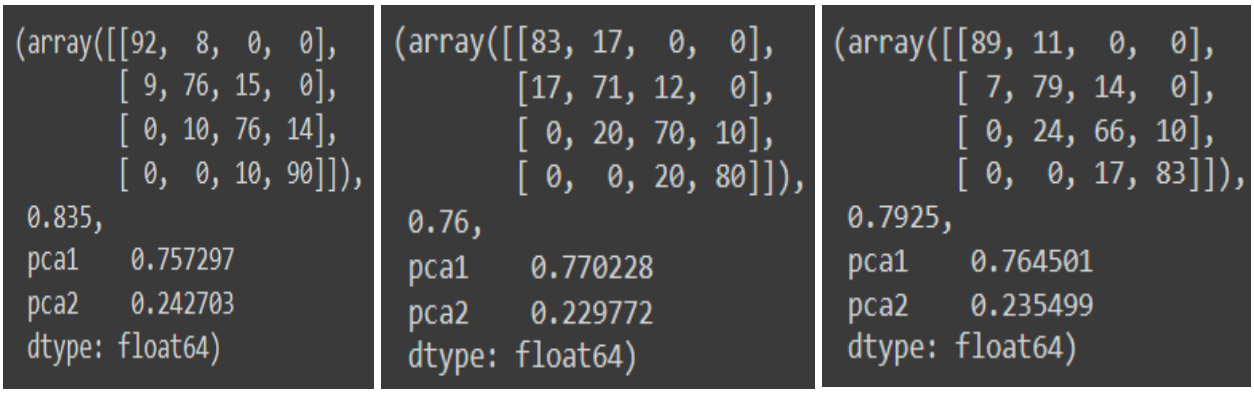


Figure 41: PCA features

Support Vector Machine Results

The following sections show the data collected with kernel functions being linear, rbf, poly.

SVM: Linear

Table 11, below, shows the average error rate and variance for a linear kernel. **Figures 42 and 43**, below, show the confusion matrices and ACCURACY rates for a feature set containing all the features and PCA features for a linear kernel over 3 iterations.

	Error Rate Mean	Error Rate Variance
All features	0.030833333	5.8333333E-5
PCA features	0.20416667	0.012829004

Table 11: linear

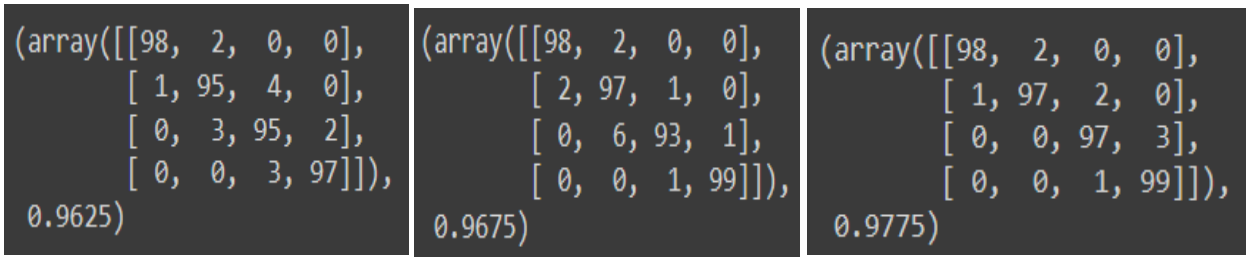


Figure 42: All features

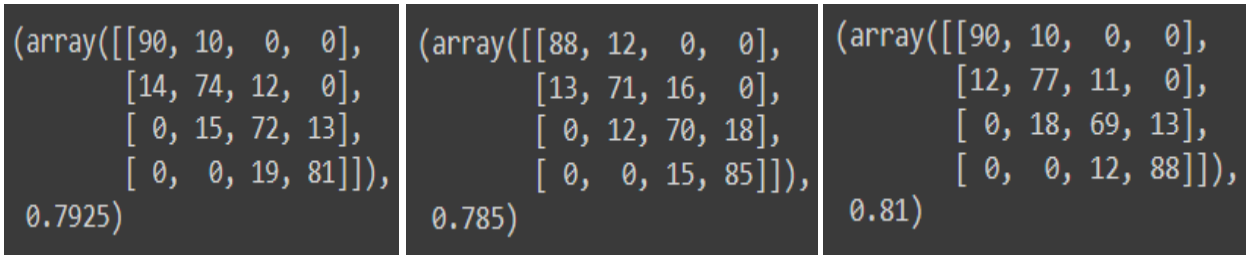


Figure 43: PCA features

SVM: rbf

Table 12, below, shows the average error rate and variance for a rbf kernel. **Figures 44 and 45**, below, show the confusion matrices and ACCURACY rates for a feature set containing all the features and PCA features for a rbf kernel over 3 iterations.

	Error Rate Mean	Error Rate Variance
All features	0.045	6.25E-6
PCA features	0.19916667	7.7083333E-5

Table 12: rbf

```
(array([[98,  2,  0,  0],
        [ 4, 95,  1,  0],
        [ 0,  8, 90,  2],
        [ 0,  0,  2, 98]]),
 0.9525)
```

```
(array([[98,  2,  0,  0],
        [ 0, 99,  1,  0],
        [ 0,  6, 87,  7],
        [ 0,  0,  1, 99]]),
 0.9575)
```

```
(array([[100,   0,   0,   0],
        [  2,  97,   1,   0],
        [  0,   8,  89,   3],
        [  0,   0,   4,  96]]),
 0.955)
```

Figure 44: All features

```
(array([[90, 10,  0,  0],
        [10, 72, 18,  0],
        [ 0, 12, 72, 16],
        [ 0,  0, 17, 83]]),
 0.7925)
```

```
(array([[86, 14,  0,  0],
        [ 9, 83,  8,  0],
        [ 0, 17, 71, 12],
        [ 0,  0, 16, 84]]),
 0.81)
```

```
(array([[89, 11,  0,  0],
        [16, 72, 12,  0],
        [ 0, 14, 73, 13],
        [ 0,  0, 14, 86]]),
 0.8)
```

Figure 45: PCA features

SVM: poly

Table 13, below, shows the average error rate and variance for a poly kernel. **Figures 46 and 47**, below, show the confusion matrices and ACCURACY rates for a feature set containing all the features and PCA features for a poly kernel over 3 iterations.

	Error Rate Mean	Error Rate Variance
All features	0.045	1.875E-5
PCA features	0.18916667	0.00051458333

Table 13: poly

```
(array([[98,  2,  0,  0],
        [ 2, 97,  1,  0],
        [ 0,  4, 91,  5],
        [ 0,  0,  5, 95]]),
 0.9525)
```

```
(array([[99,  1,  0,  0],
        [ 4, 93,  3,  0],
        [ 0,  7, 91,  2],
        [ 0,  0,  2, 98]]),
 0.9525)
```

```
(array([[99,  1,  0,  0],
        [ 2, 95,  3,  0],
        [ 0,  3, 93,  4],
        [ 0,  0,  3, 97]]),
 0.96)
```

Figure 46: All features

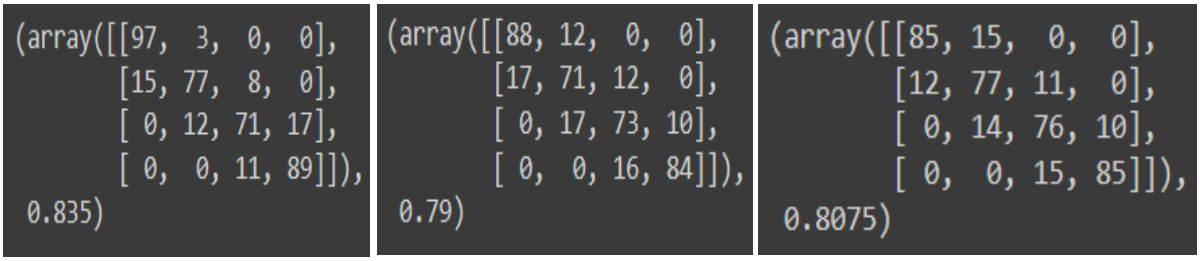


Figure 47: PCA features

Logistical Regression Results

The following sections show the data collected with varying max iteration values.

Logistical Regression: max_iter = 1000

Table 14, below, shows the average error rate and variance for a max iteration of 1000. **Figures 48 and 49**, below, show the confusion matrices and ACCURACY rates for a feature set containing all the features and PCA features for a max iteration of 1000 over 3 iterations.

	Error Rate Mean	Error Rate Variance
All features	0.2975	0.000925
PCA features	0.20083333	3.3333333E-5

Table 14: max_iter = 1000

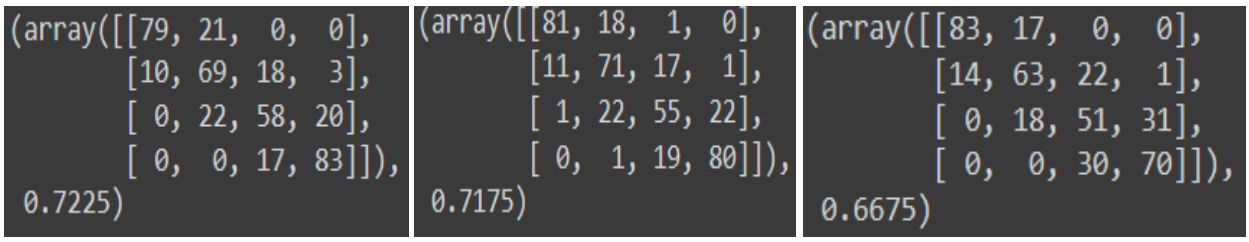


Figure 48: All features

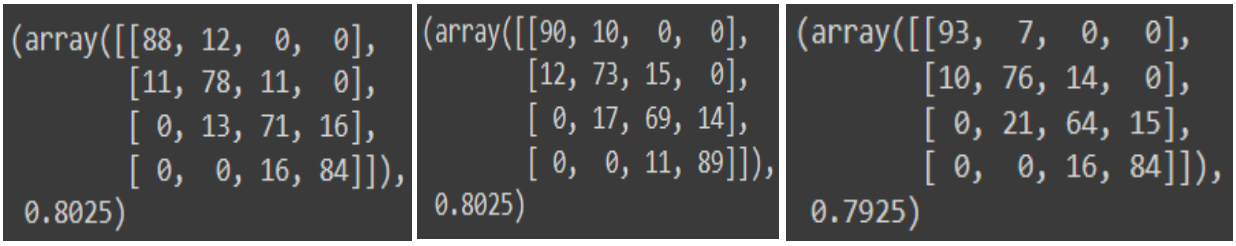


Figure 49: PCA features

Logistical Regression: max_iter = 10000

Table 15, below, shows the average error rate and variance for a max iteration of 10000. **Figures 50 and 51**, below, show the confusion matrices and ACCURACY rates for a feature set containing all the features and PCA features for a max iteration of 10000 over 3 iterations.

	Error Rate Mean	Error Rate Variance
All features	0.235	5.625E-5
PCA features	0.2	0.00030625

Table 15: max_iter = 10000

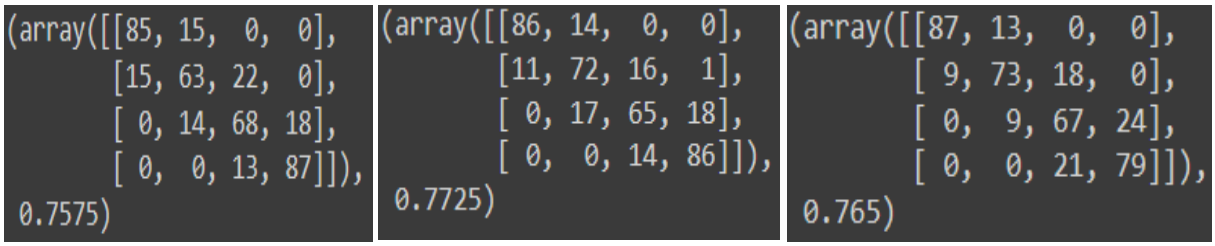


Figure 50: All features

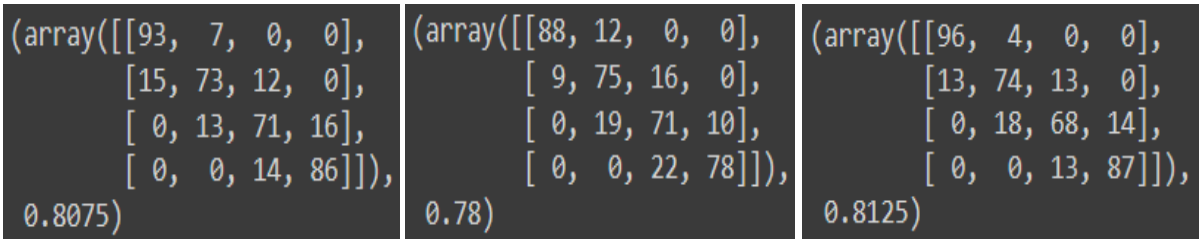


Figure 51: PCA features

Logistical Regression: max_iter = 100000

Table 16, below, shows the average error rate and variance for a max iteration of 100000. **Figures 52 and 53**, below, show the confusion matrices and ACCURACY rates for a feature set containing all the features and PCA features for a max iteration of 100000 over 3 iterations.

	Error Rate Mean	Error Rate Variance
All features	0.205	0.000325
PCA features	0.20916667	0.00041458333

Table 16: max_iter = 100000

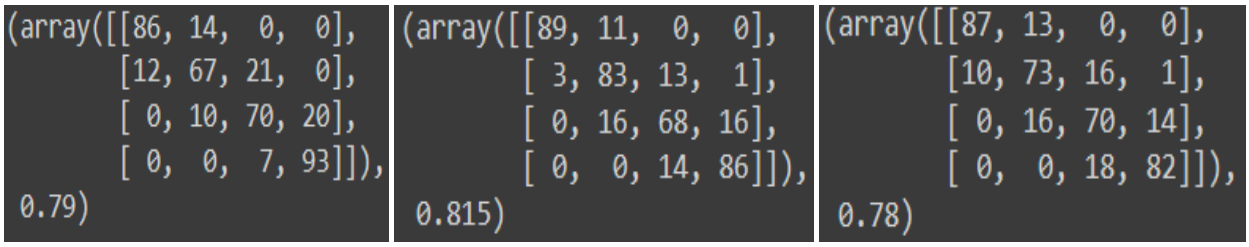


Figure 52: All features

```
(array([[89, 11,  0,  0],
        [ 9, 76, 15,  0],
        [ 0, 16, 63, 21],
        [ 0,  0, 21, 79]]),
 0.7675)
```

```
(array([[88, 12,  0,  0],
        [16, 72, 12,  0],
        [ 0, 18, 68, 14],
        [ 0,  0,  8, 92]]),
 0.8)
```

```
(array([[87, 13,  0,  0],
        [12, 76, 12,  0],
        [ 0, 14, 76, 10],
        [ 0,  0, 17, 83]]),
 0.805)
```

Figure 53: PCA features

Analysis of Results

In the MLE classifier, SFS and SFFS clearly show the effects of the curse of dimensionality. The accuracy of a feature set peaks at around 10 features and decreases after that. Additionally, SFS and SFFS produced nearly identical results. PCA had the worst error rate (around 20%). While the chosen feature set and MDA had around the same error rate (around 6%).

In the Parzen window classifier, as the h-value increased the accuracies also increased. Using all the features did not result in good classifier accuracy but had nearly the same accuracy as PCA. MDA had by far the best classifier accuracy in this situation.

In the k-nn classifier, as the k value increased the accuracy stated relatively the same. The chosen feature set and MDA had relatively the same good accuracy, while PCA had an abysmal accuracy.

In the Random Forest classifier, as the n-value increased the accuracy of using all the features improved while PCA stated about the same.

In the SVM classifier, the accuracy of using all the features was incredibly high (around 96%). The different kernel functions did not really produce differing results. PCA still performed at around 80% accuracy, like the previous classifiers.

In Logistical Regression, increasing the maximum iterations did not affect the results too much. However, this is the only case where PCA is better than the feature set containing all the features. The accuracy was still low (around 80%) but still notable how it performed better by 1-2%.

Overall, MDA works extremely well on any classifier and has relatively the same accuracy throughout. In most scenarios, a feature set not containing all the features performed better except for in SVM. PCA was by far the worst accuracy, but 80% accuracy is not terrible.

Summary and Conclusions

Overall, I was able to reinforce the concepts I learned in class with concrete examples. I learned the true effect of the curse of dimensionality on a classifier. I learned the need for feature reduction not only for improved accuracy but also for computational efficiency. I was able to play around with classifiers not covered in this course and saw how they compared against the traditional classifiers derived from the Bayes Rule. If I had more time, I would want to compare the accuracy of another distribution estimated by MLE, run feature selection algorithms on all the classifiers, apply a whitening transformation before running the classifiers, and optimize my existing classifiers to be more efficient. Anyone can just import an open-source classifier and make a machine learning model, but as this course has taught us, there is a lot more to classification than just running a few lines.

References

[1] "Mobile Price Classification," *www.kaggle.com*. https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification/data