Homework 2
Anushka Nair, Bam Nakapakorn, and Madeline Luong

Study Datasets: BELL, CHILLER, and FREESTYLE fonts from given website.
BELL contains 956 observations/ instances and 412 features.
CHILLER contains 962 observations/ instances and 412 features.
FREESTYLE contains 956 observations/ instances and 412 features.

Preliminary treatment:
Unneeded excess features, non-numerical values, and non- defined classes discarded.
BELL_CLEAN contains 239 observations/ instances and 404 features.
CHILLER_CLEAN contains 238 observations / instances and 404 features.
FREESTYLE_CLEAN contains 239 observations/ instances and 404 features.

Then defining the three classes (CL1, CL2. And CL3) to each dataset and unionizing them into a full dataset. DATA then contains 716 observations / instances and 404 features.

Part 0
Computing the mean(where m1 = mean(x1)…mean(400)=m400) and standard deviation (s1= sd(x1)..sd(x400) =sd400) of the full data set. Here we standardize and scale the data. We want to standardize the features to the center at 0 and a standard deviation of 1 because different variables can be measured at different scales leading to bias.
We can see here that the standard deviation values are not one, so we will then normalize the data by the function $y_j = (x_j – m_j)/ s_j$. After standardizing the data, a columns standard deviation will be extracted. Column 3's standard deviation is one, confirming that the data set has been rescaled and standardized, which we then name SDATA. To view the table containing all means and standard deviation prior to the standardization refer to the appendix section.

We then compute the correlation matrix of the 400 features. Finding the correlation matrix will show us the correlation coefficients between the 400 features. We want to display the ten highest absolute correlation coefficient values (shown in table below). A correlation matrix only calculates the correlation between numeric features, so we took out all non-numeric data, leaving us with a 400 x 400 array.

Pixel Positions Xi, Xj, Pair		Correlation Coefficient
r19c16	r19c15	0.91627
r0c4	r0c3	0.91518
r15c16	r14c16	0.91107
r19c15	r19c14	0.90619
r15c17	r14c17	0.90612
r14c18	r13c18	0.90470
r11c18	r10c18	0.90164
r0c5	r0c4	0.89906
r12c1	r11c1	0.89357
r0c3	r0c2	0.88880

Part 1
1.0 - Creating a TRAINSET and TESTSET
Taking the SDATA set, we will classify CL to SETROW, where CL1 is equal to SETROW1, CL2 is equal to SETROW2, and CL3 is equal to SETROW3. We then separate each class into individual subsets of SETROW1, SETROW2, and SETROW3 to create the proper train and test set distribution. Each subset will then be split arbitrarily where TRAIN will be about 80% of the subset and TEST will be about 20% of the

subset. After each subset has been split into train and test sets, all train sets will be unionized into a full TRAINSET (trainsetcl1, trainsetcl2, and trainset cl3). The same will be done to the TESTSET (testsetcl1, testsetcl2, and testsetcl3).

1.1 - *Using the K- Nearest Neighbor algorithm (KNN) on the classification of CL1, CL2 , CL3.*

To apply the KNN algorithm, a matrix with the predictors in the TRAINSET is created labeled TRAIN_no. Similarly, a matrix containing the predictors in the TESTSET is created labeled TEST_no. Then two vectors are created that contain the class labels for the training observations and testing observations which are labeled TRAIN_label and TEST_label respectively. We then apply the knn() function on the TRAINSET and TESTSET to predict the CL classifications of each font based on grey level image intensity for pixel at each position. We set a random seed before applying knn() because if several observations are tied as nearest neighbor then R will break the tie. Here we used k = 12.

Percent correct classifications
Trainperf^{12} = 0.733
Testperf^{12} = 0.687

We can see that the testperf accuracy is lower compared to trainperf. This makes sense because in the case of trainperf, the knn algorithm is trained and tested on the same data, while in the case of testperf, the algorithm has not been exposed to the data it is testing. Additionally, our test set is too small to conclude that it would perform similarly on new data. As we change the k value, the train set error could decline, however the test error may not. As we increase the K value, the variance will decrease but there will be a higher bias in classifier. If we decrease the K-value we might overfit the boundaries. So further exploration is needed to understand if increasing or decreasing the k value will decrease the error rate.

1.2 - *Replicating KNN algorithm for K = 5,10,15,20,30,40,50 and 100 and visualizing it.*

By using the same prior KNN function, to replicate this for values 5,10,15,20,30,40, 50 and 100 a loop will be created to identify the percent accuracy. This will be done to both the train and the test set. Then plotted to visualize the best range of k value. The plot of the k values will indicate if the values will overfit between each set.

Percent accuracy of each K value
Testperf^{5}= 68.75%
Testperf^{10}= 66.66%
Testperf^{15}= 66.66%
Testper^{20}= 68.05%
Testperf^{30}= 61.80%
Testperf^{40}= 59.02%
Testperf^{50}= 58.33%
Testperf^{100}= 58.33%

Based on these values we can conclude that where k is [5 < k < 20] is at least 68% accuracy.

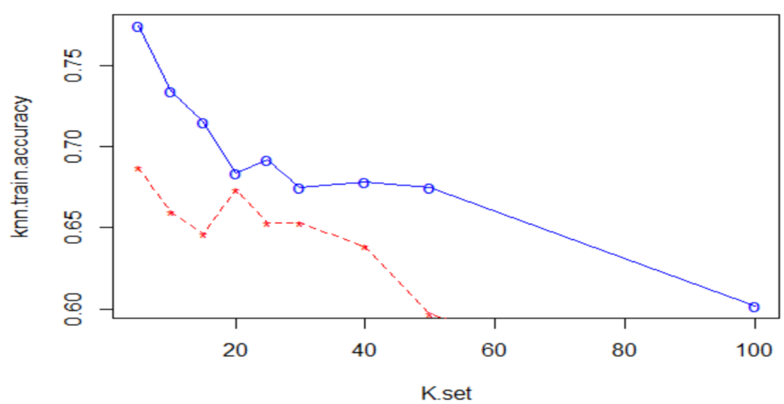

Figure 1:plot of percent accuracy with varying k with both test and train sets. A plot showing a negative linear correlation of percent accuracy which is obtained by using k = 5,10, 15, 20,30,40,50 100. The blue line is the trainset and the red line is the testset values. Each dot indicates the correlated k value and accuracy.

According to the curves of train and test sets, we can see that we have slightly overfit model, where the test error is higher than the train error by (3%). The discrepancy between the 2 curves show the magnitude of overfit. Allowing us to select knn =5 as the best fit of accuracy.

1.3 - *Identifying the "best" k value within the prior range.*

From the prior loop, we will run it again using a sequence function from the range [5:100] in sets of 5. Thus, testing it on 5,10,15,20,25…100. Which we can conclude that the "ideal" k value will be 5 with the percent accuracy of 68.75%

1.4 - *KNN algorithm using K = 5 and visualizing the respected correlation matrices and confidence intervals*

Based on the prior loop, we determine that k = 5. Then, we will run KNN algorithm on both the TESTSET and TRAINSET. After receiving the accuracy values, we will then do a 3x3 confusion matrix on both TESTSET and TRAINSEST to better visualize the classification for each class.

```
                 TRAIN_label
knn.predtrainbest CL1 CL2 CL3
              CL1 154  43  18
              CL2  24 132   8
              CL3  13  15 165
```

Train

	Test K = 5	PREDICTION CL1	CL2	CL3
TRUE	CL1	71.63%	20.00%	8.37%
	CL2	14.63%	80.49%	4.88%
	CL3	6.74%	7.77%	85.49%

```
                TEST_label
knn.predtestbest CL1 CL2 CL3
             CL1  36  16   4
             CL2   7  23   3
             CL3   5   9  41
```

Test

	Test K = 5	PREDICTION CL1	CL2	CL3
TRUE	CL1	64.29%	28.57%	7.14%
	CL2	21.21%	69.70%	9.09%
	CL3	9.09%	16.36%	74.55%

Figure 2: Confusion matrices and percentage accuracy of TESTSET and TRAINSET

Top left and bottom left are the confusion matrix for train set and test set, which the rows are the predictions and the columns are the actual values. Right side of the figure will be the corresponding accuracy percentage for each classifier.

Train set

Based on the figure, we can see that 154 instances of "CL1" are classified correctly as "CL1". Then, 132 Instances of "CL2" are also classified correctly as "CL2". "CL3" was classified correctly 165 instances. 61 instances of "CL1" were classified incorrectly as "CL2" or "CL3". 32 instances of "CL2" were classified incorrectly as either "CL1" or "CL3". Lastly, 28 instances of "CL3" were classified incorrectly as either "CL1" or "CL2". Based on the percentages we can see that "CL3" was classified with the highest accuracy of 85.5%, comparatively to "CL2" at 80.5% and "CL1" at 71.6%.

Testset

Based on the figure, we can see that 36 instances of "CL1" are classified correctly as "CL1". Then, 23 Instances of "CL2" are also classified correctly as "CL2". "CL3" was classified correctly 41 instances. 20 instances of "CL1" were classified incorrectly as "CL2" or "CL3". 10 instances of "CL2" were classified incorrectly as either "CL1" or "CL3". Lastly, 14 instances of "CL3" were classified incorrectly as either "CL1" or "CL2". Based on the percentages we can see that "CL3" was classified with the highest accuracy of 74.5% comparatively to "CL2" at 69.7% and "CL1" at 64.3%

Overall, we can see that "CL3" had the "best" prediction with the k value at 5.

1.5 - *Determining the confidence interval for both TESTSET and TRAINSET.*

After determining the confusion matrix, we seek to find the confidence intervals diagonals to determine they overlap or are nearly disjoint with each other.

Confidence interval
Trainset: 95% CI : (0.7531, 0.8199)
Testset: 95% CI : (0.6150, 0.7638)

We can see that the confidence intervals overlap with each other, which is evidence that quantities are close to each other. However, we cannot conclude that trainset quantile is greater than testset quantile.

1.6 - Creating the individual packages from the standardize full data set (SDATA).

We going to separate the data into 4 packs (PACK1, PACK2, PACK3, and PACK 4). Each pack will have 100 features corresponded to the 100 pixel intensities. We will divide the pack by rLcM as follows:

PACK 1: L = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
M = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
PACK 2: L = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
M = 10, 11, 12, 13, 14, 15, 16, 17, 18, 19
PACK 3: L = 10, 11, 12, 13, 14, 15, 16, 17, 18, 19
M = 10, 11, 12, 13, 14, 15, 16, 17, 18, 19
PACK 4: L = 10, 11, 12, 13, 14, 15, 16, 17, 18, 19
M = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

For each pack, we create train (80%) and test (20%) subsets. Then we will apply KNN classification using Kbest = 5 to each pack and assign the accuracy percentage as weighted values w1, w2, w3, and w4.

Accuracy of PACK 1
$W1 = testperf^{bestk} = 58\%$

1.7 - Finding the accuracy for each PACK using KNN algorithm, where K= 5.

Using the same script from 1.6, replicating for the following PACKS.

Accuracy of PACKS
$W2 = testperf^{bestk} = 75\%$
$W3 = testperf^{bestk} = 63\%$
$W4 = testperf^{bestk} = 66\%$

According to the KNN classification to each pack, we can see that pack 2 has the highest accuracy rate of 75%. On the other hand, pack 1 resulted to be the lowest accuracy rate of 58%. This is 5 and 8 percentual points behind packs 3 and 4 respectively.

1.8 - Finding the KNN with the weighted features on global performance and TESTSET

We use the accuracy rates we found in 1.7 to weight each pack before standardizing and combining them into one data set. Pack 2 will have more weight than the other packs because it has the highest accuracy rate. The non-weighted set assumes that all features are equally important when they are not. Hence, we weigh the features to account for the importance of different features.

Weighted Trainset Confusion Matrix

Weighted Trainset Confusion Matrix

Train

	PREDICTION		
Test K = 5	**CL1**	**CL2**	**CL3**

Test

		PREDICTION		
	Test K = 5	CL1	CL2	CL3
TRUE	CL1	73.89%	17.73%	8.37%
	CL2	17.05%	77.84%	5.11%
	CL3	5.70%	8.81%	85.49%

		PREDICTION		
	Test K = 5	CL1	CL2	CL3
TRUE	CL1	75.47%	15.09%	9.43%
	CL2	8.51%	78.72%	12.77%
	CL3	9.09%	6.82%	84.09%

The global train set accuracy = 79.07%
The global test set accuracy = 79.42%
As we see, the global test accuracy is higher than our trainset, which is a good sign that there is no overfitting. There is an increase in accuracy in CL1 and CL2 when we move from trainset to test set, while there is a slight loss of accuracy in CL3 when we move from train set to test set.

Ordinary Distance KNN Test Set

Ordinary

TEST

		PREDICTION		
	Test K = 5	CL1	CL2	CL3
TRUE	CL1	64.29%	28.57%	7.14%
	CL2	21.21%	69.70%	9.09%
	CL3	9.09%	16.36%	74.55%

Weighted Distance KNN Test Set

Weighted

TEST

		PREDICTION		
	Test K = 5	CL1	CL2	CL3
TRUE	CL1	75.47%	15.09%	9.43%
	CL2	8.51%	78.72%	12.77%
	CL3	9.09%	6.82%	84.09%

Ordinary KNN: Global test set accuracy = 69.51%
Weighted KNN: Global test set accuracy = 79.42%
The performance of the knn algorithm improved drastically after implementing the weighted distance knn. Our global performance increased by approximately 10%. Similarly, all three classes see an increase in accuracy by around 10%

Summary of Work done:
Script/work: 33% Anushka
33% Bam
33% Madeline
Interpretation and formatting: 33% Madeline
33% Bam
33%Anushka
All work done was dividedly evenly as we coded the script together over a team meeting and interpreted together.

Appendix:

Mean table:

m1	m2	m3	m4	m5	m6	m7	m8	m9	m10
26.7081	36.92179	46.43017	55.00419	60.21508	66.82123	71.97346	79.44693	81.97486	83.86034

m11	m12	m13	m14	m15	m16	m17	m18	m19	m20
88.7081	89.04749	91.94553	95.91341	95.49581	90.9162	83.73464	79.82961	82.23464	65.19972

m21	m22	m23	m24	m25	m26	m27	m28	m29	m30
28.48883	42.46788	57.28212	72.61592	83.20251	93.10056	96.30866	96.7081	97.27933	102.4232

m31	m32	m33	m34	m35	m36	m37	m38	m39	m40
106.8142	109.1313	112.1439	113.8841	111.9972	110.3506	108.7696	108.1578	100.5126	70.08939

m41	m42	m43	m44	m45	m46	m47	m48	m49	m50
25.73464	41.55307	61.39944	84.64665	96.72486	98.11732	93.86453	96.34777	98.97626	105.433

m51	m52	m53	m54	m55	m56	m57	m58	m59	m60
106.2682	100.3827	97.75279	93.25698	96.82542	105.4106	111.7877	106.1397	88.23045	57.96648

m61	m62	m63	m64	m65	m66	m67	m68	m69	m70
26.51117	44.41061	65.39804	84.6243	91.09916	91.68156	89.34218	85.93994	88.74162	94.17458

m71	m72	m73	m74	m75	m76	m77	m78	m79	m80
94.99302	84.22207	78.65503	80.50279	83.31564	91.29888	99.26257	95.79749	81.15503	54.63687

m81	m82	m83	m84	m85	m86	m87	m88	m89	m90
31.68994	51.38966	69.23184	82.30587	88.89665	87.60475	82.59358	83.44134	88.93296	90.5

m91	m92	m93	m94	m95	m96	m97	m98	m99	m100
86.35056	79.75	79.48184	83.94693	87.02654	90.92598	90.69972	90.39804	78.70531	52.76816

m101	m102	m103	m104	m105	m106	m107	m108	m109	m110
34.80587	56.18017	71.91201	87.16899	91.76816	91.64385	85.90084	88.4595	90.05866	88.43156

m111	m112	m113	m114	m115	m116	m117	m118	m119	m120
87.06285	83.18715	84.82682	87.87151	90.98883	89.27235	90.74302	83.57542	71.46089	46.07821

m121	m122	m123	m124	m125	m126	m127	m128	m129	m130
38.17458	58.43017	72.93436	91.46229	94.81425	92.9162	92.61034	88.0852	85.48464	88.35754

m131	m132	m133	m134	m135	m136	m137	m138	m139	m140
89.8324	88.07821	87.51397	88.1257	85.42598	84.42737	88.91061	81.97067	64.95251	43.82682

m141	m142	m143	m144	m145	m146	m147	m148	m149	m150
42.85056	58.81704	72.43017	90.09637	95.11732	94.88687	95.42877	95.30726	91.45112	95.56145

m151	m152	m153	m154	m155	m156	m157	m158	m159	m160
100.2109	100.6313	99.70251	88.32961	83.58939	85.28073	86.26536	78.32402	59.49581	43.56006

m161	m162	m163	m164	m165	m166	m167	m168	m169	m170
46.69693	63.60335	76.88547	94.75559	99.97905	101.9316	104.7612	105.8464	100.6034	104.6397

m171	m172	m173	m174	m175	m176	m177	m178	m179	m180
113.014	110.4441	100.1271	91.85196	90.48883	88.23324	86.32542	73.97486	58.37709	41.77654

m181	m182	m183	m184	m185	m186	m187	m188	m189	m190
47.51676	69.37291	85.98184	104.2095	106.2709	108.5461	108.0028	105.3478	104.3939	110.0545

m191	m192	m193	m194	m195	m196	m197	m198	m199	m200
107.3757	96.6662	93.67318	94.45531	92.03073	88.23045	87.01257	73.92039	57.44274	43.86453

m201	m202	m203	m204	m205	m206	m207	m208	m209	m210
47.67318	70.52374	89.88128	105.8911	109.5894	106.8799	102.7793	101.1047	103.5782	103.0391
m211	**m212**	**m213**	**m214**	**m215**	**m216**	**m217**	**m218**	**m219**	**m220**
92.43436	89.40642	91.94274	94.38687	89.35335	88.04469	85.08799	72.8324	60.12849	42.51816
m221	**m222**	**m223**	**m224**	**m225**	**m226**	**m227**	**m228**	**m229**	**m230**
46.96369	74.15782	91.32402	108.0209	111.7793	104.3911	96.69274	97.54609	99.99022	96.56425
m231	**m232**	**m233**	**m234**	**m235**	**m236**	**m237**	**m238**	**m239**	**m240**
89.95531	88.1676	90.79469	90.72905	86.94274	82.74581	81.83659	72.56006	59.98603	42.47765
m241	**m242**	**m243**	**m244**	**m245**	**m246**	**m247**	**m248**	**m249**	**m250**
51.99302	76.88128	97.09777	114.1201	112.9302	104.5601	94.80307	94.55447	94.3757	97.4148
m251	**m252**	**m253**	**m254**	**m255**	**m256**	**m257**	**m258**	**m259**	**m260**
95.10894	89.85475	90.93017	88.59218	85.42318	82.03771	82.26117	71.89246	58.2067	43.5433
m261	**m262**	**m263**	**m264**	**m265**	**m266**	**m267**	**m268**	**m269**	**m270**
53.55307	82.01955	102.8073	116.8589	110.7332	102.1215	91.22765	86.50978	85.61313	89.84078
m271	**m272**	**m273**	**m274**	**m275**	**m276**	**m277**	**m278**	**m279**	**m280**
88.0824	84.97486	86.62291	81.9162	79.17318	75.89246	77.90363	66.14385	53.25838	38.96229
m281	**m282**	**m283**	**m284**	**m285**	**m286**	**m287**	**m288**	**m289**	**m290**
50.73743	88.76257	107.1034	118.8073	109.4721	93.23464	77.59218	75.58101	77.34777	82.05307
m291	**m292**	**m293**	**m294**	**m295**	**m296**	**m297**	**m298**	**m299**	**m300**
85.16201	82.36313	78.02793	76.03212	74.44274	75.5852	77.13547	66.03771	52.36872	36.41341
m301	**m302**	**m303**	**m304**	**m305**	**m306**	**m307**	**m308**	**m309**	**m310**
57.83799	93.67598	117.5154	121.2807	105.2472	85.12849	70.23743	71.2067	72.85615	76.26397
m311	**m312**	**m313**	**m314**	**m315**	**m316**	**m317**	**m318**	**m319**	**m320**
80.76536	76.34497	72.51676	71.56285	74.19274	74.78911	76.81425	64.84078	48.77654	32.9567
m321	**m322**	**m323**	**m324**	**m325**	**m326**	**m327**	**m328**	**m329**	**m330**
62.48883	97.34218	119.6425	121.7123	104.5042	87.53352	77.50838	70.58939	71.02374	75.5014
m331	**m332**	**m333**	**m334**	**m335**	**m336**	**m337**	**m338**	**m339**	**m340**
79.65363	75.66201	70.96369	71.43855	75.08939	76.88268	75.81564	58.55168	42.13268	24.60056
m341	**m342**	**m343**	**m344**	**m345**	**m346**	**m347**	**m348**	**m349**	**m350**
66.76397	101.324	113.1592	123.8869	114.6969	99.82821	86.76117	83.02095	80.04609	83.0852
m351	**m352**	**m353**	**m354**	**m355**	**m356**	**m357**	**m358**	**m359**	**m360**
86.09358	79.51816	77.0433	74.55307	76.02654	77.21229	74.80587	52.57821	37.72067	22.63408
m361	**m362**	**m363**	**m364**	**m365**	**m366**	**m367**	**m368**	**m369**	**m370**
74.89804	103.352	109.3031	115.7514	119.1327	116.324	103.7346	89.72486	86.31983	90.04469
m371	**m372**	**m373**	**m374**	**m375**	**m376**	**m377**	**m378**	**m379**	**m380**
90.88547	84.53492	78.83939	79.26955	79.95531	73.06983	63.76117	47.6662	35.25	26.96927
m381	**m382**	**m383**	**m384**	**m385**	**m386**	**m387**	**m388**	**m389**	**m390**
71.08101	90.88128	92.72626	94.53911	93.59497	93.39106	93.85056	93.97626	91.61313	94.40503
m391	**m392**	**m393**	**m394**	**m395**	**m396**	**m397**	**m398**	**m399**	**m400**
93.66061	84.03352	77.71648	72.85056	66.63827	61.72346	52.72067	43.76536	41.03212	33.62151

Standard Deviation table:

std1	std2	std3	std4	std5	std6	std7	std8	std9	std10
65.01184	77.77399	89.92133	96.12808	99.45747	99.72622	98.61528	102.475	103.1712	105.9436
std11	**std12**	**std13**	**std14**	**std15**	**std16**	**std17**	**std18**	**std19**	**std20**
107.6655	104.0043	105.9508	107.8664	109.742	109.6022	105.7334	103.4696	104.9	98.1944
std21	**std22**	**std23**	**std24**	**std25**	**std26**	**std27**	**std28**	**std29**	**std30**
73.17727	86.51969	96.27368	106.1496	109.0525	110.0521	110.8936	110.9029	111.1358	114.3951
std31	**std32**	**std33**	**std34**	**std35**	**std36**	**std37**	**std38**	**std39**	**std40**
114.3278	113.6467	114.3165	112.2357	111.1863	110.0747	110.2678	108.8677	111.6475	99.15723
std41	**std42**	**std43**	**std44**	**std45**	**std46**	**std47**	**std48**	**std49**	**std50**
66.57551	84.47719	97.02959	107.7706	113.2899	111.2893	109.3168	109.8132	111.3044	112.7589
std51	**std52**	**std53**	**std54**	**std55**	**std56**	**std57**	**std58**	**std59**	**std60**
112.8439	110.385	108.4825	106.3678	109.7662	110.2009	112.0855	112.2007	107.0797	95.61727
std61	**std62**	**std63**	**std64**	**std65**	**std66**	**std67**	**std68**	**std69**	**std70**
69.36742	85.77395	98.22711	108.7697	110.647	110.9331	109.0668	107.3659	107.3419	110.3963
std71	**std72**	**std73**	**std74**	**std75**	**std76**	**std77**	**std78**	**std79**	**std80**
110.3127	104.9195	103.6291	104.8655	105.7869	107.8932	111.6919	108.3138	106.6593	94.56195
std81	**std82**	**std83**	**std84**	**std85**	**std86**	**std87**	**std88**	**std89**	**std90**
75.43729	92.55121	102.674	109.417	110.6596	110.3763	107.0574	108.2463	109.1284	108.89
std91	**std92**	**std93**	**std94**	**std95**	**std96**	**std97**	**std98**	**std99**	**std100**
108.0318	104.181	104.6612	106.5682	107.9011	108.3206	109.3257	109.7276	104.9904	93.3721
std101	**std102**	**std103**	**std104**	**std105**	**std106**	**std107**	**std108**	**std109**	**std110**
77.89744	94.80273	104.4505	112.626	113.1938	111.1614	108.8512	110.2526	110.0633	110.4286
std111	**std112**	**std113**	**std114**	**std115**	**std116**	**std117**	**std118**	**std119**	**std120**
109.6303	105.2838	106.7398	107.8662	109.614	107.5424	108.1313	108.2292	101.9768	89.27608
std121	**std122**	**std123**	**std124**	**std125**	**std126**	**std127**	**std128**	**std129**	**std130**
80.53627	99.00712	104.8335	111.2779	112.9202	109.4507	109.9541	109.7387	106.6592	111.197
std131	**std132**	**std133**	**std134**	**std135**	**std136**	**std137**	**std138**	**std139**	**std140**
109.768	109.3948	108.6424	108.226	106.7134	106.2038	108.989	108.5067	100.8332	87.79576
std141	**std142**	**std143**	**std144**	**std145**	**std146**	**std147**	**std148**	**std149**	**std150**
86.72678	98.77327	104.2127	111.5359	112.1147	112.5665	109.2385	107.8951	107.8542	108.746
std151	**std152**	**std153**	**std154**	**std155**	**std156**	**std157**	**std158**	**std159**	**std160**
111.6187	110.6202	110.2695	106.4121	105.9861	108.3065	109.6404	106.6664	98.57256	90.1915
std161	**std162**	**std163**	**std164**	**std165**	**std166**	**std167**	**std168**	**std169**	**std170**
91.06014	101.5346	105.3303	110.7277	112.8108	109.7708	111.0405	111.2728	109.0092	111.9694
std171	**std172**	**std173**	**std174**	**std175**	**std176**	**std177**	**std178**	**std179**	**std180**
114.6164	110.6253	107.4026	106.6358	109.0423	109.2527	107.7716	103.7001	96.97228	87.56876
std181	**std182**	**std183**	**std184**	**std185**	**std186**	**std187**	**std188**	**std189**	**std190**
90.50053	104.3334	108.927	114.7246	114.2001	111.7583	112.395	110.4983	110.1754	114.7988
std191	**std192**	**std193**	**std194**	**std195**	**std196**	**std197**	**std198**	**std199**	**std200**
113.1017	108.6419	109.2085	108.7343	109.1884	109.3082	108.5542	103.8954	98.97647	87.74166

std201	**std202**	**std203**	**std204**	**std205**	**std206**	**std207**	**std208**	**std209**	**std210**
92.01754	104.2321	109.8432	114.6027	113.2071	113.8214	109.4575	110.4317	111.8202	112.487
std211	**std212**	**std213**	**std214**	**std215**	**std216**	**std217**	**std218**	**std219**	**std220**
109.4322	108.0547	110.4015	109.7561	110.5369	109.9244	107.6943	103.3079	100.1288	88.22832
std221	**std222**	**std223**	**std224**	**std225**	**std226**	**std227**	**std228**	**std229**	**std230**
91.25816	105.2779	109.9194	112.8844	114.9206	111.5774	107.163	107.874	107.9325	109.2925
std231	**std232**	**std233**	**std234**	**std235**	**std236**	**std237**	**std238**	**std239**	**std240**
109.4669	105.0402	107.7556	108.6229	109.1807	109.9411	106.8203	102.5766	100.6049	87.09678
std241	**std242**	**std243**	**std244**	**std245**	**std246**	**std247**	**std248**	**std249**	**std250**
93.7575	108.449	109.9511	115.9125	114.2496	111.2856	109.6687	110.8891	109.2339	111.9685
std251	**std252**	**std253**	**std254**	**std255**	**std256**	**std257**	**std258**	**std259**	**std260**
111.8796	106.9934	109.0582	108.1925	108.5237	107.9135	107.5531	102.8561	98.68922	87.52956
std261	**std262**	**std263**	**std264**	**std265**	**std266**	**std267**	**std268**	**std269**	**std270**
93.36729	109.8922	111.4953	112.6009	113.0407	112.3428	106.9395	107.1242	107.0708	110.5673
std271	**std272**	**std273**	**std274**	**std275**	**std276**	**std277**	**std278**	**std279**	**std280**
108.9861	107.1868	108.3472	106.5621	106.8883	106.1196	105.1396	100.1424	96.23932	83.93577
std281	**std282**	**std283**	**std284**	**std285**	**std286**	**std287**	**std288**	**std289**	**std290**
91.57904	110.9244	110.3894	111.9533	115.3169	109.2023	101.6306	102.8668	104.9358	109.178
std291	**std292**	**std293**	**std294**	**std295**	**std296**	**std297**	**std298**	**std299**	**std300**
110.7465	107.8505	105.1314	105.8037	106.1284	105.7429	106.1472	102.521	94.94	80.98636
std301	**std302**	**std303**	**std304**	**std305**	**std306**	**std307**	**std308**	**std309**	**std310**
95.86064	112.5366	111.2259	114.6515	113.2413	106.5227	100.4918	104.3293	104.7076	106.6137
std311	**std312**	**std313**	**std314**	**std315**	**std316**	**std317**	**std318**	**std319**	**std320**
108.2493	105.1838	103.7918	104.2295	105.9652	104.5951	107.7814	100.857	90.04037	74.62591
std321	**std322**	**std323**	**std324**	**std325**	**std326**	**std327**	**std328**	**std329**	**std330**
99.64644	111.1248	109.7204	114.3687	112.5473	108.0533	103.521	103.6508	102.5066	107.8383
std331	**std332**	**std333**	**std334**	**std335**	**std336**	**std337**	**std338**	**std339**	**std340**
109.4559	105.4097	103.7165	102.8979	104.2572	107.4286	105.4292	93.91935	82.48611	64.88382
std341	**std342**	**std343**	**std344**	**std345**	**std346**	**std347**	**std348**	**std349**	**std350**
99.94294	114.1181	110.0377	109.68	112.3892	110.8319	105.0558	107.7337	105.5764	109.6136
std351	**std352**	**std353**	**std354**	**std355**	**std356**	**std357**	**std358**	**std359**	**std360**
109.7017	106.2313	104.8682	104.0605	103.1199	104.9302	102.6363	88.94438	77.34717	60.94595
std361	**std362**	**std363**	**std364**	**std365**	**std366**	**std367**	**std368**	**std369**	**std370**
100.9806	115.4522	107.9828	108.6567	109.7589	110.6269	107.9677	105.5335	106.5381	109.3004
std371	**std372**	**std373**	**std374**	**std375**	**std376**	**std377**	**std378**	**std379**	**std380**
109.5063	105.0214	103.2706	103.7606	101.7176	101.7131	97.80341	83.94673	78.43226	69.63844
std381	**std382**	**std383**	**std384**	**std385**	**std386**	**std387**	**std388**	**std389**	**std390**
96.92621	104.1647	106.2246	108.6647	108.2422	105.9109	104.6085	106.9265	105.6322	108.3754
std391	**std392**	**std393**	**std394**	**std395**	**std396**	**std397**	**std398**	**std399**	**std400**
107.99	104.29	101.3722	98.43293	99.17809	99.26074	93.79278	87.30378	82.70183	72.51219

Full script:

```
attach(BELL)
library(dplyr) #Package for subseting data
bell_clean<- select(BELL,-c(2,3,6,7,8,9,10,11,12))#discard the 9 columns
View(bell_clean)
#956x403

attach(CHILLER)
chiller_clean<- select(CHILLER,-c(2,3,6,7,8,9,10,11,12))
View(chiller_clean)
#952x403

attach(FREESTYLE)
freestyle_clean<- select(FREESTYLE,-c(2,3,6,7,8,9,10,11,12))
View(freestyle_clean)
#956x403

#disccarding row containing missing numerical data
BELL_clean<- na.omit(bell_clean)
CHILLER_clean<- na.omit(chiller_clean)
FREESTYLE_clean<- na.omit(freestyle_clean)

#Defining three classes of images of normal characters
#cl1 = all rows of BELL_clean.csv file for which (strength = 0.4 and italic =0)
#cl2 = all rows of CHILLER_clean.csv file for which (strength = 0.4 and italic =0)
#cl3 = all rows of FREESTYLE_clean.csv file for which (strength = 0.4 and italic =0)

BELL_clean<-data.frame(BELL_clean)#creating a data frame to add conditional statements to
filter out non needed i features
BELL_clean$CL = ifelse((BELL_clean$strength == 0.4 & BELL_clean$italic == 0),"CL1","NA")
BELL_CLEAN = BELL_clean[which(BELL_clean$CL =="CL1"),] #labeling the new filter data as
cl1
View(BELL_CLEAN)
#404 columns
#239 rows

CHILLER_clean<-data.frame(CHILLER_clean)
CHILLER_clean$CL = ifelse((CHILLER_clean$strength == 0.4 & CHILLER_clean$italic ==
0),"CL2","NA")
CHILLER_CLEAN = CHILLER_clean[which(CHILLER_clean$CL =="CL2"),]
View(CHILLER_CLEAN)
#404 columns
#238 rows

FREESTYLE_clean<-data.frame(FREESTYLE_clean)
FREESTYLE_clean$CL = ifelse((FREESTYLE_clean$strength == 0.4 & FREESTYLE_clean$italic ==
0),"CL3","NA")
FREESTYLE_CLEAN = FREESTYLE_clean[which(FREESTYLE_clean$CL =="CL3"),]
View(FREESTYLE_CLEAN)
#404 columns
#239 rows

# Combine CL1, CL2, CL3 into DATA
DATA<-rbind(BELL_CLEAN,CHILLER_CLEAN,FREESTYLE_CLEAN)
View(DATA)
# Binded all 3 data sets to a full data set (DATA) which is the union of 3 classes (CL1,
CL2, CL3)
# where N = 716

# Part 0
# Compute mean
DATAMEAN<-DATA %>% summarize_if(is.numeric,mean)
mean(DATA[,3]) #mean of this column is 0 which is ok
```

```r
# Compute standard deviation
DATASD<-DATA %>% summarize_if(is.numeric, sd)
var(DATA[,3]) #sd is 0 which is not okay, which we need to standardize to have a
comparable scale
###standardizing to make a comparable scale
library(standardize)
SDATA<- DATA %>% mutate_if(is.numeric, function (x) as.vector(scale(x))) # scaling by
(xj-mj)/sd
SDATA = SDATA[,-c(2,3)] #taking out numerical functions of strength and italics
sDATA<- data.matrix(SDATA) #creating it into a data matrix for correlation matrix
beforehand
View(SDATA)
### SDATA contains CL classes and font name, but not strength and italics.
### confirming the standardization has properly worked by looking at mean and sd again of
the SDATA
var(SDATA[,3])#sd =1 which is good



# Scale the data again for the
sDATA1<-scale(sDATA[,-c(1,2,3,404)])# scaling and removing non-numerical values
sDATA1
#### sDATA1 is data set containing standardized features, but without non-numerical
values.
# correlation matrix
cor(sDATA1)
cor.df = data.frame(cor(DATA1)) #renaming to view actual full matrix
View(cor.df)



#packages needed to find the top 10 values
library(dplyr)
library(tidyr)
#finding the top 10 highest values
topvalues_sdata<-cor(sDATA1) %>%
  as.data.frame() %>%
  mutate(var1 = rownames(.)) %>%
  gather(var2, value, -var1) %>%
  arrange(desc(value)) %>%
  group_by(value) %>%
  filter(row_number()==1)
View(topvalues_sdata)



## loop to classify cl to SETROW columns into data set
SDATA$SETROW<-NA

for (i in 1:716){
  if(SDATA$CL[i]=="CL1"){
    SDATA$SETROW[i] = "SETROW1"
}else if(SDATA$CL[i]=="CL2"){
  SDATA$SETROW[i] = "SETROW2"
}else{
  SDATA$SETROW[i] = "SETROW3"
}
}
#creating the 80% random train set interval by taking ONLY using setrow 1, we replicate
this for the other setrow functions
SETROW1 = SDATA[which(SDATA$SETROW =="SETROW1"),]
n<-nrow(SETROW1[which(SETROW1$SETROW=="SETROW1"),])
trainset<-sample(1:n, 0.8*n)
```

```r
#
trainsetcl1 <- SETROW1[trainset,]
testsetcl1 <- SETROW1[-trainset,]

SETROW2 = SDATA[which(SDATA$SETROW =="SETROW2"),]
n<-nrow(SETROW2[which(SETROW2$SETROW=="SETROW2"),])
trainset<-sample(1:n, 0.8*n)
trainsetcl2 <- SETROW2[trainset,]
testsetcl2 <- SETROW2[-trainset,]

SETROW3 = SDATA[which(SDATA$SETROW =="SETROW3"),]
n<-nrow(SETROW3[which(SETROW3$SETROW=="SETROW3"),])
trainset<-sample(1:n, 0.8*n)
trainsetcl3 <- SETROW3[trainset,]
testsetcl3 <- SETROW3[-trainset,]

#combining the sets to full trainset and testset

TRAIN_SET<-rbind(trainsetcl1,trainsetcl2,trainsetcl3)
TEST_SET<- rbind(testsetcl1,testsetcl2,testsetcl3)

#train and test labels
library(class)
SDATA_no <- SDATA[,-c(1,402,403)]
SDATA_label <- SDATA[,"CL"]
TRAIN_no <- TRAIN_SET[,-c(1,402,403)]
TRAIN_label <- TRAIN_SET[, "CL"]
TEST_no <- TEST_SET[,-c(1,402,403)]
TEST_label <- TEST_SET[,"CL"]

# Compute the percentage of correct classification
RNGkind(sample.kind = "Rounding")
set.seed(1)
knn.predtrain12 <- knn(train=TRAIN_no,
                test=TRAIN_no,
                cl = TRAIN_label,
                k=12)

RNGkind(sample.kind = "Rounding")
set.seed(1)
knn.predtest12 <- knn(train=TRAIN_no,
                test=TEST_no,
                cl = TRAIN_label,
                k=12)

mean(knn.predtrain12!= TRAIN_label) #[1] 0.2814685 pretty bad
mean(knn.predtest12 != TEST_label) #[1] 0.3194444 very high which makes sense because its
the test set

#confusion matrix
table(data.frame(knn.predtrain,TRAIN_label))
table(data.frame(knn.predtest, TEST_label))

#1.2
#fit the model on the training set finding the optimized value of k for test set
#running a loop
set.seed(1)
K.set = c(5,10,15,20,25,30,40,50,100)
knn.test.accuracy <- numeric(length(K.set))

for (j in 1:length(K.set)){
  knn.pred <- knn(train=TRAIN_no,
                  test=TEST_no,
```

```
                  cl=TRAIN_label,
                  k=K.set[j])
  knn.test.accuracy[j] <- mean(knn.pred == TEST_label)
}

####finding percent accuracy for each value of 5,10...100
set.seed(1)
i=1
k.optm=1
for (i in seq(5, 100, by = 5)){
  knn.mod<- knn(train = TRAIN_no, test = TEST_no, cl = TRAIN_label, k= i)
  k.optm[i]<- 100 * sum(TEST_label == knn.mod)/ NROW(TEST_label)
  k=i
  cat(k,"=", k.optm[i],'\n')
}

### finding accuracy for train which will be higher
set.seed(1)
K.set = c(5,10,15,20,25,30,40,50,100)
knn.train.accuracy <- numeric(length(K.set))

for (j in 1:length(K.set)){
  knn.pred <- knn(train=TRAIN_no,
                  test=TRAIN_no,
                  cl=TRAIN_label,
                  k=K.set[j])
  knn.train.accuracy[j] <- mean(knn.pred == TRAIN_label)
}
###plotting the figure with each other.
##red = test set
## blue = trainset

plot(K.set, knn.train.accuracy, type="o", col="blue", pch="o", lty=1 )
points(K.set, knn.test.accuracy, col="red", pch="*")
lines(K.set, knn.test.accuracy, col="red",lty=2)

#### based on the figure we can do 5:20 range to explore more k values
K.set = c(5:20)
knn.test.accuracy <- numeric(length(K.set))
set.seed(1)
for (j in 1:length(K.set)){
  knn.pred <- knn(train=TRAIN_no,
                  test=TEST_no,
                  cl=TRAIN_label,
                  k=K.set[j])
  knn.test.accuracy[j] <- mean(knn.pred == TEST_label)
}
# Find best k
max(knn.test.accuracy)
K.set[which.max(knn.test.accuracy)]
##based on the information provided, we use K=5 as k best

##Applying the "best" k value to the both train and test set.
set.seed(1)
knn.predtrainbest<- knn(train=TRAIN_no,
                test=TRAIN_no,
                cl = TRAIN_label,
                k=5)
set.seed(1)
knn.predtestbest <- knn(train=TRAIN_no,
                test=TEST_no,
                cl = TRAIN_label,
                k=5)
```

```
##displaying the percent error values
mean(knn.predtrainbest == TRAIN_label)
mean(knn.predtestbest == TEST_label)

##displaying confusion matrix of cl
trainmt<-table(data.frame(knn.predtrainbest,TRAIN_label))
testtt<-table(data.frame(knn.predtestbest, TEST_label))

#####finding confidence intervals of confusion matrix
library(DescTools)
Conf(trainmt)
#yielding 95% CI : (0.7975, 0.8589)
Conf(testtt)
#yielding 95% CI : (0.6734, 0.8136)

###1.6
###PACK 1 L: 0-9 and M: 0-9 making a 100 attributes
PACK1<-
SDATA[,c(2:11,22:31,42:51,62:71,82:91,102:111,122:131,142:151,162:171,182:191,402,403)]
PACK2<-
SDATA[,c(12:21,32:41,52:61,72:81,92:101,112:121,132:141,152:161,172:181,192:201,402,403)]
PACK3<-
SDATA[,c(212:221,232:241,252:261,272:281,292:301,312:321,332:341,352:361,372:381,392:401,
402,403)]
PACK4<-
SDATA[,c(202:211,222:231,242:251,262:271,282:291,302:311,322:331,342:351,362:371,382:391,
402,403)]

##dividing set of pack 1 of .8 of train .2 test
packcl1 = PACK1[which(PACK1$SETROW=="SETROW1"),]
n<-nrow(packcl1)
PACKCL11<-sample(1:n, 0.8*n)

PACKCL1_train1 <- packcl1[PACKCL11,]
PACKCL1_test1 <- packcl1[-PACKCL11,]

#replicating for cl2,
packcl2 = PACK1[which(PACK1$SETROW=="SETROW2"),]
n<-nrow(packcl2)
PACKCL21<-sample(1:n, 0.8*n)

PACKCL2_train1 <- packcl2[PACKCL21,]
PACKCL2_test1 <- packcl2[-PACKCL21,]

### replicating for cl3
packcl3 = PACK1[which(PACK1$SETROW=="SETROW3"),]
n<-nrow(packcl3)
PACKCL31<-sample(1:n, 0.8*n)

PACKCL3_train1 <- packcl3[PACKCL31,]
PACKCL3_test1 <- packcl3[-PACKCL31,]
####UNIONIZING PACK1 CLs
PACK1_TRAINALL<- rbind(PACKCL1_train1,PACKCL2_train1,PACKCL3_train1)
PACK1_TESTALL<- rbind(PACKCL1_test1,PACKCL2_test1,PACKCL3_test1)

######PACK 2 #######
##dividing set of pack 1 of .8 of train .2 test

packcl1p2 = PACK2[which(PACK2$SETROW=="SETROW1"),]
n<-nrow(packcl1p2)
PACKCL1p2<-sample(1:n, 0.8*n)

PACKCL1_train2 <- packcl1p2[PACKCL1p2,]
```

```r
PACKCL1_test2 <- packcl1p2[-PACKCL1p2,]

#pack2 cl2
packcl2p2 = PACK2[which(PACK2$SETROW=="SETROW2"),]
n<-nrow(packcl2p2)
PACKCL2p2<-sample(1:n, 0.8*n)

PACKCL2_train2 <- packcl2p2[PACKCL2p2,]
PACKCL2_test2 <- packcl2p2[-PACKCL2p2,]

#pack 2 cl3
packcl3p2 = PACK2[which(PACK2$SETROW=="SETROW1"),]
n<-nrow(packcl3p2)
PACKCL3p2<-sample(1:n, 0.8*n)

PACKCL3_train2 <- packcl3p2[PACKCL3p2,]
PACKCL3_test2 <- packcl3p2[-PACKCL3p2,]

PACK2_TRAINALL<- rbind(PACKCL1_train2,PACKCL2_train2,PACKCL3_train2)
PACK2_TESTALL<- rbind(PACKCL1_test2,PACKCL2_test2,PACKCL3_test2)

######PACK 3
##CL1
packcl1p3 = PACK3[which(PACK3$SETROW=="SETROW1"),]
n<-nrow(packcl1p3)
PACKCL1p3<-sample(1:n, 0.8*n)

PACKCL1_train3 <- packcl1p3[PACKCL1p3,]
PACKCL1_test3 <- packcl1p3[-PACKCL1p3,]
#####CL2

packcl2p3 = PACK3[which(PACK3$SETROW=="SETROW2"),]
n<-nrow(packcl2p3)
PACKCL2p3<-sample(1:n, 0.8*n)

PACKCL2_train3 <- packcl2p3[PACKCL2p3,]
PACKCL2_test3 <- packcl2p3[-PACKCL2p3,]

###CL3
packcl3p3 = PACK3[which(PACK3$SETROW=="SETROW3"),]
n<-nrow(packcl3p3)
PACKCL3p3<-sample(1:n, 0.8*n)

PACKCL3_train3 <- packcl3p3[PACKCL3p3,]
PACKCL3_test3 <- packcl3p3[-PACKCL3p3,]

PACK3_TRAINALL<- rbind(PACKCL1_train3,PACKCL2_train3,PACKCL3_train3)
PACK3_TESTALL<- rbind(PACKCL1_test3,PACKCL2_test3,PACKCL3_test3)

###PACK 4
#CL1
packcl1p4 = PACK4[which(PACK4$SETROW=="SETROW1"),]
n<-nrow(packcl1p4)
PACKCL1p4<-sample(1:n, 0.8*n)

PACKCL1_train4 <- packcl1p4[PACKCL1p4,]
PACKCL1_test4 <- packcl1p4[-PACKCL1p4,]

#CL2
packcl2p4 = PACK4[which(PACK4$SETROW=="SETROW2"),]
n<-nrow(packcl2p4)
PACKCL2p4<-sample(1:n, 0.8*n)
```

```r
PACKCL2_train4 <- packcl2p4[PACKCL2p4,]
PACKCL2_test4 <- packcl2p4[-PACKCL2p4,]

#CL3
packcl3p4 = PACK4[which(PACK4$SETROW=="SETROW3"),]
n<-nrow(packcl3p4)
PACKCL3p4<-sample(1:n, 0.8*n)

PACKCL3_train4 <- packcl3p4[PACKCL3p4,]
PACKCL3_test4 <- packcl3p4[-PACKCL3p4,]

PACK4_TRAINALL<- rbind(PACKCL1_train4,PACKCL2_train4,PACKCL3_train4)
PACK4_TESTALL<- rbind(PACKCL1_test4,PACKCL2_test4,PACKCL3_test4)

################labeling test entry for knn #################

PACK1_TRAINALL_no<- PACK1_TRAINALL[,-c(101,102)]
PACK1_TRAINALL_LABEL <- PACK1_TRAINALL[,"CL"]
PACK2_TRAINALL_no<- PACK2_TRAINALL[,-c(101,102)]
PACK2_TRAINALL_LABEL <- PACK2_TRAINALL[,"CL"]
PACK3_TRAINALL_no<- PACK3_TRAINALL[,-c(101,102)]
PACK3_TRAINALL_LABEL <- PACK3_TRAINALL[,"CL"]
PACK4_TRAINALL_no<- PACK4_TRAINALL[,-c(101,102)]
PACK4_TRAINALL_LABEL <- PACK4_TRAINALL[,"CL"]

PACK1_TESTALL_no<- PACK1_TESTALL[,-c(101,102)]
PACK1_TESTALL_LABEL <- PACK1_TESTALL[,"CL"]
PACK2_TESTALL_no<- PACK2_TESTALL[,-c(101,102)]
PACK2_TESTALL_LABEL <- PACK2_TESTALL[,"CL"]
PACK3_TESTALL_no<- PACK3_TESTALL[,-c(101,102)]
PACK3_TESTALL_LABEL <- PACK3_TESTALL[,"CL"]
PACK4_TESTALL_no<- PACK4_TESTALL[,-c(101,102)]
PACK4_TESTALL_LABEL <- PACK4_TESTALL[,"CL"]

# Apply KNN using K = 5 to all four pack testssets
set.seed(1)
knn.predPACK1 <- knn(train=PACK1_TRAINALL_no,
                     test=PACK1_TESTALL_no,
                     cl = PACK1_TRAINALL_LABEL,
                     k=5)


set.seed(1)
knn.predPACK2 <- knn(train=PACK2_TRAINALL_no,
                    test=PACK2_TESTALL_no,
                    cl = PACK2_TRAINALL_LABEL,
                    k=5)



set.seed(1)
knn.predPACK3 <- knn(train=PACK3_TRAINALL_no,
                    test=PACK3_TESTALL_no,
                    cl = PACK3_TRAINALL_LABEL,
                    k=5)



set.seed(1)
knn.predPACK4 <- knn(train=PACK4_TRAINALL_no,
                    test=PACK4_TESTALL_no,
                    cl = PACK4_TRAINALL_LABEL,
                    k=5)
```

```r
# Find accuracy and set it to weight
w1 <- mean(knn.predPACK1 == PACK1_TESTALL_LABEL)
w2 <- mean(knn.predPACK2 == PACK2_TESTALL_LABEL)
w3 <- mean(knn.predPACK3 == PACK3_TESTALL_LABEL)
w4 <- mean(knn.predPACK4 == PACK4_TESTALL_LABEL)
#displaying values of accuracy

w1 #0.5486111
w2 #0.7569444
w3 #0.6319444
w4 #0.6666667
### we can see pack2 had the highest accuracy here

# Multiply weights to each pack
wpack1<-PACK1_no*w1
wpack2<-PACK2_no*w2
wpack3<-PACK3_no*w3
wpack4<-PACK4_no*w4

#binding the weighted packs
wpackfull<- cbind(wpack1,wpack2,wpack3,wpack4,PACK1[,101])
View(wpackfull)
#### normalizing the full weight packs
Swpackfull<- wpackfull %>% mutate_if(is.numeric, function (x) as.vector(scale(x))

#LABELS
waptrainset_no <- waptrainset[,-401]
waptrainset_label <- waptrainset[,401]

waptestset_no <- waptestset[,-401]
waptestset_label <- waptestset[,401]

# Global Performance for both train and test set with weighted values where knn= 5
set.seed(1)

knn.predwtrain <- knn(train=waptrainset_no,
                   test=waptrainset_no,
                   cl = waptrainset_label,
                   k=5)

full.predwtrain<- mean(knn.predwtrain == waptrainset_label)
full.predwtrain #0.78 accuracy yay

##testset
knn.predwtest <- knn(train=waptrainset_no,
                   test=waptestset_no,
                   cl = waptrainset_label,
                   k=5)
full.predwtest<- mean(knn.predwtest == waptestset_label)
full.predwtest

#confusion matrix
table(data.frame(knn.predwtrain,waptrainset_label))
table(data.frame(knn.predwtest, waptestset_label))
```