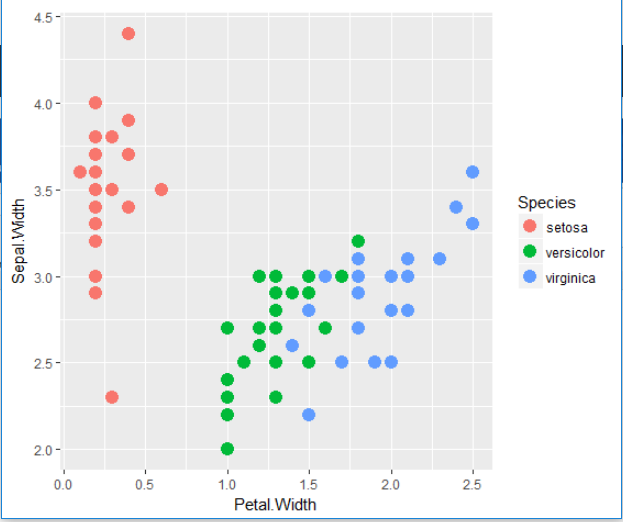
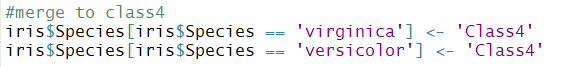
**P1.**

1. **Versicolor and virginica seems similar to each other**

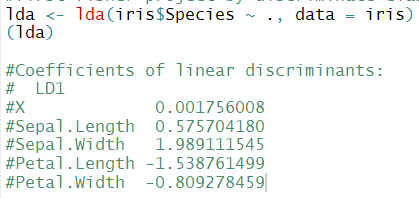




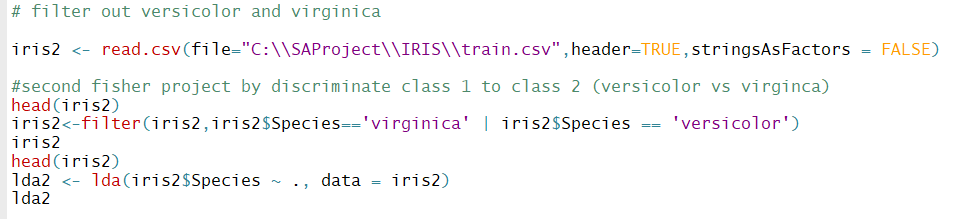
1. **Creating metaclass**



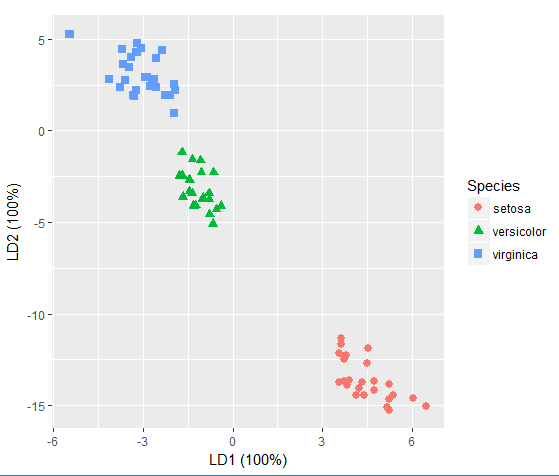
1. **First fisher project to discriminate class4 to class 3**



1. **Second fisher project by discriminate class 1 to class 2**



1. **Project the entire data in two projections**



**-------------------------- -------- end of P1-----------------------------------------------------------------------------**

**P2.**

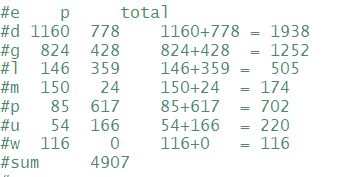
**Following is matrix for each feature in data.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Feature Name** | **Feature Accuracy** | **Gini Index** | **1- Entropy** | **Importance** |
| 1 | V17 | 0.516608926 | 0.500551713 | 0.000795451 | 4.00E-04 |
| 2 | V6 | 0.966014898 | 0.93433977 | 0.997977485 | 0.9641 |
| 3 | V21 | 0.576153041 | 0.379837079 | 0.998674123 | 0.5754 |
| 4 | V9 | 0.62213964 | 0.467649493 | 0.851924096 | 0.53 |
| 5 | V18 | 0.50491529 | 0.50004832 | 0.999156303 | 0.5045 |
| 6 | V8 | 0.51000119 | 0.42742988 | 0.933007242 | 0.4758 |
| 7 | V19 | 0.449732508 | 0.394829325 | 0.958865208 | 0.4312 |
| 8 | V15 | 0.407431154 | 0.279766882 | 0.999156303 | 0.4071 |
| 9 | V5 | 0.569812004 | 0.40504612 | 0.713082623 | 0.4063 |
| 10 | V12 | 0.397268816 | 0.271610887 | 0.998307827 | 0.3966 |
| 11 | V16 | 0.375776718 | 0.240717043 | 0.99896391 | 0.3754 |
| 12 | V14 | 0.350770491 | 0.188761365 | 0.96913223 | 0.3399 |
| 13 | V20 | 0.516068203 | 0.368244149 | 0.643758213 | 0.3322 |
| 14 | V13 | 0.329301867 | 0.177964449 | 0.996756295 | 0.3282 |
| 15 | V22 | 0.246914241 | 0.126775392 | 0.793148855 | 0.1958 |
| 16 | V23 | 0.160105275 | 0.063556574 | 0.998305315 | 0.1598 |
| 17 | V11 | 0.307833138 | 0.256233801 | 0.43941776 | 0.1353 |
| 18 | V3 | 0.20019496 | 0.141891683 | 0.59415257 | 0.1189 |
| 19 | V10 | 0.112286107 | 0.03614281 | 0.99025664 | 0.1112 |
| 20 | V2 | 0.178407464 | 0.126017901 | 0.552594326 | 0.0986 |
| 21 | V4 | 0.024230545 | 0.007173375 | 0.874236464 | 0.0212 |
| 22 | V7 | 0.453407221 | 0.407106811 | 0.027597038 | 0.0125 |

**Chart between accuracy vs 1-entropy –**

**This is how calculation was done**

* **Example of how entropy was calculated for feature V2 and same applied across**



**Information gain(V2) - sum of**

**infogain(d)** - > infogain(1160,778)=entropy(1160/1938,778/1938) = -1160/1938\*log(1160/1938) - 778/1938\*log(778/1938) = 0.67

**infogain(g)** - > infogain(824,428)=entropy(824/1252,428/1252) = so on....

**infogain(l)** - > infogain(146,359)=entropy(146/505,359/505)

**infogain(m)** - > infogain(150,24)=entropy(150/174,24/174)

**infogain(p)** - > infogain(85,617)=entropy(85/702,617/702)

**infogain(u)** - > infogain(54,166)=entropy(54/220,166/220)

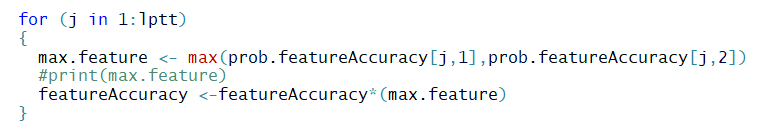
**infogain(w)** - > infogain(116,0)=entropy(116/116,0/116)

**Expected Infogain(V2) - 1938/4907 \* 0.67 + ........**

**Similar calculation applied across all features.**

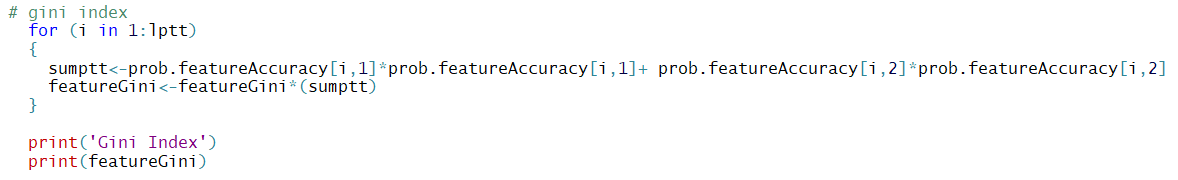
* **Accuracy**

**Feature Accuracy = Max(class(i)) / size of class**



* **Gini index**

Gini = sum [ weighted \* probability of class ]

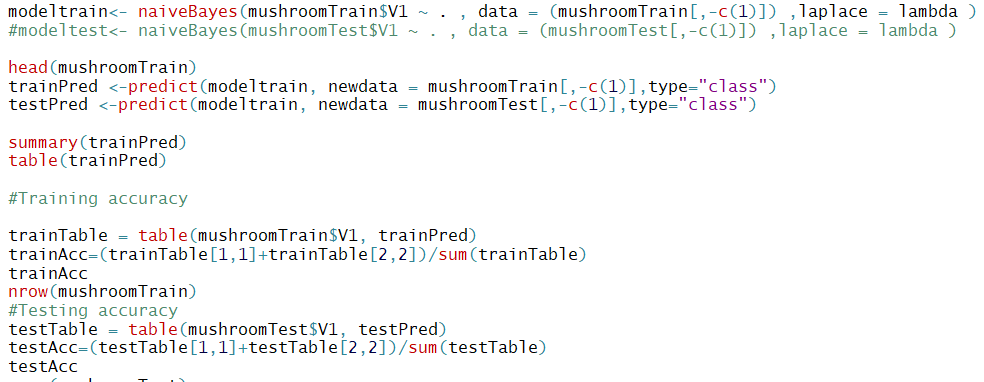


---------------------------------------------end of P2--------------------------------------------------------

**P3.**

1. **Result of naïve bayes as lambda increases accuracy goes down. After like lambda = 14-15 , accuracy goes down pretty much.**

**Code used to run**



**Results**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Lambda | trainAcc | testAcc |
| 1 | 0 | 0.94171592 | 0.939073671 |
| 2 | 1 | 0.94986754 | 0.95057507 |
| 3 | 2 | 0.94334624 | 0.94156046 |
| 4 | 3 | 0.94110454 | 0.939695368 |
| 5 | 4 | 0.93845527 | 0.936897731 |
| 6 | 5 | 0.9378439 | 0.935654336 |
| 7 | 6 | 0.93600978 | 0.933789245 |
| 8 | 7 | 0.93458325 | 0.930991607 |
| 9 | 8 | 0.93417567 | 0.929748213 |
| 10 | 9 | 0.93437946 | 0.928815667 |
| 11 | 10 | 0.93336051 | 0.928815667 |
| 12 | 11 | 0.93315671 | 0.92819397 |
| 13 | 12 | 0.93295292 | 0.92819397 |
| 14 | 13 | 0.93254534 | 0.927883121 |
| 15 | 14 | 0.93234155 | 0.927572272 |
| 16 | 15 | 0.93213776 | 0.927261424 |
| 17 | 16 | 0.9313226 | 0.926639726 |
| 18 | 17 | 0.9313226 | 0.926639726 |
| 19 | 18 | 0.93071123 | 0.926018029 |
| 20 | 19 | 0.93030365 | 0.925707181 |
| 21 | 20 | 0.93030365 | 0.925085483 |
| 22 | 21 | 0.93030365 | 0.925085483 |
| 23 | 22 | 0.93009986 | 0.924774635 |
| 24 | 23 | 0.93009986 | 0.924463786 |
| 25 | 24 | 0.92969228 | 0.924463786 |
| 26 | 25 | 0.9290809 | 0.924463786 |
| 27 | 26 | 0.92867332 | 0.924463786 |
| 28 | 27 | 0.92846953 | 0.923842089 |
| 29 | 28 | 0.92826574 | 0.92353124 |
| 30 | 29 | 0.92806195 | 0.922909543 |
| 31 | 30 | 0.92785816 | 0.922909543 |
| 32 | 31 | 0.92765437 | 0.922909543 |
| 33 | 32 | 0.927043 | 0.922909543 |
| 34 | 33 | 0.92683921 | 0.922598694 |
| 35 | 34 | 0.92683921 | 0.922287846 |
| 36 | 35 | 0.92663542 | 0.921976997 |
| 37 | 36 | 0.92622784 | 0.921666149 |
| 38 | 37 | 0.92602405 | 0.920733603 |
| 39 | 38 | 0.92520889 | 0.919801057 |
| 40 | 39 | 0.92439372 | 0.918868511 |
| 41 | 40 | 0.92439372 | 0.918868511 |
| 42 | 41 | 0.92398614 | 0.918246814 |
| 43 | 42 | 0.92378235 | 0.918246814 |
| 44 | 43 | 0.92357856 | 0.918246814 |
| 45 | 44 | 0.92337477 | 0.917935965 |
| 46 | 45 | 0.92317098 | 0.917935965 |
| 47 | 46 | 0.92317098 | 0.917935965 |
| 48 | 47 | 0.92317098 | 0.917935965 |
| 49 | 48 | 0.92317098 | 0.917935965 |
| 50 | 49 | 0.92317098 | 0.917625117 |

1. **Decision tree**

**Result by decision tree remains as below. Does not seem like size threshold has much impact on accuracy.**

sizeThreshold trainAcc testAcc

[1,] 4 0.9981659 0.9978241

[2,] 8 0.9981659 0.9978241

[3,] 12 0.9981659 0.9978241

[4,] 16 0.9981659 0.9978241

[5,] 20 0.9981659 0.9978241

[6,] 24 0.9981659 0.9978241

[7,] 28 0.9981659 0.9978241

[8,] 32 0.9981659 0.9978241

[9,] 36 0.9981659 0.9978241

[10,] 40 0.9981659 0.9978241

[11,] 44 0.9981659 0.9978241

[12,] 48 0.9981659 0.9978241

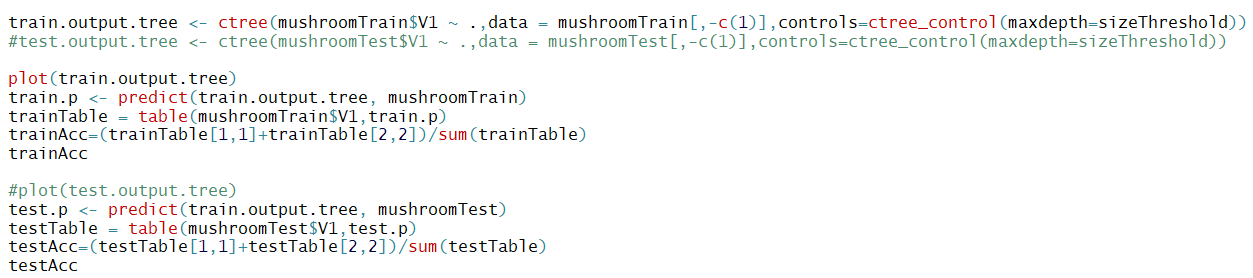
[13,] 52 0.9981659 0.9978241

[14,] 56 0.9981659 0.9978241

[15,] 60 0.9981659 0.9978241

[16,] 64 0.9981659 0.9978241

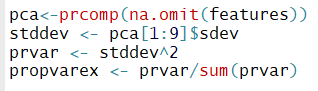
**Code used to run decision tree.**

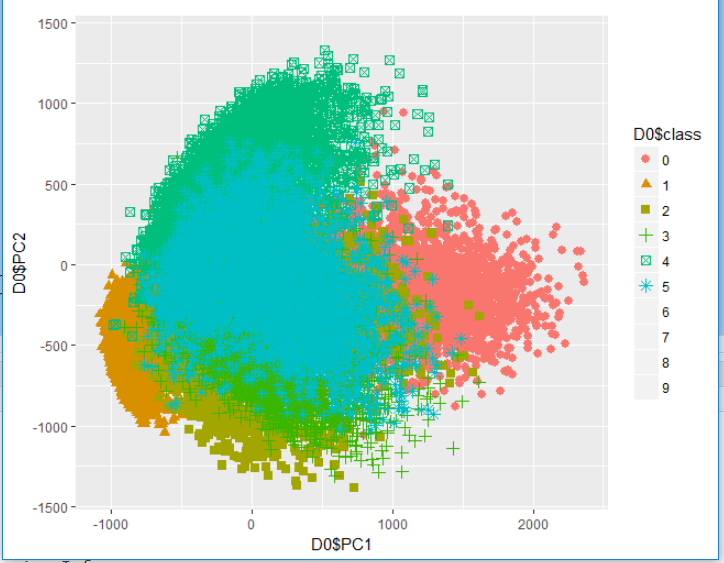


**--------------------------------------end of P3---------------------------------------------------------**

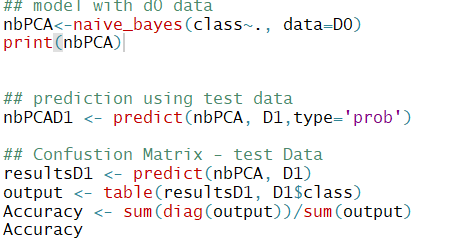
**P4.**

**PCA project plotted using MNIST data. 9 PC dimensions -**





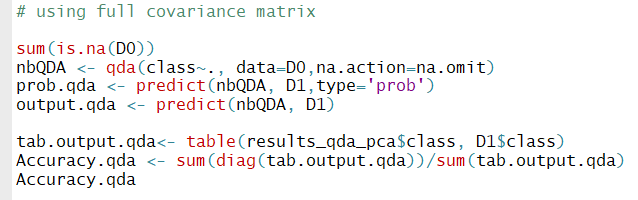
* **Classifier using diagonal covariance matrix using PCA**



**[1] 0.7539066**

**Accuracy comes around 75%**

* **Classifier using full covariance matrix using PCA**

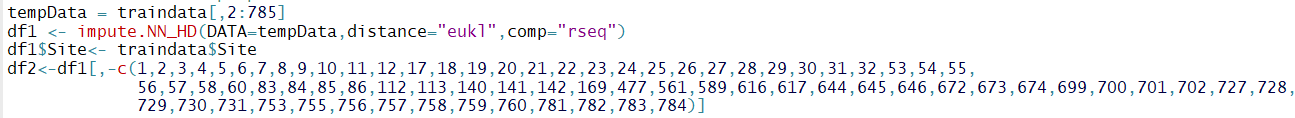


**[1] 0.8716866**

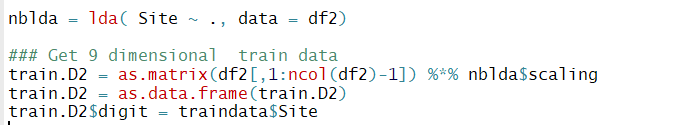
**Accuracy comes around 87%**

**FISHER projection using 9 dimensional LDA.**

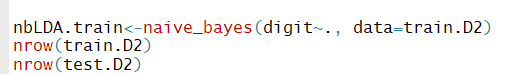
**First we imputed data and removed collinear variables before applying LDA.**



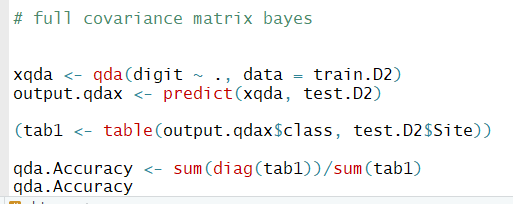
* **Classifier with diagonal matrix using LDA**



**Now we prepared test data and predict using model trained above in lda.**



* **Classifier with full covariance matrix using QDA**

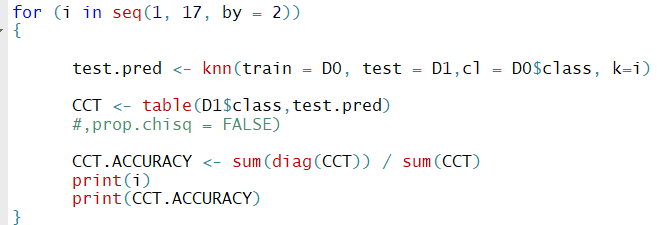


-------------------------------------------------end of P4------------------------------------------------------

**P5.**

* **KNN :**

**Applied KNN and across various K values = 1,3,5…..17.**



**Results : so looks like after K = 5 , accuracy almost got stable at 91%.**

[1] 1

[1] 0.895309

[1] 3

[1] 0.9079354

[1] 5

[1] 0.9114056

[1] 7

[1] 0.9109875

[1] 9

[1] 0.9114474

[1] 11

[1] 0.9109875

[1] 13

[1] 0.9111966

[1] 15

[1] 0.9107367

[1] 17

[1] 0.9105694

* **PARZEN window**

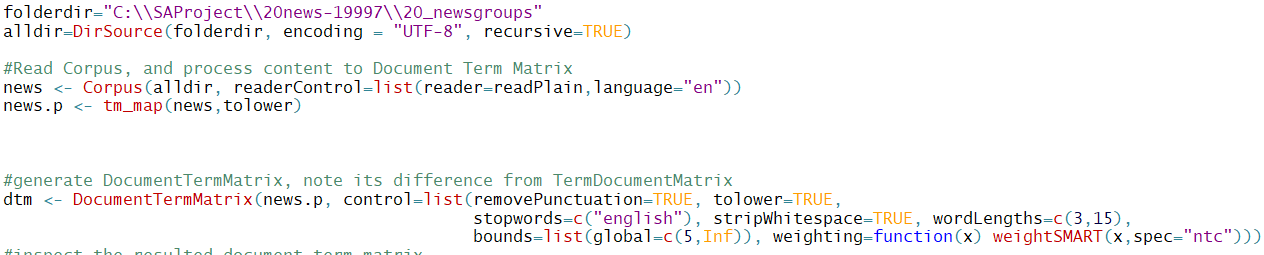
**Could not complete it.**

**----------------------------------------------------------end of P5-------------------------------------------**

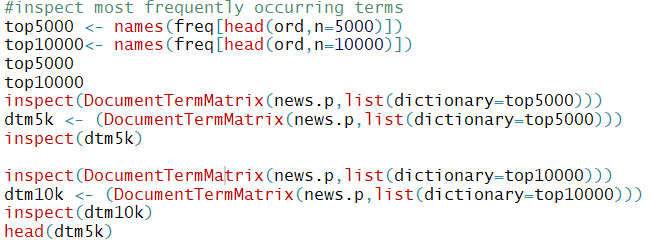
**P6.**

* **DICTIONARY:**
  + **Compute the document frequency of all words (how many documents each word occurred in)**
  + **Sort this in descending order of document frequency**
  + **Pick the top 5000 and 10000 words as the dictionary.**

**To solve above , we created corpus and document term matrix –**



**And order them to pick top 5000 and top 10000**



**Top 5000 words – few of them**

> inspect(dtm5k)

<<DocumentTermMatrix (documents: 19997, terms: 5000)>>

Non-/sparse entries: 2412945/97572055

Sparsity : 98%

Maximal term length: 15

Weighting : term frequency (tf)

Sample :

Terms

Docs and are com for news not that the this you

38403 150 76 2 117 16 58 85 362 91 190

39078 150 76 2 123 16 59 86 359 91 190

39638 152 77 2 120 16 61 89 361 91 189

53468 5 2 9 0 1 3 4 8 3 4

53569 2 2 3 3 2 1 6 6 1 3

66322 167 79 9 101 5 50 78 551 90 46

67882 176 51 15 93 30 39 82 401 73 171

75881 3 0 0 1 4 0 0 6 0 3

76392 391 26 1 73 2 41 239 549 44 91

76479 0 0 0 2 1 0 0 1 0 1

**Top 10000 terms.- few of them.**

> inspect(dtm10k)

<<DocumentTermMatrix (documents: 19997, terms: 10000)>>

Non-/sparse entries: 2657374/197312626

Sparsity : 99%

Maximal term length: 15

Weighting : term frequency (tf)

Sample :

Terms

Docs and are com for news not that the this you

38403 150 76 2 117 16 58 85 362 91 190

39078 150 76 2 123 16 59 86 359 91 190

39638 152 77 2 120 16 61 89 361 91 189

53468 5 2 9 0 1 3 4 8 3 4

53569 2 2 3 3 2 1 6 6 1 3

66322 167 79 9 101 5 50 78 551 90 46

67882 176 51 15 93 30 39 82 401 73 171

75881 3 0 0 1 4 0 0 6 0 3

76392 391 26 1 73 2 41 239 549 44 91

76479 0 0 0 2 1 0 0 1 0 1

* **Learn w/c , apply Laplacian and accuracy.**

**First we align the corpus and label them with folder name i.e. sport data with sport label , then we apply naïve bayes with lambda = 30 to build our prediction model. I could able to finish upto labelling as below.**





