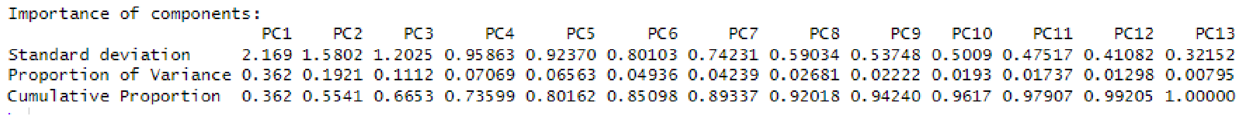
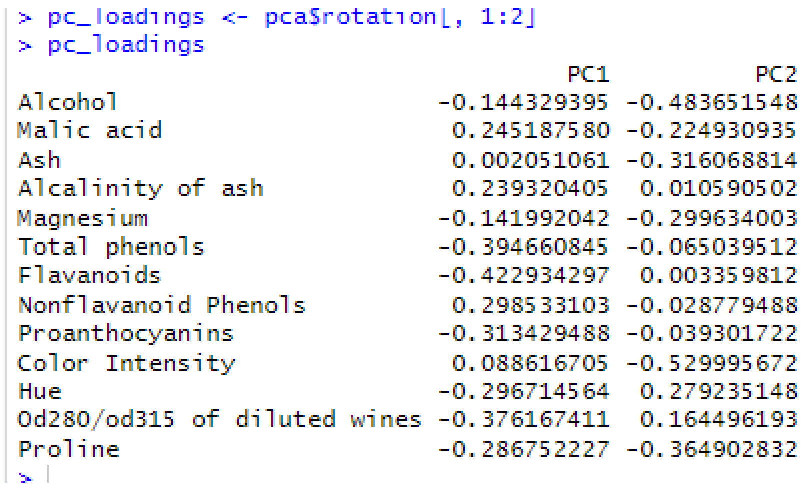
Principle components



```
Importance of components:
                         PC1    PC2    PC3     PC4     PC5     PC6     PC7     PC8     PC9   PC10    PC11    PC12    PC13
Standard deviation     2.169 1.5802 1.2025 0.95863 0.92370 0.80103 0.74231 0.59034 0.53748 0.5009 0.47517 0.41082 0.32152
Proportion of Variance 0.362 0.1921 0.1112 0.07069 0.06563 0.04936 0.04239 0.02681 0.02222 0.0193 0.01737 0.01298 0.00795
Cumulative Proportion  0.362 0.5541 0.6653 0.73599 0.80162 0.85098 0.89337 0.92018 0.94240 0.9617 0.97907 0.99205 1.00000
```

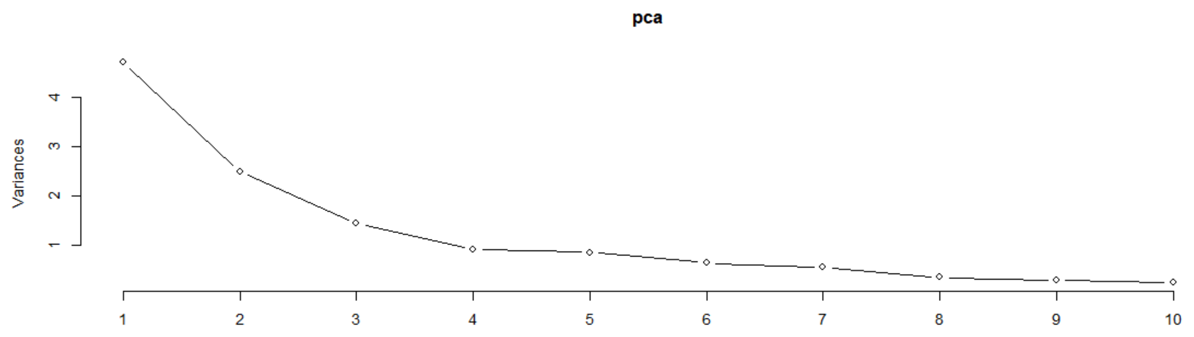
Contributions by original variables to top 2 principle components

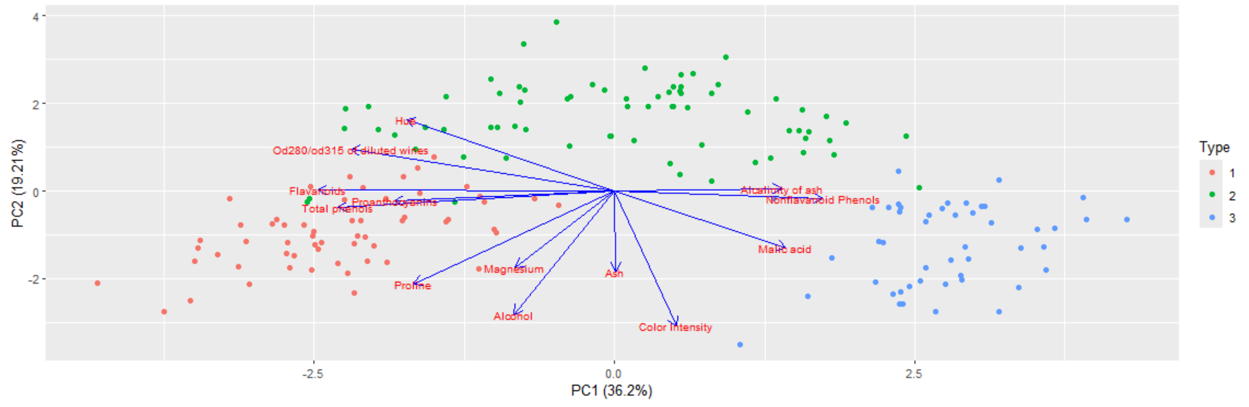


```
> pc_loadings <- pca$rotation[, 1:2]
> pc_loadings
                                    PC1          PC2
Alcohol                    -0.144329395 -0.483651548
Malic acid                  0.245187580 -0.224930935
Ash                         0.002051061 -0.316068814
Alcalinity of ash           0.239320405  0.010590502
Magnesium                  -0.141992042 -0.299634003
Total phenols              -0.394660845 -0.065039512
Flavanoids                 -0.422934297  0.003359812
Nonflavanoid Phenols        0.298533103 -0.028779488
Proanthocyanins            -0.313429488 -0.039301722
Color Intensity             0.088616705 -0.529995672
Hue                        -0.296714564  0.279235148
Od280/od315 of diluted wines -0.376167411  0.164496193
Proline                    -0.286752227 -0.364902832
> 
```

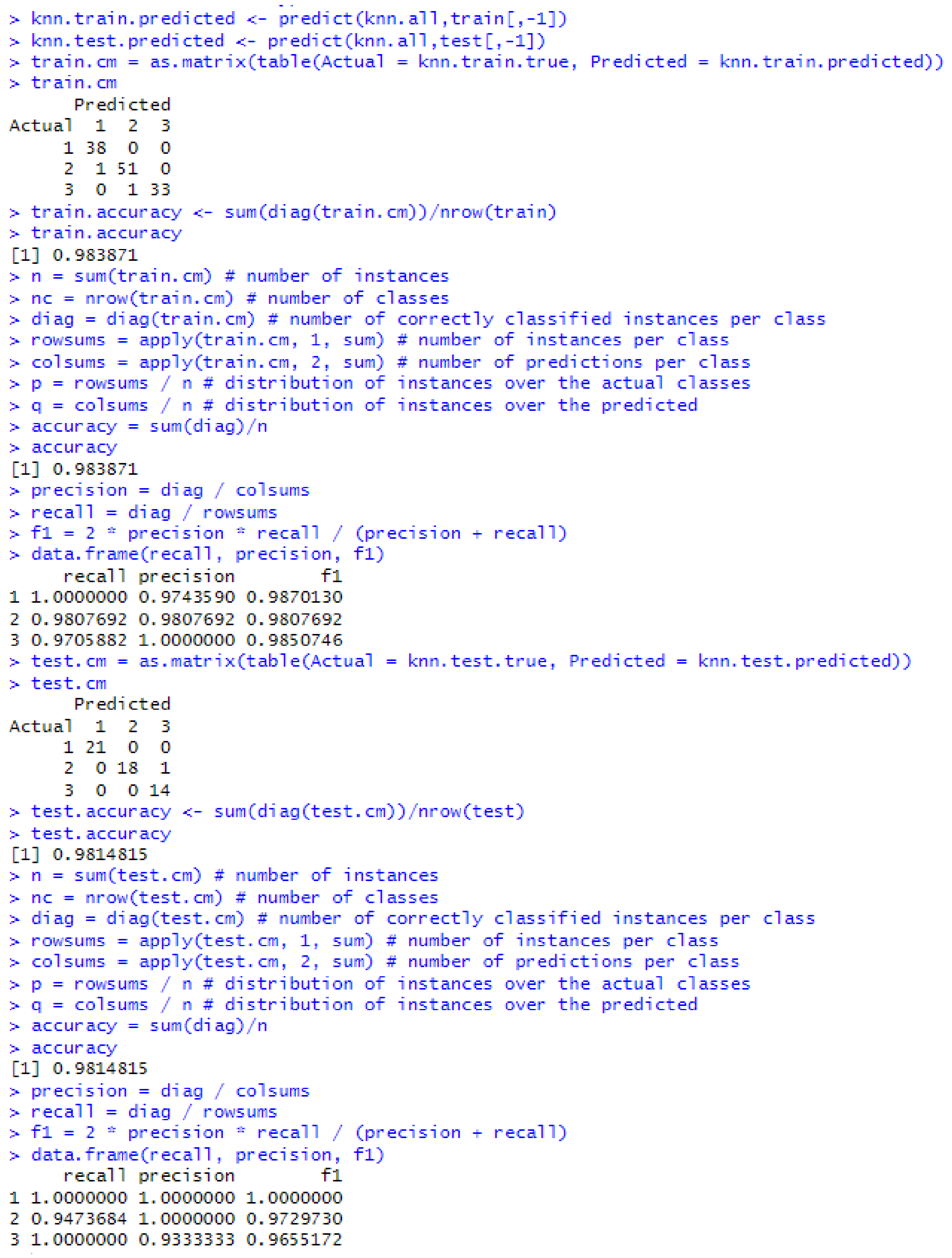
As we can see flavanoids, total phenols, and Od280/od315 of diluted wines contribute the most to PC1, while Alcohol, Proline, and Color Intensity contribute the most to PC2

Plot of principle components





KNN evaluation with use of all original features



```
> knn.train.predicted <- predict(knn.all,train[,-1])
> knn.test.predicted <- predict(knn.all,test[,-1])
> train.cm = as.matrix(table(Actual = knn.train.true, Predicted = knn.train.predicted))
> train.cm
      Predicted
Actual  1  2  3
     1 38  0  0
     2  1 51  0
     3  0  1 33
> train.accuracy <- sum(diag(train.cm))/nrow(train)
> train.accuracy
[1] 0.983871
> n = sum(train.cm) # number of instances
> nc = nrow(train.cm) # number of classes
> diag = diag(train.cm) # number of correctly classified instances per class
> rowsums = apply(train.cm, 1, sum) # number of instances per class
> colsums = apply(train.cm, 2, sum) # number of predictions per class
> p = rowsums / n # distribution of instances over the actual classes
> q = colsums / n # distribution of instances over the predicted
> accuracy = sum(diag)/n
> accuracy
[1] 0.983871
> precision = diag / colsums
> recall = diag / rowsums
> f1 = 2 * precision * recall / (precision + recall)
> data.frame(recall, precision, f1)
     recall precision        f1
1 1.0000000 0.9743590 0.9870130
2 0.9807692 0.9807692 0.9807692
3 0.9705882 1.0000000 0.9850746
> test.cm = as.matrix(table(Actual = knn.test.true, Predicted = knn.test.predicted))
> test.cm
      Predicted
Actual  1  2  3
     1 21  0  0
     2  0 18  1
     3  0  0 14
> test.accuracy <- sum(diag(test.cm))/nrow(test)
> test.accuracy
[1] 0.9814815
> n = sum(test.cm) # number of instances
> nc = nrow(test.cm) # number of classes
> diag = diag(test.cm) # number of correctly classified instances per class
> rowsums = apply(test.cm, 1, sum) # number of instances per class
> colsums = apply(test.cm, 2, sum) # number of predictions per class
> p = rowsums / n # distribution of instances over the actual classes
> q = colsums / n # distribution of instances over the predicted
> accuracy = sum(diag)/n
> accuracy
[1] 0.9814815
> precision = diag / colsums
> recall = diag / rowsums
> f1 = 2 * precision * recall / (precision + recall)
> data.frame(recall, precision, f1)
     recall precision        f1
1 1.0000000 1.0000000 1.0000000
2 0.9473684 1.0000000 0.9729730
3 1.0000000 0.9333333 0.9655172
```

KNN evaluation with top 2 principle components

```
> knn.train.predicted <- predict(knn.pca,train[,-3])
> knn.test.predicted <- predict(knn.pca,test[,-3])
> train.cm = as.matrix(table(Actual = knn.train.true, Predicted = knn.train.predicted))
> train.cm
      Predicted
Actual  1  2  3
     1 44  1  0
     2  1 47  1
     3  0  0 30
> train.accuracy <- sum(diag(train.cm))/nrow(train)
> train.accuracy
[1] 0.9758065
> n = sum(train.cm) # number of instances
> nc = nrow(train.cm) # number of classes
> diag = diag(train.cm) # number of correctly classified instances per class
> rowsums = apply(train.cm, 1, sum) # number of instances per class
> colsums = apply(train.cm, 2, sum) # number of predictions per class
> p = rowsums / n # distribution of instances over the actual classes
> q = colsums / n # distribution of instances over the predicted
> accuracy = sum(diag)/n
> accuracy
[1] 0.9758065
> precision = diag / colsums
> recall = diag / rowsums
> f1 = 2 * precision * recall / (precision + recall)
> data.frame(recall, precision, f1)
     recall precision        f1
1 0.9777778 0.9777778 0.9777778
2 0.9591837 0.9791667 0.9690722
3 1.0000000 0.9677419 0.9836066
> test.cm = as.matrix(table(Actual = knn.test.true, Predicted = knn.test.predicted))
> test.cm
      Predicted
Actual  1  2  3
     1 14  0  0
     2  2 20  0
     3  0  2 16
> test.accuracy <- sum(diag(test.cm))/nrow(test)
> test.accuracy
[1] 0.9259259
> n = sum(test.cm) # number of instances
> nc = nrow(test.cm) # number of classes
> diag = diag(test.cm) # number of correctly classified instances per class
> rowsums = apply(test.cm, 1, sum) # number of instances per class
> colsums = apply(test.cm, 2, sum) # number of predictions per class
> p = rowsums / n # distribution of instances over the actual classes
> q = colsums / n # distribution of instances over the predicted
> accuracy = sum(diag)/n
> accuracy
[1] 0.9259259
> precision = diag / colsums
> recall = diag / rowsums
> f1 = 2 * precision * recall / (precision + recall)
> data.frame(recall, precision, f1)
     recall precision        f1
1 1.0000000 0.8750000 0.9333333
2 0.9090909 0.9090909 0.9090909
3 0.8888889 1.0000000 0.9411765
```

Analysis/Comparison between PCA and Original Features

PCA performs worse overall than using all the original features, however it still does a really good job with accuracy and the other scores for using only 2 features. This shows that using principle component analysis can let us use less features but replicate nearly the same performance as the original features, which can help us in a lot of situations where greater number of features may lead to overfitting or other issues (such as it also improving time).