Bios/CS 534 Project 2

Chenxi Cai

May 6, 2018

1 Abstract

In this final project, a dataset of 1000 samples with 120 proteins features came from the blood plasma of some patients and controls. Some techniques were used to analyze this dataset in order to develop blood protein biomarkers for a certain type of cancer. First, the Principal component analysis (PCA) was used for dimension reduction and the visualizations showed there appeared to be subgroups of proteins, whereas no subgroups of samples. Then Bayesian Information Criterion (BIC) estimated the number of clusters in K-Means and AgglomerativeClustering clustering techniques. These two methods have similar results after clustering the proteins. Finally, the dataset was split in a 3:1:1 ratio into training, validation, and testing sets. Four classifiers: KNeighborsClassifier, RandomForestClassifier, GradientBoostingClassifier and LogisticRegressionCV were tuned with cross-validation GridSearchCV using the training data and GradientBoostingClassifier is the best one based on its performance on the validation data. Then I evaluated the model with GradientBoostingClassifier and got the error rate 0.3. I also got the conclusion, based on feature importance ranking, that protein 74th, 34th and 54th are the top predictors.

2 Methods

2.1 Dimension Reduction Technique

I used Principal component analysis (PCA) technique here. It is a linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space.[1] The class is $''sklearn.decomposition.PCA''$ and the parameter needeeds to set is the **n_components**, which is number of components to keep. Here we set n_components = 2.

2.2 Clustering

K-means and AgglomerativeClustering were applied here. The KMeans algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares.[2]. Hierarchical clustering is a general family of clustering algorithms that build nested clusters by merging or splitting them successively.[3] The AgglomerativeClustering object performs a hierarchical clustering using a bottom up approach: each observation starts in its own cluster, and clusters are successively merged together. These two algorithms require the number of clusters to be specified. The classes are "$sklearn.cluster.KMeans$" and "$sklearn.cluster.AgglomerativeClustering$". The parameter of the number of clusters is **n_clusters**. I used Bayesian Information Criterion (BIC) method to visualize the data character and then determined the number of clusters K. As shown in Fig. 1, the number of clusters should be set to 3. n_clusters = 3.

2.3 Classifier

The dataset was split in a 3:1:1 ratio into training, validation, and testing sets. With training data and GridSearchCV cross-validation method, I tuned KNeighborsClassifier, RandomForestClassifier and GradientBoostingClassifier. The GridSearchCV is an exhaustive search over specified parameter values for an estimator.[4] Neighbors-based classification is a type of instance-based learning or non-

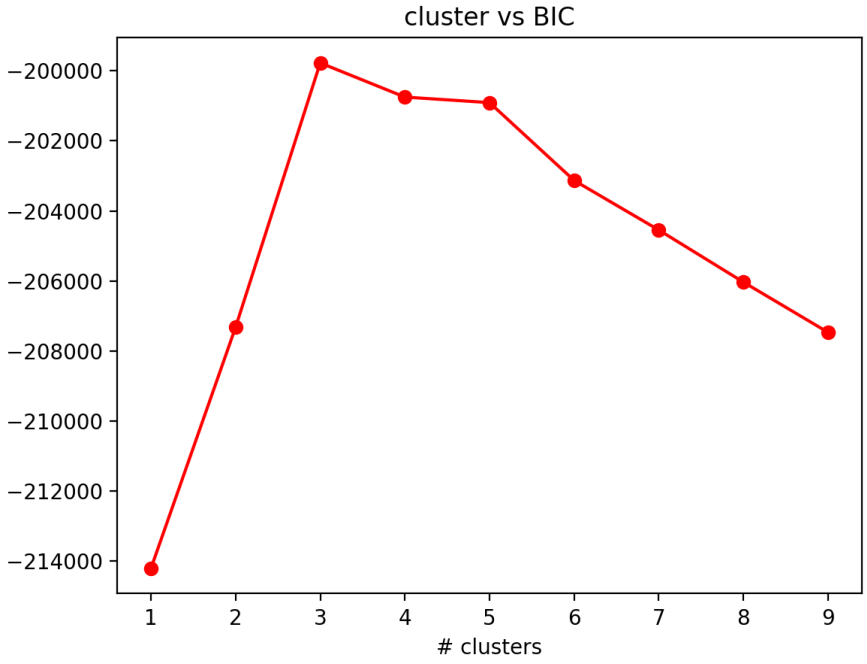

Figure 1: Plots of BIC

generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data.[5] RandomForestClassifier is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.[6] GradientBoostingClassifier builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions.[7] The best parameters after tuning of each classifiers are list in Table.1. The parameter **cv** is set to 10 for all classifiers.

Table 1: Best Parameters of Each Classifiers

Classifier	best parameter
KNeighborsClassifier	n_neighbors=2
RandomForestClassifier	n_estimators=510
GradientBoostingClassifier	learning_rate=0.52

The tuning procedure and selection of parameters is in Supplementary File 1.

I also use LogisticRegressionCV classifier[8], the class is "*sklearn.linear_model.LogisticRegressionCV*" and the parameter **cs** is set by default to be ten values in a logarithmic scale between 1e-4 and 1e4.

3 Results and Discussion

3.1 Dimension Reduction Technique

With PCA, I did twice (a) for the samples; (b) for the proteins. And the 2-dimensional visualizations of them are shown in Fig. 2 and Fig. 3, separately. From Fig. 2, we can know that there appears to be no subgroups of sample. All the points are messy. However, from Fig. 3 we can see that there are three main subgroups of proteins and the points in center may be noise points. Those subgroups are not associated with the clinical outcome because the outcomes are binary values and there are only two classes, while there appears three subgroups in proteins.

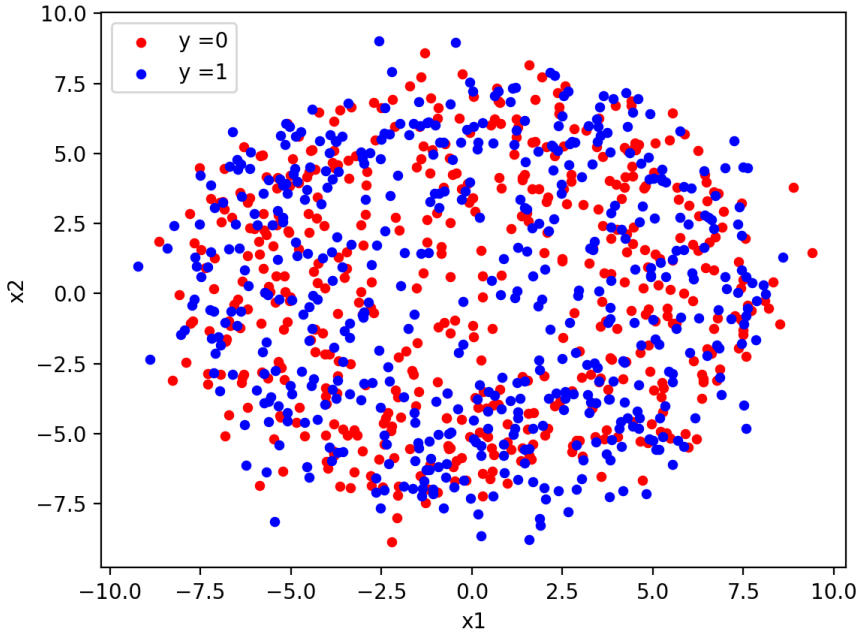

Figure 2: Dimension Reduction Visualization of Samples

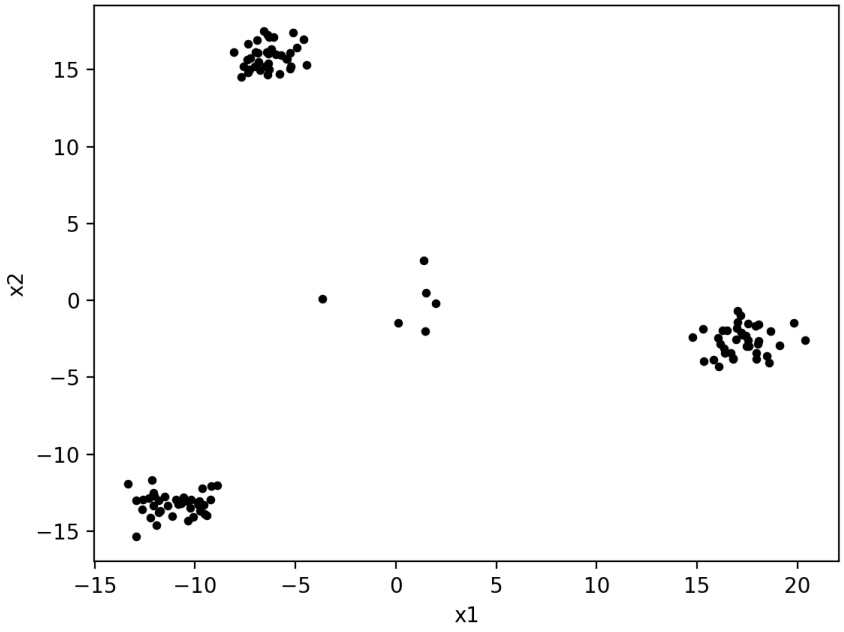

Figure 3: Dimension Reduction Visualization of Proteins

3.2 Clustering

By calculating BIC values, we set the number of cluster K to 3 in K-means and AgglomerativeClustering, which is shown in Fig. 1. The predict labels of proteins are shown in Table.2.

Only 117th protein is clustered in different clusters by Kmeans and AgglomerativeClustering. The two methods yield very similar results.

3.3 Classifier

After tuning each KNeighborsClassifier, RandomForestClassifier and GradientBoostingClassifier with cross-validation using the training data, the feature rankings of RandomForestClassifier and GradientBoostingClassifier are show in Table. 4. We can know that the 74th, 34th and 54th proteins are the top predictors.

I also used LogisticRegressionCV classifier. The scores of validation data based on all these four

Table 2: 120 Predicted Labels of Kmeans and AgglomerativeClustering

numbers	Kmeans	Agglomerative
1	1	2
⋮	⋮	⋮
38	1	2
39	2	1
⋮	⋮	⋮
76	2	1
78	0	0
⋮	⋮	⋮
117	1	0
118	0	0
⋮	⋮	⋮
120	0	0

classifiers are listed in Table. 3. We can draw the conclusion that GradientBoostingClassifier is the best classifier.

Then using the testing data, the performance of the GradientBoostingClassifier can be showed by score, which is 0.7. So the error rate is 0.3.

Table 3: Scores of Four Classifiers

Classifier	Scores
KNeighborsClassifier	0.57
RandomForestClassifier	0.68
GradientBoostingClassifier	0.74
LogisticRegressionCV	0.505

Table 4: Protein Importance Ranking

rank	RandomForest		GradientBoosting	
	index of proteins	**importances**	**index of proteins**	**importances**
1	74	0.041290	74	0.061533
2	34	0.040276	34	0.059901
3	54	0.024249	54	0.042605
4	99	0.012785	14	0.023738
5	71	0.011016	109	0.021023
6	46	0.010563	24	0.017180
7	5	0.010535	94	0.015839
8	51	0.010214	59	0.014754
9	14	0.010142	63	0.014634
10	84	0.010130	92	0.014601
11	9	0.009541	20	0.014460
12	90	0.009364	53	0.013950
13	76	0.009331	65	0.013924
14	79	0.009252	81	0.013860
15	110	0.009147	37	0.013822
16	55	0.009135	5	0.013655
17	100	0.008991	116	0.013440
18	92	0.008876	33	0.012900
19	53	0.008810	113	0.012848
20	107	0.008809	19	0.012110
21	20	0.008807	97	0.011919
22	97	0.008726	21	0.011645

Continued on next page

Table 4 – continued from previous page

rank	RandomForest		GradientBoosting	
	index of proteins	**importances**	**index of proteins**	**importances**
23	44	0.008691	2	0.011479
24	41	0.008687	87	0.011190
25	6	0.008677	46	0.010907
26	105	0.008659	115	0.010212
27	75	0.008644	99	0.010203
28	65	0.008474	66	0.010202
29	61	0.008437	60	0.010168
30	28	0.008361	7	0.010051
31	15	0.008324	75	0.009972
32	48	0.008238	67	0.009945
33	31	0.008187	73	0.009844
34	73	0.008170	105	0.009712
35	93	0.008064	84	0.009660
36	111	0.008034	62	0.009635
37	87	0.007999	70	0.009555
38	117	0.007985	111	0.009513
39	27	0.007967	11	0.009510
40	2	0.007951	15	0.009481
41	69	0.007940	17	0.009348
42	109	0.007851	78	0.009287
43	72	0.007843	51	0.009266
44	67	0.007830	31	0.009201
45	88	0.007810	29	0.009095

Continued on next page

Table 4 – continued from previous page

rank	RandomForest		GradientBoosting	
	index of proteins	**importances**	**index of proteins**	**importances**
46	29	0.007808	9	0.009044
47	0	0.007807	18	0.008933
48	32	0.007799	32	0.008920
49	116	0.007742	61	0.008718
50	21	0.007738	91	0.008648
51	101	0.007710	100	0.008641
52	59	0.007706	28	0.008447
53	39	0.007654	58	0.008376
54	40	0.007592	76	0.008371
55	33	0.007585	107	0.007643
56	19	0.007560	93	0.007606
57	119	0.007548	22	0.007597
58	45	0.007523	95	0.007158
59	70	0.007506	79	0.007112
60	50	0.007461	1	0.007038
61	95	0.007451	72	0.006953
62	66	0.007422	0	0.006749
63	106	0.007416	10	0.006659
64	30	0.007364	48	0.006451
65	63	0.007332	39	0.006217
66	23	0.007329	44	0.006201
67	60	0.007328	71	0.006098
68	104	0.007300	102	0.005870

Continued on next page

Table 4 – continued from previous page

rank	RandomForest		GradientBoosting	
	index of proteins	**importances**	**index of proteins**	**importances**
69	49	0.007299	86	0.005755
70	25	0.007254	89	0.005743
71	38	0.007250	82	0.005696
72	52	0.007246	36	0.005570
73	57	0.007236	90	0.005562
74	37	0.007162	108	0.005267
75	115	0.007151	68	0.005057
76	94	0.007132	101	0.005011
77	58	0.007130	56	0.004981
78	83	0.007129	110	0.004787
79	103	0.007116	88	0.004385
80	102	0.007105	16	0.004247
81	80	0.007027	27	0.004153
82	1	0.006986	43	0.004122
83	85	0.006936	57	0.003839
84	96	0.006911	55	0.003818
85	78	0.006903	13	0.003715
86	108	0.006875	85	0.003683
87	64	0.006873	83	0.003669
88	82	0.006847	119	0.003620
89	17	0.006782	26	0.003616
90	13	0.006781	30	0.003600
91	91	0.006773	98	0.003413

Continued on next page

Table 4 – continued from previous page

rank	RandomForest		GradientBoosting	
	index of proteins	**importances**	**index of proteins**	**importances**
92	113	0.006764	25	0.003366
93	8	0.006742	12	0.003357
94	18	0.006698	6	0.003322
95	26	0.006658	47	0.003127
96	62	0.006658	117	0.003115
97	35	0.006654	49	0.003096
98	47	0.006627	41	0.002864
99	56	0.006626	103	0.002860
100	24	0.006604	35	0.002667
101	11	0.006596	50	0.002574
102	114	0.006593	38	0.002545
103	42	0.006576	80	0.002452
104	3	0.006571	52	0.002249
105	89	0.006567	40	0.002220
106	12	0.006556	23	0.002145
107	10	0.006492	104	0.002026
108	81	0.006452	96	0.001766
109	43	0.006348	3	0.001555
110	86	0.006330	64	0.001375
111	77	0.006319	106	0.001129
112	16	0.006282	114	0.000605
113	118	0.006282	69	0.000432
114	68	0.006253	112	0.000360

Continued on next page

Table 4 – continued from previous page

rank	RandomForest		GradientBoosting	
	index of proteins	**importances**	**index of proteins**	**importances**
115	112	0.006166	4	0.000222
116	4	0.006128	45	0.000032
117	98	0.005962	8	0.000000
118	7	0.005894	118	0.000000
119	36	0.005721	42	0.000000
120	22	0.005487	77	0.000000

4 Conclusion

In this final project, I did dimension reduction with PCA and found there were three subgroups in proteins and they had no relationship with the clinical outcome. Then the BIC values estimates that there are three clusters in K-Means and AgglomerativeClustering clustering techniques. Only 117th protein has different results with two clustering methods. Finally, the protein importance ranking by RandomForestClassifier and GradientBoostingClassifier showed that the 74th, 34th and 54th proteins are the top predictors. Compared with other two classifiers, KNeighborsClassifier and LogisticRegressionCV, GradientBoostingClassifier is the best one based on the performance on validation data. After evaluating the dataset with GradientBoostingClassifier, the the mean accuracy on the given testing data and labels is 0.7. Error rate is 0.3.

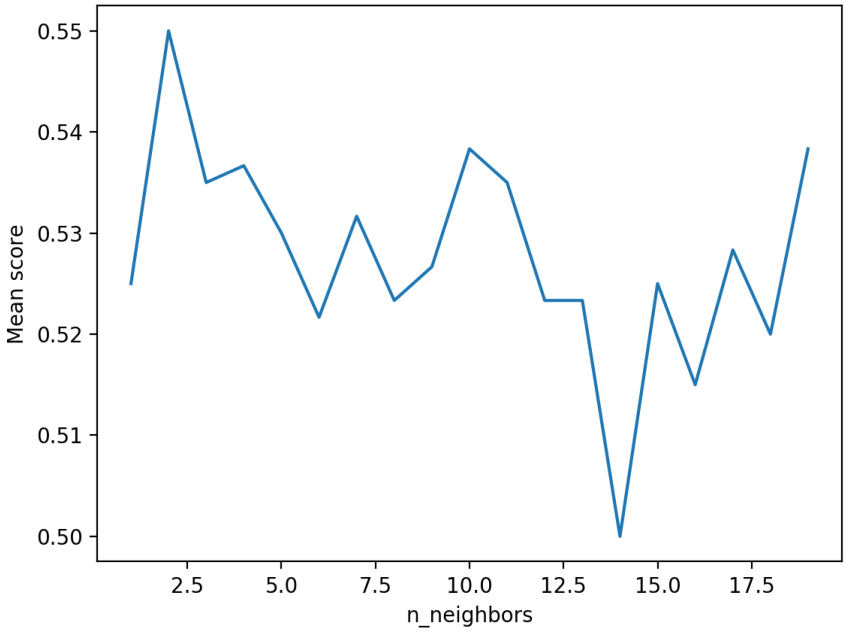

Figure 4: KNeighborsClassifier Tuning

Supplementary File 1

This part shows details of tuning classifiers. When using GridSearchCV, the parameter **(estimator, param_grid, cv)** set for classifiers is shown in Table, 5. The plots of scores based on training data and the parameter of classifier that need to be tuned are Fig. 4, Fig. 5, Fig. 6. From these figures, I got the best parameter of each classifier shown in Table. 1.

Table 5: Parameters of GridSearchCV

Classifier	estimator, param_grid, cv
KNeighborsClassifier	KNeighborsClassifier(), [1,2,..,19], 10
RandomForestClassifier	RandomForestClassifier(oob_score = True), [10,110,...,910], 10
GradientBoostingClassifier	GradientBoostingClassifier(), [0.01, 0.02,..,0.99], 10

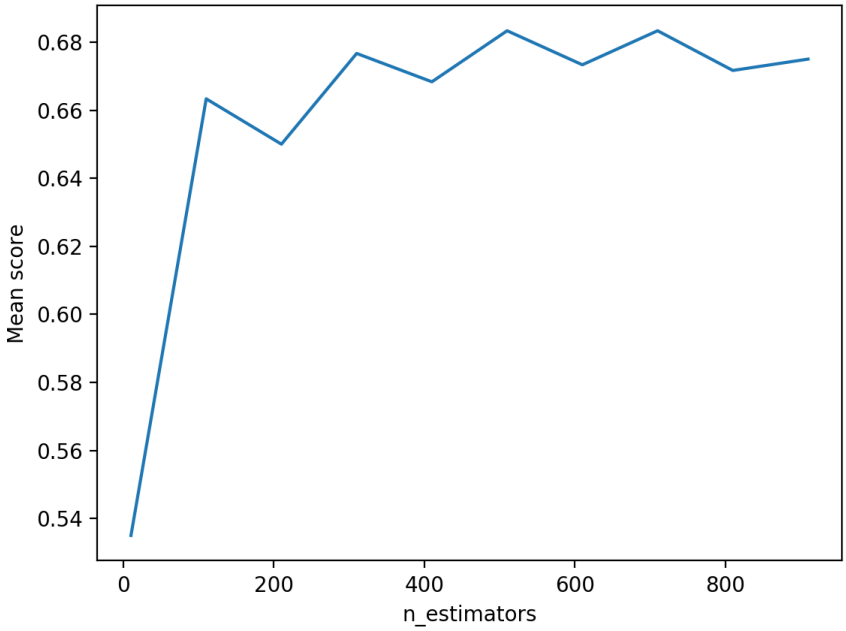

Figure 5: RandomForestClassifier Tuning

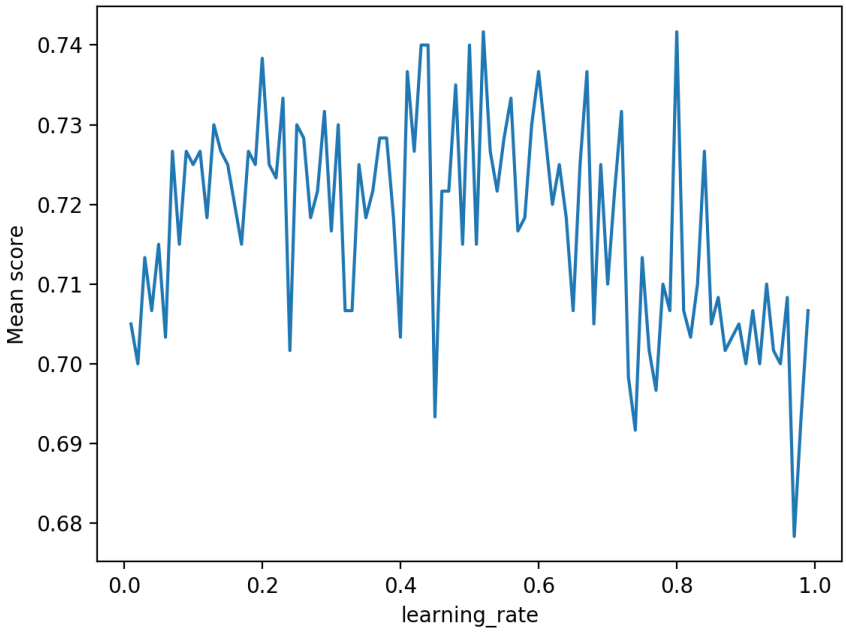

Figure 6: GradientBoostingClassifier Tuning

Supplementary File 2

This part includes all codes.

Python Codes of Final Project:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

from scipy.spatial import distance
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import AffinityPropagation

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegressionCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

def pcaprojection(X):
    '''
    :param X: Original data
    :1.Standardize the Data 2.Fit the model with X and apply the
    dimensionality reduction on X.
    :return: dimension reduced X
    '''
    #Standardize the Data
    X_stand = StandardScaler().fit_transform(X)

    #PCA Projection to 2D
```

```
29      pca = PCA(n_components=2)
30      X_new = pca.fit_transform(X_stand)
31
32      return X_new
33
34  def classifytarget(X,Y):
35      '''
36      :param X: dimension reduced X
37      :param Y: target Y array
38      :Classify new X into differnt class according to target Y
39      :return: n arrays with differnt target
40      '''
41      X_0 = []
42      X_1 = []
43      for i, y in enumerate(Y):
44          if (y == 0):
45              X_0.append(X[i,:]) #put all samples with target 0
46          else:
47              X_1.append(X[i,:]) ##put all samples with target 1
48
49      return X_0,X_1
50
51  def visualization1(X_0,X_1):
52      '''
53      :param X_0: all samples with target 0
54      :param X_1: all samples with target 1
55      :plot newdata with known targets
56      :return: plot of component 1 vs component 2
57      '''
58      X_0 = np.array(X_0)
59      X_1 = np.array(X_1) #convert list to array
60      #plot points
61      plt.scatter(X_0[:,0], X_0[:,1],15,
62              color='red', marker='o', label='y =0')
```

```
63      plt.scatter(X_1[:,0], X_1[:,1],15,
64      color='blue', marker='o', label='y =1')
65      plt.xlabel('x1')
66      plt.ylabel('x2')
67      plt.legend(loc='upper left')
68      plt.show()
69
70  def visualization2(X):
71      '''
72      :param X: all samples with unknown targets
73      :plot newdata with unknown targets
74      :return: plot of component 1 vs component 2
75      '''
76      #plot points
77      plt.scatter(X[:,0], X[:,1],10,
78              color='black', marker='o')
79      plt.xlabel('x1')
80      plt.ylabel('x2')
81      plt.legend()
82      plt.show()
83
84  def compute_bic(kmeans,X):
85      '''
86      :param kmeans: List of clustering object from scikit learn
87      :param X:  multidimension np array of data points
88      :Computes the BIC metric for a given clusters
89      :returns:BIC value
90      '''
91      # assign centers and labels
92      centers = [kmeans.cluster_centers_]
93      labels  = kmeans.labels_
94      #number of clusters
95      m = kmeans.n_clusters
96      # size of the clusters
```

```
97      n = np.bincount(labels)
98      #size of data set
99      N, d = X.shape
100
101     #compute variance for all clusters beforehand
102     cl_var = (1.0 / (N - m) / d) * sum([sum(distance.cdist(X[np.where(labels
        == i)], [centers[0][i]],
103             'euclidean')**2) for i in range(m)])
104
105     const_term = 0.5 * m * np.log(N) * (d+1)
106
107     BIC = np.sum([n[i] * np.log(n[i]) -
108               n[i] * np.log(N) -
109             ((n[i] * d) / 2) * np.log(2*np.pi*cl_var) -
110             ((n[i] - 1) * d/ 2) for i in range(m)]) - const_term
111
112     return(BIC)
113
114 def logisticregre(X_train,Y_train,X_vali,Y_vali,cv):
115     '''
116     :param X_train: train X
117     :param Y_train: train Y
118     :param X_vali: validaiton X
119     :param Y_vali: validaiton Y
120     :param cv: cross-validation generator
121     :Use LogisticRegressionCV classifiers, and tune each classifier with cross
        -validation using the training data.
122     :return: score of validation data
123     '''
124     logisticregre = LogisticRegressionCV(cv = cv)
125     logisticregre.fit(X_train, Y_train)
126     score = logisticregre.score(X_vali,Y_vali)
127     return score
128
```

```
129 def gridcv(X_train,Y_train,X_vali,Y_vali,cv,classifier,param_grid):
130     '''
131     :param X_train: train X
132     :param Y_train: train Y
133     :param X_train: validation X
134     :param Y_train: validation Y
135     :param cv: cross-validation generator
136     :param classifier: differnt classifiers
137     :param param_grid: parameter in grid
138     :Use gridsearchCV and tune each classifier with cross-validation using the
        training data.
139     :return: best_estimator
140     '''
141
142     grid = GridSearchCV(classifier,param_grid=param_grid,cv=cv)
143     #reshape Y
144     c, r = Y_train.shape
145     Y_train_new = Y_train.reshape(c,)
146
147     grid.fit(X_train, Y_train_new)
148     best_estimator = grid.best_estimator_
149     print("The best classifier is: ", best_estimator)
150
151     best_estimator.fit(X_train,Y_train)  #fit training data
152     score = best_estimator.score(X_vali,Y_vali) #calculate score with
        validation  data
153
154     return grid, best_estimator,score
155
156 def importances(best_estimator,n_features):
157     '''
158     :param best_estimator: best_estimator of classifier
159     :param n_features : number of proteins
160     :return: variable importance
```

```
161     '''
162     #importance and feature
163     importances = best_estimator.feature_importances_
164     indices = np.argsort(importances)[::-1]
165 
166     # Print the feature ranking
167     print("Feature ranking:")
168     for f in range(n_features):
169         print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[
        f]]))
170 
171 def plotclassifier(grid,para_range,label):
172     '''
173     :param grid: grid
174     :param C_range : range of C parameter
175     :param para_range : range of tuning parameter
176     :param label : string, name of parameter
177     :return: plot of score vs parameter
178     '''
179     scores = [x[1] for x in grid.grid_scores_]
180 
181     #plot
182     plt.plot(para_range, scores)
183 
184     plt.legend()
185     plt.xlabel(label)
186     plt.ylabel('Mean score')
187     plt.show()
188 
189 
190 #read files
191 f = pd.read_csv('/Users/ccai28/Desktop/CS/ML/Project4/project_data.txt',sep='\
        t', header = None, skiprows = 1)
192 
```

```
193 #read data
194 X = f.loc[:,1:120].values
195 Y = f.loc[:,[121]].values
196
197 #split arrays or matrices into random train ,vaidation and test subsets
198 X_train, X_nontrain, Y_train, Y_nontrain  = train_test_split(X,Y,test_size =
        0.4,random_state=42)
199 X_vali, X_test, Y_vali, Y_test = train_test_split(X_nontrain,Y_nontrain,
        test_size = 0.5)
200 X_trans = np.transpose(X)
201
202 '''
203 ---------------------- Question 1 ----------------------
204 '''
205 #for the subjects
206 X_new_s = pcaprojection(X)
207 X_0_s,X_1_s = classifytarget(X_new_s,Y)
208 visualization1(X_0_s,X_1_s)
209
210 #for the proteins
211 X_new_p = pcaprojection(X_trans)
212 visualization2(X_new_p)
213
214 '''
215 ---------------------- Question 2 ----------------------
216 '''
217
218 ks = range(1,10)
219
220 # run 9 times kmeans and save each result in the KMeans object
221 kMeans = [KMeans(n_clusters = i, init="k-means++").fit(X_trans) for i in ks]
222
223 # now run for each cluster the BIC computation
224 BIC = [compute_bic(kmeansi,X_trans) for kmeansi in kMeans]
```

```
225 print BIC
226
227 plt.plot(ks,BIC,'r-o')
228 plt.title("cluster vs BIC")
229 plt.xlabel("# clusters")
230 plt.ylabel("# BIC")
231 plt.show()
232
233 #k-means clustring
234 kmeans = KMeans(n_clusters=3).fit(X_trans)#n_clusters=3
235 # Fitting the input data and getting the cluster labels
236 labels_k = kmeans.labels_
237 print labels_k
238
239 #AgglomerativeClustering
240 agglomerative = AgglomerativeClustering(n_clusters=3).fit(X_trans)
241 # Fitting the input data and getting the cluster labels
242 labels_agg = agglomerative.labels_
243 print labels_agg
244
245 #AffinityPropagation
246 affinity = AffinityPropagation().fit(X_trans)
247 # Fitting the input data and getting the cluster labels
248 labels_aff = affinity.labels_
249 print labels_aff
250
251 '''
252 ---------- Question 3 ----------
253 '''
254
255 #set parameter for Gridsearch
256 cv = 10
257 n_features = X_train.shape[1]
258
```

```
259 #KNeighborsClassifier
260 n_neighbors_range = np.arange(1,20,1)
261 param_grid_n = dict(n_neighbors=n_neighbors_range) #set tuning parameter range
262
263 grid_neigh,best_estimator_neigh,score_neigh = gridcv(X_train,Y_train,X_vali,
        Y_vali,cv,KNeighborsClassifier(),param_grid_n) #get best estimator
264 plotclassifier(grid_neigh,n_neighbors_range,'n_neighbors') #plot score vs
        tuning parameter
265 print('KNeighborsClassifier: \nvalidation data score = {:f} \n'.format(
        score_neigh))
266
267 #RandomForestClassifier
268 n_estimators_range = np.arange(10,1000,100)
269 param_grid_f = dict(n_estimators=n_estimators_range)
270
271 grid_forest,best_estimator_forest,score_forest = gridcv(X_train,Y_train,X_vali
        ,Y_vali,cv,RandomForestClassifier(oob_score = True),param_grid_f)
272 plotclassifier(grid_forest,n_estimators_range,'n_estimators')
273 print('RandomForestClassifier: \nvalidation data score = {:f} \n'.format(
        score_forest))
274 importances(best_estimator_forest,n_features)
275
276 #GradientBoostingClassifier
277 learning_rate_range = np.arange(1e-2,1,1e-2)
278 param_grid_b = dict(learning_rate=learning_rate_range)
279
280 grid_boost, best_estimator_boost,score_boost = gridcv(X_train,Y_train,X_vali,
        Y_vali,cv,GradientBoostingClassifier(),param_grid_b)
281 plotclassifier(grid_boost,learning_rate_range,'learning_rate')
282 print('GradientBoostingClassifier: \nvalidation data score = {:f} \n'.format(
        score_boost))
283 importances(best_estimator_boost,n_features)
284
285 #LogisticRegressionCV
```

```
score_log = logisticregre(X_train,Y_train,X_vali,Y_vali,10)
print('LogisticRegressionCV: \nvalidation data score = {:f} \n'.format(
    score_log))

#Returns the mean accuracy on the given test data and labels.
score_test = best_estimator_boost.score(X_test,Y_test)
print score_test
```

Reference

[1] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[2] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[3] Lior Rokach and Oded Maimon. Clustering methods. In *Data mining and knowledge discovery handbook*, pages 321–352. Springer, 2005.

[4] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[5] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[6] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[7] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[8] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.