Arivan Thillaikumaran
CSE 514 HW2
425096

Problem 1

1.)

For **d** dimension, the Guassian distribution of a vector x is defined by:

$N(x|\mu,\Sigma) = \frac{1}{((2\Pi)^{d/2}\sqrt{\Sigma})} \exp(\frac{-1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)$, μ *is the mean,* Σ *is the covariance matrix*

The probability of given in a mixture model of k Gaussians is :

$p(x) = \sum_{j=1}^{k} \pi_j * N(x|\mu_j,\Sigma_j)$, w_j *is the weight of the jth Gaussian. All weights sum to* 1.

Given a set of data X=$\{x_1, x_{2.....}\ x_N\}$ drawn from the Gaussian Mixture Model, estimate the parameters that fit the data. We do this by maximizing the likelihood $p(X|\theta)$ of the data with regard to the model parameters. This is where the **Expectation-Maximization Algorithm** comes into play. The basic ideas of the E-M algorithm are:

- Introduce a hidden variable such that it can help simplify the maximization of the likelihood.
- At each iteration:
  - **Expectation step:** Estimate the distribution of the hidden variable given the data and the current value of the parameters. Denote the current (hidden variable that we introduced) as θ, and compute the probability that each point y_i comes from a θ_i:

    $P_i = \frac{\Pi_i * \phi(\theta_i) * y_i}{\sum_{k=1}^{N} \pi_k \phi_k y_i}$, $\phi(\theta)$ *represents Normal distribution*
  - **Maximization step:** Maximize the joint distribution of the data and the hidden variable. We find the means and variances of the distribution by weighting the points based on the likelihood they are from each model.

    $\mu_i = \frac{\sum_{i=1}^{N} P_i * y_i}{\sum_{i=1}^{N} P_i}$, $\sigma^2{}_i = \frac{\sum_{i=1}^{k} P_i (y_i - \mu_i)^2}{\sum_{i=1}^{N} P_i}$, $\pi_i = \frac{\sum_{i=1}^{N} P_i}{N}$

    Repeat these two steps until convergence.

Note: The initialization of the parameters lead to differing parameter convergences. The algorithm guarantees that you converge to a local extrema. However, to make sure that you are converging to a global extrema, you must vary your initialization of parameters to make sure that they are all converging to the global extrema, in case one of your initialization settings returns the local extrema rather than global. I saw this very occasionally, with around one out of every ten to 15 runs giving me a different set of parameter values.

2.) $\{\pi_1, \pi_2, \mu_1, \mu_2, \sigma_1, \sigma_2\}$ = {.56922, .43077, 15.5122, 43.34077, 8.277, 8.2309}

I observed that during some runs, the values for i=1 and 2 switch places, which makes sense as they can be a part of any of the distributions as long as the corresponding means and variances are respectively associated with the weights, which they are.

Problem 2

I dealt with the missing data in this problem by substituting the mean across the axis. I called Python's Sklearn.preprocecessing.Imputer, which is an imputation transformer for completing missing values, and allows me to specify that I want the mean to be substituted in. You then must transform the data that you have fitted based on the training data and target values.

Euclidean Distance as similarity measure:

K=2:

	0
cluster0 cluster1 Average	10.4726803834
cluster0 cluster1 Centroid	5.54957043992
cluster0 cluster1 Complete Link	21.185210665
cluster0 cluster1 Single Link	3.3390878506

K=3:

	0
cluster0 cluster1 Average	11.1624180432
cluster0 cluster1 Centroid	6.95781530301
cluster0 cluster1 Complete Link	19.860130551
cluster0 cluster1 Single Link	4.94805757601
cluster2 cluster0 Average	9.6110699485
cluster2 cluster0 Centroid	4.279728426
cluster2 cluster0 Complete Link	21.185210665
cluster2 cluster0 Single Link	4.15384144119
cluster2 cluster1 Average	10.0488935092
cluster2 cluster1 Centroid	5.22445504724
cluster2 cluster1 Complete Link	19.7436836941
cluster2 cluster1 Single Link	3.3390878506

K=4:

	0
cluster0 cluster1 Average	9.52603699616
cluster0 cluster1 Centroid	4.48916961385
cluster0 cluster1 Complete Link	21.185210665
cluster0 cluster1 Single Link	4.11560350797
cluster2 cluster0 Average	10.6703709275
cluster2 cluster0 Centroid	5.99312089072
cluster2 cluster0 Complete Link	18.9757656884
cluster2 cluster0 Single Link	5.1975021243
cluster2 cluster1 Average	9.79481250645
cluster2 cluster1 Centroid	4.97674473345
cluster2 cluster1 Complete Link	18.7687520683
cluster2 cluster1 Single Link	4.84069432626
cluster2 cluster3 Average	12.6040844689
cluster2 cluster3 Centroid	9.16275647213
cluster2 cluster3 Complete Link	19.860130551
cluster2 cluster3 Single Link	7.13900713126
cluster3 cluster0 Average	9.93693104201
cluster3 cluster0 Centroid	4.80403450133
cluster3 cluster0 Complete Link	19.7436836941
cluster3 cluster0 Single Link	4.75203577887
cluster3 cluster1 Average	9.72543130487
cluster3 cluster1 Centroid	5.13894292108
cluster3 cluster1 Complete Link	18.6737000014
cluster3 cluster1 Single Link	4.15384144119

Dot product as similarity measure:

k=2:

	0
cluster0 cluster1 Average	1.1406071972
cluster0 cluster1 Centroid	2
cluster0 cluster1 Complete Link	1.79048396588
cluster0 cluster1 Single Link	0.333608584079

k=3:

	0
cluster0 cluster1 Average	1.09027332728
cluster0 cluster1 Centroid	1.62230912646
cluster0 cluster1 Complete Link	1.72285185893
cluster0 cluster1 Single Link	0.336186851504
cluster2 cluster0 Average	1.187945676
cluster2 cluster0 Centroid	1.78763543361
cluster2 cluster0 Complete Link	1.79048396588
cluster2 cluster0 Single Link	0.38111384492
cluster2 cluster1 Average	0.99575396681
cluster2 cluster1 Centroid	0.99214538672
cluster2 cluster1 Complete Link	1.7243241677
cluster2 cluster1 Single Link	0.223269828555

K=4

	0
cluster0 cluster1 Average	1.09199964886
cluster0 cluster1 Centroid	1.90431153295
cluster0 cluster1 Complete Link	1.7243241677
cluster0 cluster1 Single Link	0.408519046801
cluster2 cluster0 Average	1.03677170896
cluster2 cluster0 Centroid	1.15976086595
cluster2 cluster0 Complete Link	1.64472270117
cluster2 cluster0 Single Link	0.382344351493
cluster2 cluster1 Average	0.98786314445
cluster2 cluster1 Centroid	0.978734081905
cluster2 cluster1 Complete Link	1.62414934656
cluster2 cluster1 Single Link	0.286919901416
cluster2 cluster3 Average	1.34730994347
cluster2 cluster3 Centroid	1.95694597139
cluster2 cluster3 Complete Link	1.79048396588
cluster2 cluster3 Single Link	0.757142664231
cluster3 cluster0 Average	0.952519412894
cluster3 cluster0 Centroid	0.796136563773
cluster3 cluster0 Complete Link	1.65603464805
cluster3 cluster0 Single Link	0.324757102061
cluster3 cluster1 Average	1.0255347498
cluster3 cluster1 Centroid	1.18408438828
cluster3 cluster1 Complete Link	1.68359480954
cluster3 cluster1 Single Link	0.29739822

Mean distances between clusters:

Euclidean	Average	Centroid	Complete Link	Single Link
k=2	**10.4727**	**5.5496**	**21.1852**	**3.3391**
k=3	**10.2741**	**5.63732**	**20.2629**	**4.147**
k=4	**10.3763**	**5.76**	**19.5345**	**5.02748**

Dot Product	Average	Centroid	Complete Link	Single Link
k=2	**1.14606**	**2**	**1.7905**	**.3336**
k=3	**1.0913**	**1.4674**	**1.7459**	**.31352**
k=4	**1.07363**	**1.3299**	**1.6872**	**.409**

As you increase k when using K-means, the results should get better. As such, from my data I can see that the dot product is much more effective than using the Euclidean distance because for each different cluster distance technique when using the dot product, the distance goes down. When using Euclidean for similarity measure, the complete link cluster distance technique yields the best results for increasing values of k. Of the four cluster distance measures, Centroid, and then Complete Link, which makes sense as the central vector that represents a central cluster that is created will approximate better than a minimum or maximum distance between the clusters.

Problem 3

1) **Using the linear kernel and SVM to select the top 200 genes:**

['GI_38327038-I', 'GI_37543009-S', 'GI_10835229-S', 'GI_45446741-S', 'GI_23238193-A', 'GI_21359861-S', 'GI_22035587-A', 'GI_18641372-S', 'GI_45598382-S', 'GI_34222316-S', 'GI_38016132-S', 'GI_19913395-I', 'GI_4507112-S', 'GI_31343498-A', 'GI_4502630-S', 'GI_33946283-I', 'GI_4557726-S', 'GI_4503874-S', 'GI_6912533-S', 'GI_31077210-A', 'GI_19923414-S', 'GI_5453596-S', 'GI_32313568-S', 'GI_45238848-S', 'GI_42661601-S', 'GI_5901937-S', 'GI_40806213-S', 'GI_13899296-S', 'GI_19557644-S', 'GI_27498358-S', 'GI_37548529-S', 'GI_4757909-S', 'GI_4557348-S', 'GI_19923110-S', 'GI_13899226-S', 'GI_8051578-S', 'GI_8922994-S', 'GI_13430873-S', 'GI_31341734-S', 'GI_18641378-S', 'GI_23957681-S', 'GI_38327635-S', 'GI_4757849-S', 'GI_13540508-S', 'GI_22035654-A', 'GI_27485046-S', 'GI_41393558-I', 'GI_27894372-A', 'GI_40254998-S', 'GI_8051576-A',

'GI_16753224-S', 'GI_45545436-I', 'GI_32171240-S', 'GI_34147581-S', 'GI_40217818-S', 'GI_42655923-S', 'GI_4504482-S', 'GI_21389570-S', 'GI_39777591-S', 'GI_4502660-S', 'GI_41872473-S', 'GI_20357531-S', 'GI_21389432-S', 'GI_38261964-A', 'GI_33946335-S', 'GI_41327755-S', 'GI_21361946-S', 'GI_22538441-S', 'GI_27437022-A', 'GI_21040361-A', 'GI_31343498-I', 'GI_42558249-I', 'GI_22095346-S', 'GI_5453687-S', 'GI_37594443-S', 'GI_34916047-S', 'GI_34452689-S', 'GI_23943885-S', 'GI_29029631-S', 'GI_8924224-S', 'GI_21361261-S', 'GI_34147673-S', 'GI_27894352-A', 'GI_21735557-A', 'GI_33598917-S', 'GI_17921992-I', 'GI_34222335-S', 'GI_42661835-S', 'GI_32483396-S', 'GI_27475984-S', 'GI_22749082-S', 'GI_37550155-S', 'GI_31543909-S', 'GI_18765747-A', 'GI_23957699-S', 'GI_39930526-S', 'GI_39841072-S', 'GI_31317298-A', 'GI_42656398-S', 'GI_4504794-S', 'GI_4503872-I', 'GI_22055338-S', 'GI_4503680-S', 'GI_41327153-S', 'GI_21389392-S', 'GI_17981703-S', 'GI_34916027-S', 'GI_13129143-S', 'GI_31542734-S', 'GI_42661719-S', 'GI_8923465-S', 'GI_24497521-I', 'GI_33636718-S', 'GI_17978499-A', 'GI_21312135-S', 'GI_4503562-S', 'GI_7262372-S', 'GI_11184225-S', 'GI_40254432-S', 'GI_4758697-S', 'GI_17975764-A', 'GI_7661869-S', 'GI_31982866-S', 'GI_37558299-S', 'GI_42476063-S', 'GI_21614543-S', 'GI_31377723-S', 'GI_12232388-S', 'GI_27894383-S', 'GI_31542848-S', 'GI_44771157-S', 'GI_40538795-S', 'GI_40255242-S', 'GI_33386702-S', 'GI_41281554-S', 'GI_40018630-A', 'GI_6912379-S', 'GI_42661634-S', 'GI_33504488-S', 'GI_13430859-S', 'GI_21361081-S', 'GI_7706468-S', 'GI_37551895-S', 'GI_14165257-S', 'GI_4755136-S', 'GI_21614536-I', 'GI_6031156-S', 'GI_4503320-S', 'GI_29893558-S', 'GI_4505564-S', 'GI_37552150-S', 'GI_33354256-S', 'GI_31621298-S', 'GI_7706116-S', 'GI_4506756-S', 'GI_45592953-S', 'GI_32171246-I', 'GI_13375845-S', 'GI_14042952-S', 'GI_14670363-I', 'GI_15718672-S', 'GI_39995079-S', 'GI_20357536-I', 'GI_30150520-S', 'GI_4557504-S', 'GI_38158004-A', 'GI_31542586-S', 'GI_33300650-S', 'GI_8922595-S', 'GI_4502854-S', 'GI_37541634-S', 'GI_31747573-S', 'GI_42661344-S', 'GI_31712021-S', 'GI_34147526-S', 'GI_25092724-S', 'GI_4580421-A', 'GI_28195383-S', 'GI_24432076-S', 'GI_38348371-S', 'GI_28557782-I', 'GI_7657503-S', 'GI_41222114-S', 'GI_4505050-S', 'GI_38570131-S', 'GI_40549416-S', 'GI_34485729-S', 'GI_10346134-S', 'GI_4503174-S', 'GI_42403584-A', 'GI_37547347-S', 'GI_25952086-S', 'GI_13375757-S', 'GI_22748812-S', 'GI_34222221-S', 'GI_14110425-S', 'GI_42476060-S', 'GI_16507197-S', 'GI_22208956-S', 'GI_40804743-S']

2) Using the top 100 genes returned by Random Forest Classifier, applying the quadratic kernel to find top 200 features

['GI_22035587-A', 'GI_10835229-S', 'GI_38261964-A^2', 'GI_40217818-S', 'GI_10835229-S^2', 'GI_38016132-S', 'GI_4503874-S', 'GI_22749082-S^2', 'GI_33946283-I^2', 'GI_21359861-S GI_27894352-A', 'GI_40254998-S', 'GI_18641372-S^2', 'GI_41327755-S', 'GI_21389432-S', 'GI_4502660-S^2', 'GI_19557644-S GI_38261964-A', 'GI_39930526-S GI_4757909-S', 'GI_41393558-I^2', 'GI_10835229-S GI_19913395-I', 'GI_10835229-S GI_4502660-S', 'GI_37543009-S', 'GI_22749082-S GI_4503874-S', 'GI_45545436-I', 'GI_4504794-S^2', 'GI_18641372-S GI_38261964-A', 'GI_39930526-S GI_5901937-S', 'GI_21389570-S', 'GI_19923110-S GI_34916047-S', 'GI_16753224-S GI_5453596-S', 'GI_10835229-S GI_4502630-S', 'GI_33598917-S GI_4757909-S', 'GI_31077210-A', 'GI_22749082-S

GI_45238848-S', 'GI_27437022-A GI_41393558-I', 'GI_22035654-A GI_27437022-A',
'GI_10835229-S GI_27437022-A', 'GI_45446741-S^2', 'GI_33946283-I GI_5901937-S',
'GI_8922994-S^2', 'GI_22035654-A GI_37543009-S', 'GI_22538441-S', 'GI_19923110-S^2',
'GI_19913395-I', 'GI_10835229-S GI_5453596-S', 'GI_32171240-S', 'GI_19923110-S
GI_45446741-S', 'GI_19923414-S^2', 'GI_10835229-S GI_22749082-S', 'GI_22035654-A',
'GI_13430873-S GI_22035654-A', 'GI_34222335-S', 'GI_20357531-S GI_37550155-S',
'GI_4502630-S', 'GI_32171240-S GI_5453596-S', 'GI_41393558-I GI_5453596-S',
'GI_37543009-S^2', 'GI_27475984-S GI_41393558-I', 'GI_21389432-S GI_38261964-A',
'GI_33598917-S GI_8924224-S', 'GI_5453596-S^2', 'GI_19557644-S GI_37550155-S',
'GI_21359861-S GI_41393558-I', 'GI_18641378-S GI_45598382-S', 'GI_10835229-S
GI_27475984-S', 'GI_27894352-A GI_40254998-S', 'GI_45545436-I^2', 'GI_4757909-S^2',
'GI_42656398-S GI_5901937-S', 'GI_39930526-S^2', 'GI_19923414-S GI_5901937-S',
'GI_31343498-I GI_4504482-S', 'GI_38261964-A GI_5901937-S', 'GI_27894352-A
GI_4502630-S', 'GI_34222316-S GI_39930526-S', 'GI_4503874-S^2', 'GI_45598382-S',
'GI_40806213-S GI_41327755-S', 'GI_19913395-I GI_38261964-A', 'GI_37543009-S
GI_4757909-S', 'GI_6912533-S^2', 'GI_22749082-S GI_41393558-I', 'GI_10835229-S
GI_38261964-A', 'GI_33598917-S GI_5901937-S', 'GI_22035587-A GI_4557726-S',
'GI_40254998-S GI_4503874-S', 'GI_22035654-A GI_4507112-S', 'GI_37550155-S
GI_4502660-S', 'GI_4757909-S GI_5453596-S', 'GI_22035587-A GI_41872473-S',
'GI_27437022-A GI_40806213-S', 'GI_21359861-S GI_45446741-S', 'GI_19923110-S
GI_38327635-S', 'GI_5901937-S', 'GI_21040361-A^2', 'GI_4502630-S GI_4502660-S',
'GI_34916047-S GI_5453596-S', 'GI_21359861-S GI_22035654-A', 'GI_10835229-S
GI_21389570-S', 'GI_5901937-S GI_6912533-S', 'GI_27437022-A GI_27475984-S',
'GI_13540508-S GI_5453596-S', 'GI_19923110-S GI_22035654-A', 'GI_21361261-S
GI_38261964-A', 'GI_42558249-I', 'GI_38261964-A GI_40806213-S', 'GI_10835229-S
GI_19923414-S', 'GI_20357531-S GI_33598917-S', 'GI_37550155-S GI_5453596-S',
'GI_17921992-I GI_37550155-S', 'GI_18765747-A GI_34916047-S', 'GI_40806213-S
GI_45238848-S', 'GI_23957681-S^2', 'GI_10835229-S GI_4504482-S', 'GI_18641372-S
GI_5901937-S', 'GI_22749082-S', 'GI_21040361-A GI_5901937-S', 'GI_22035654-A
GI_27475984-S', 'GI_21359861-S GI_4757849-S', 'GI_13430873-S GI_37550155-S',
'GI_13899226-S^2', 'GI_31317298-A^2', 'GI_34452689-S GI_8922994-S', 'GI_19913395-I^2',
'GI_20357531-S GI_4502660-S', 'GI_10835229-S GI_34916047-S', 'GI_45545436-I
GI_4757909-S', 'GI_34452689-S GI_38016132-S', 'GI_33598917-S GI_4507112-S',
'GI_19557644-S GI_23957681-S', 'GI_4504482-S GI_4757849-S', 'GI_23957681-S
GI_5453596-S', 'GI_23238193-A GI_4757849-S', 'GI_10835229-S GI_6912533-S',
'GI_31317298-A GI_41393558-I', 'GI_21361261-S GI_34222316-S', 'GI_19923110-S
GI_4503874-S', 'GI_23238193-A GI_39777591-S', 'GI_21389570-S GI_33598917-S',
'GI_40806213-S GI_8051578-S', 'GI_18641378-S GI_34452689-S', 'GI_37594443-S
GI_5901937-S', 'GI_19923110-S', 'GI_16753224-S^2', 'GI_13540508-S GI_31343498-I',
'GI_34147673-S GI_37543009-S', 'GI_21359861-S GI_38327635-S', 'GI_18765747-A
GI_21359861-S', 'GI_19923414-S GI_4502630-S', 'GI_10835229-S GI_13430873-S',
'GI_37550155-S GI_40806213-S', 'GI_4503874-S GI_4557726-S', 'GI_4557348-S
GI_5901937-S', 'GI_33946283-I GI_4502630-S', 'GI_33946283-I GI_38327038-I',

'GI_38327038-I', 'GI_10835229-S GI_33598917-S', 'GI_18641372-S GI_32171240-S', 'GI_16753224-S', 'GI_21361946-S GI_4503874-S', 'GI_22749082-S GI_34222335-S', 'GI_40806213-S GI_4503874-S', 'GI_22749082-S GI_23943885-S', 'GI_18641378-S GI_22035654-A', 'GI_34452689-S GI_4503874-S', 'GI_18765747-A', 'GI_40254998-S GI_6912533-S', 'GI_27894372-A GI_33946283-I', 'GI_37594443-S^2', 'GI_19923414-S GI_42655923-S', 'GI_33946335-S', 'GI_38016132-S GI_4502630-S', 'GI_34916047-S GI_6912533-S', 'GI_31343498-A', 'GI_19923414-S GI_39930526-S', 'GI_17921992-I GI_19923414-S', 'GI_27894372-A GI_4502660-S', 'GI_23238193-A', 'GI_23957681-S GI_27894352-A', 'GI_27498358-S GI_5901937-S', 'GI_16753224-S GI_22035587-A', 'GI_19557644-S GI_32483396-S', 'GI_38261964-A GI_5453687-S', 'GI_5901937-S GI_8051576-A', 'GI_39841072-S', 'GI_22035654-A GI_42661601-S', 'GI_4504794-S GI_5901937-S', 'GI_27475984-S GI_38261964-A', 'GI_4502660-S GI_4504794-S', 'GI_41393558-I GI_5453687-S', 'GI_17921992-I', 'GI_32483396-S^2', 'GI_34147581-S GI_4757909-S', 'GI_13899226-S GI_5453596-S', 'GI_19923110-S GI_40254998-S', 'GI_19923110-S GI_20357531-S', 'GI_38327635-S', 'GI_37543009-S GI_4557726-S', 'GI_18641372-S GI_33946335-S', 'GI_34222335-S GI_5901937-S', 'GI_32313568-S GI_8922994-S']

You can see that polynomial features were generated. This means that some interactions between specific genes are important.

Code for Problem 1:

```
import numpy as np
from sklearn import mixture

D = [[35], [41], [21], [20], [17], [55], [12], [33], [15], [18], [4],[51], [17], [46]]

gmm = mixture.GaussianMixture(2, init_params="random", n_init = 100).fit(D)
print gmm.means_
print gmm.covariances_
print gmm.weights_
```

Code for Problem 2 and 3

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import preprocessing
from sklearn import cluster
from sklearn import svm
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist
from sklearn.metrics.pairwise import cosine_similarity
import itertools
import numpy as np
import random

import matplotlib.pyplot as plt
import csv as csv
from collections import defaultdict
import pandas as pd


#PROBLEM 2

dataset = pd.read_csv('allPatients.csv',low_memory=False)
x = dataset.drop('Classes',axis=1)
x.replace(to_replace='?',value='NaN',inplace=True)
features = x.columns
imp = preprocessing.Imputer(missing_values='NaN', strategy='mean', axis=0)
x = imp.fit_transform(x)
x_imputed = pd.DataFrame(x,columns = features)
#features = x.columns
y = dataset['Classes']

clf = RandomForestClassifier(n_estimators=300, criterion = 'entropy', max_features=100)
clf.fit(x_imputed,y)



gene_freq = np.zeros(8560)
gene_freq = {}
for i in range(8560):
        gene_freq[i] = 0
for tree in clf.estimators_:
        i = 0
        for gene in tree.feature_importances_:
                if gene>0:
```

```
                    gene_freq[i] = gene_freq[i]+1
            i = i+1

sortedgenes = sorted(gene_freq, key=gene_freq.get, reverse=True)
featuresp2 = sorted(list(clf.feature_importances_), reverse=True)
sortedtwohundred = sortedgenes[0:200]

#get top 200
top200 = []

column_name = list(x_imputed.columns.values)

for i in sortedtwohundred:
        top200.append(column_name[i])

#new dataset using chosen features
filtered = pd.read_csv('allPatients.csv',low_memory=False, usecols =top200)
filtered.replace(to_replace='?', value='NaN', inplace=True)

featuresp2_ = filtered.columns
imp1 = preprocessing.Imputer(missing_values='NaN', strategy='mean', axis=0)
relevant_features = imp1.fit_transform(filtered)
relevant_genes = pd.DataFrame(relevant_features, columns=featuresp2_)

def getClusters(labels,data,k):
   d = {}
   for i in range(0,k):
      d["cluster"+str(i)] =  data.ix[np.nonzero(labels==i)]
   return d

def getDistance(clusters,distance):
   distDict = {}
   for cluster1 in itertools.combinations(clusters, 2):
      dist = cdist(clusters[cluster1[0]],clusters[cluster1[1]], distance)
      n,m = dist.shape
      distDict[cluster1[0]+" " +cluster1[1] + " Single Link"] = np.nanmin(dist)
      distDict[cluster1[0]+" " +cluster1[1] + " Complete Link"] = np.nanmax(dist)
      dist = np.nan_to_num(dist)
      distDict[cluster1[0]+" " + cluster1[1] + " Average"] = sum(sum(dist))/(n*m)
      distDict[cluster1[0] + " "+ cluster1[1] + " Centroid"] =
np.amax(cdist(pd.DataFrame(clusters[cluster1[0]].mean()).transpose(),pd.DataFrame(clusters[cl
uster1[1]].mean()).transpose(),distance))
   return distDict
```

```
estimateEuclidean = {'euclidean_k2': KMeans(n_clusters=2).fit(relevant_genes),
            'euclidean_k3': KMeans(n_clusters=3).fit(relevant_genes),
            'euclidean_k4': KMeans(n_clusters=4).fit(relevant_genes)}

def new_euclidean_distance(X, Y=None, Y_norm_squared=None, squared=False):
   return cosine_similarity(X,Y)

#monkey patch for runtime attribute cosine similary measurements
KMeans.euclidean_distances = new_euclidean_distance

estimateDot = {'dot_k2': KMeans(n_clusters=2).fit(relevant_genes),
            'dot_k3': KMeans(n_clusters=3).fit(relevant_genes),
            'dot_k4': KMeans(n_clusters=4).fit(relevant_genes)}




for cluster in estimateEuclidean:
   k = len(np.unique(estimateEuclidean[cluster].labels_))
   groups =
getClusters(estimateEuclidean[cluster].labels_,relevant_genes,len(np.unique(estimateEuclidean
[cluster].labels_)))
   dist = getDistance(groups,'euclidean')
   pd.DataFrame([dist]).transpose().to_csv('ResultsEuclideanK({0}).csv'.format(k))
for cluster in estimateDot:
   k = len(np.unique(estimateDot[cluster].labels_))
   groups = getClusters(estimateDot[cluster].labels_,relevant_genes,k)
   dist = getDistance(groups,'cosine')
   pd.DataFrame([dist]).transpose().to_csv('ResultsDotK({0}).csv'.format(k))
```

#END PROBLEM 2

#BEGIN PROBLEM 3

#PROBLEM 3 PART 1

```
clf = svm.SVC(kernel='linear', C=1.0)
clf.fit(x_imputed, y)

weights = clf.coef_[0]
```

```
coefs = {}
for i in range(8560):
	coefs[i] = 0
index = 0
for coef in clf.coef_:
	for wrap in coef:
		coefs[index] = abs(wrap)
		index = index+1


sortedgenesp3 = sorted(coefs, key=coefs.get, reverse=True)
sorted200p3 = sortedgenesp3[0:200]

top200p3 = []
columns = list(x_imputed.columns.values)
for coef in sorted200p3:
	top200p3.append(columns[coef])

print top200p3
```

#Problem 3 Part 2

```
#get best 100 features
sortedonehundred = sortedgenesp3[0:100]

top100 = []

for i in sortedonehundred:
	top100.append(column_name[i])

filteredp3 = pd.read_csv('allPatients.csv', low_memory=False, usecols = top100)
filteredp3.replace(to_replace='?', value='NaN', inplace=True)

features_p3_100 = filteredp3.columns
imp2 = preprocessing.Imputer(missing_values='NaN', strategy='mean', axis=0)
relevant_featuresp3 = imp2.fit_transform(filteredp3)
relevant_genesp3 = pd.DataFrame(relevant_featuresp3, columns=features_p3_100)

#process and transform via quadratic kernel
poly = preprocessing.PolynomialFeatures(2)
poly_ = poly.fit_transform(relevant_genesp3)
```

```
featuresp3 = poly.get_feature_names(filteredp3.columns)

transformed = pd.DataFrame(poly_, columns = featuresp3)

#applying linear kernel to our transformation
clf1 = svm.SVC(kernel="linear")
clf1_p3 = clf1.fit(transformed,y)

poly_coefs = {}
for n in range(5151):
        poly_coefs[n] = 0
poly_index=0

for coeflist in clf1_p3.coef_:
        for coef in coeflist:
                poly_coefs[poly_index] = abs(coef)
                poly_index=poly_index+1

sorted_poly_coefs=sorted(poly_coefs, key=poly_coefs.get, reverse=True)
top200poly = sorted_poly_coefs[0:200]


top200p3_ = []

for i in top200poly:
        top200p3_.append(featuresp3[i])

print top200p3_
```