Universitetet i Tromsø

Fys-2021 - Machine Learning

Take home exam 1

Candidate number: 6

October 5, 2018

1 Problem 1

$$r\left(P_i\right) = \sum_{P_j \in B_{P_i}} \frac{r\left(P_j\right)}{\left|P_j\right|} \tag{1}$$

1.1 a)

In eq(1) the $r(P_i)$ is the rank of a given page, the interval $P_j \epsilon B_{p_i}$ is all the pages pointing to the page P_i. $r(P_j)$ are the number of out links from the P_j pages. The $|P_j|$ is the number of out links from the other pages. This will give a sensible ranking because the more 'important' pages will carry more weight when it links to another page, see example below.[1]

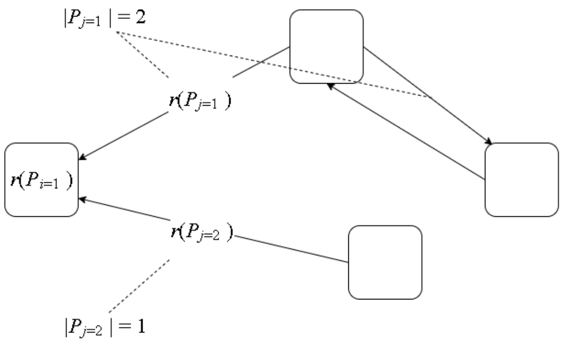


Figure 1: Here is an example of a network of pages being ranked.

1.2 b)

We turn the matrix in equation(1) into vectors of probabilities (each row in the matrix sum to one). Turning the matrix into probabilities is done by summing up the rows and divide every element in each row by the sum (example of this is shown in equation(2) and (3) below). If there are a row which are all zero we add an equal probability into each row to make it stochastic. If we have matrix we can find the steady state or the eigenvector of the system. This is a state in which there is no change after the transformation. By using the power method we iterate by multiply the matrix with an initial vector $\mathbf{x}_0$ So we have $\mathbf{A}\mathbf{x}_0$, if we have two eigenspaces, the vector $\mathbf{x}_0$ will be scaled by two eigenvalues, but the closest eigenspace will have a bigger scalar

[1]Langville & Mayer p.144-145

and will pull $\mathbf{x}_0$ towards the eigenspace of λ_1. When the vector arrives at the eigenspace after n iterations the vector will be stable and not change. [2]

The matrix represents the probability(number of links to a page which are normalized) for each page, also called the transition matrix of the Markov chain.(This must be stochastic which will be discussed later.)

1.3 c)

When having the raw transition matrix there might be states that are not reachable because it might be a dangling node which can act as a "trap", see figure(2) below.

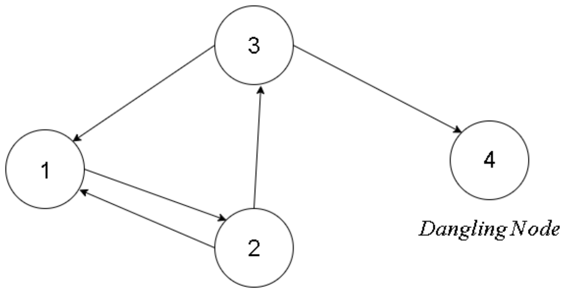


Figure 2: Example of dangling node.

In this case if we ever enter the dangling node there is no way to get out of it because it doesn't link to any other in the system. Here we need to alter the transition matrix such that we get an irreducible Markov chain, this could be done by making a new transition matrix and do the following, under we have the raw transition matrix for figure(2).[3]

$$\mathbf{P} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 1/2 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{2}$$

Here we see that the last row is all zero, this is because there is no back link from the dangling node, so there are zero chance to get out of it. We now have a non stochastic matrix, to alter this we can add a even chance to get out to any other node, this lessens the 'voting power' of the node, but now it is stochastic;

[2]Elementary Linear Algebra, Howard, Anton & Rorres, Chris,p.489-490

[3]https://pi.math.cornell.edu/ mec/Winter2009/RalucaRemus/Lecture3/lecture3.html

$$\overline{\mathbf{P}} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 1/2 & 0 & 0 & 1/2 \\ 1/4 & 1/4 & 1/4 & 1/4 \end{pmatrix} \quad (3)$$

Now we have a stochastic and irreducible matrix. It could be the case that it was reducible if we had a system that was closed off from the rest of the system. Here we could have the possibility of becoming trapped. Mathematically this means that there exists a permutation matrix such that[4],

$$\mathbf{Q^T B Q} = \begin{pmatrix} X & Y \\ 0 & Z \end{pmatrix} \quad (4)$$

Where both X and Z are squares.
This also means that it doesn't exist one unique steady state. Using equation 5 below we can force the matrix to become irreducible.[5]

$$\bar{\bar{P}} = \alpha\bar{P} + (1-\alpha)E \quad (5)$$

Here α is some scalar between 0 and 1 and we can call this a tuning/damping parameter, $\mathbf{E} = \mathbf{e}\mathbf{e}^T/n$ with $e_j = 1$.

1.4 d)

The reason why this is different is because now one node can point to another node more then one time. When used in hyperlinks, another node/website can only link once. An example is given in figure(3) below.

[4] Langville & Mayer p.151-152
[5] Langville & Mayer p.141

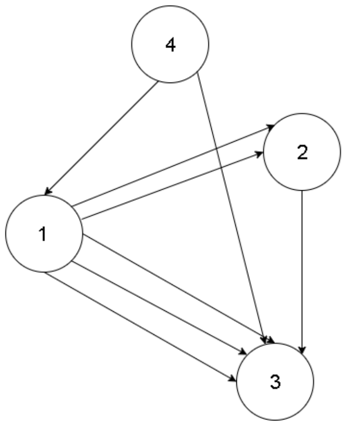
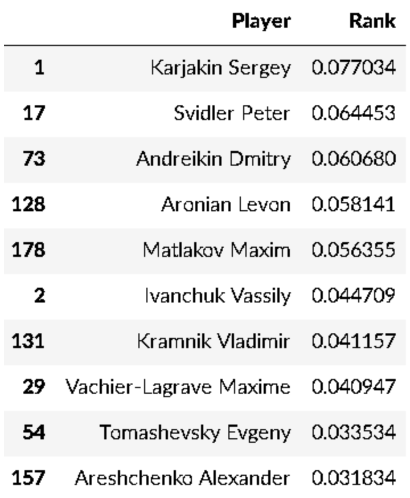


Figure 3: In the chess player case, node/player1 have lost against node/player3 three times.

1.5 e)

In the figure below the ranks given from the power method are shown.

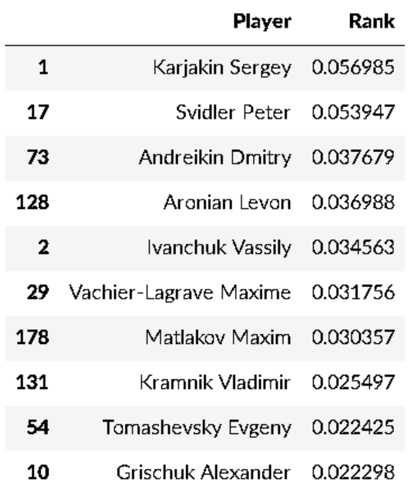


	Player	Rank
1	Karjakin Sergey	0.077034
17	Svidler Peter	0.064453
73	Andreikin Dmitry	0.060680
128	Aronian Levon	0.058141
178	Matlakov Maxim	0.056355
2	Ivanchuk Vassily	0.044709
131	Kramnik Vladimir	0.041157
29	Vachier-Lagrave Maxime	0.040947
54	Tomashevsky Evgeny	0.033534
157	Areshchenko Alexander	0.031834

Figure 4: The top 10 given from the ranking algorithm

1.6 f)

Here we adjust the transition matrix according to equation 5 with $\alpha = 0.85$. The table is given in figure(5) below.



	Player	Rank
1	Karjakin Sergey	0.056985
17	Svidler Peter	0.053947
73	Andreikin Dmitry	0.037679
128	Aronian Levon	0.036988
2	Ivanchuk Vassily	0.034563
29	Vachier-Lagrave Maxime	0.031756
178	Matlakov Maxim	0.030357
131	Kramnik Vladimir	0.025497
54	Tomashevsky Evgeny	0.022425
10	Grischuk Alexander	0.022298

Figure 5: The top 10 given from the ranking algorithm with equation(5).

Here we see the ranks are more spread out, and even a player have been switched out. The reason for this is because the Google formula/matrix[6] fills every row with some probability. This means that the players who have not played against someone doesn't still have a chance to lose to other players. Lets say before we change the transition matrix it looks like the equation below, [7],

$$\mathbf{P} = \left(\begin{array}{cc|cc} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{array}\right) \tag{6}$$

Here all rows sum to one so it is stochastic, but notice the "block of zero" in the upper and lower corner. This means that some nodes have not yet played with other players. In theory this means that not all players have "proven" their loss yet to all the players. Mathematically this means we have two eigenvectors with eigenvalue of one, and what we want is only one unique steady vector. When we then apply the Google equation (equation(5)) with tuning factor of 0.85 we get this,

[6] Langville & Mayer p.147

[7] https://pi.math.cornell.edu/ mec/Winter2009/RalucaRemus/Lecture3/lecture3.html

$$\mathbf{P} = \begin{pmatrix} 0.0375 & 0.8875 & 0.0375 & 0.0375 \\ 0.8875 & 0.0375 & 0.0375 & 0.0375 \\ 0.0375 & 0.0375 & 0.0375 & 0.8875 \\ 0.0375 & 0.0375 & 0.8875 & 0.0375 \end{pmatrix} \tag{7}$$

Here the transition matrix is more fair to those who have not yet played with all the other players, this lowers the overall ranking of all, and if someone had only a few wins then its not that significant unless it was against someone with high rank. This is why some players have lost their place in the top 10.

1.7 g)

For the regression model we want to find some function on the form,

$$r = f(x) + \epsilon \tag{8}$$

We want to approximate this function which of course will be perfect in nature, but we cant find this perfectly without having unlimited data points. Therefore we need to find its estimators. If we assume that our error ϵ is Gaussian with zero mean and constant variance $\epsilon \sim N(0, \sigma^2)$ and we now place our estimating function $g(\cdot)$ in place of the unknown function $f(\cdot)$
We then have the function

$$p(r|x) = N \sim (g(x|\theta), \sigma^2) \tag{9}$$

To get the parameters we need to do the most likely ones with the use of maximum likelihood. The log likelihood of the normal distribution is,

$$\log \mathcal{L}(\theta|X) = \log \prod_{t=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} e^{\left(-\frac{[r^t - g(x^t|\theta)]^2}{2\sigma^2}\right)} \tag{10}$$

Here we can already see the term in the exponential, if we minimize this we will maximize the overall probability. In the case of linear regression we have the model[8],

$$g(x^t|\theta) = \beta_1 x^t + \beta_0 \tag{11}$$

[8]The more general way to find the linear regression is by minimizing the squared error by first placing a random line through the data and then minimizing the squared distance to each point. In this example we use a probability approach and maximize the probability instead.

where β_1 is the parameter which in the case of chess players is how much higher rank you have based on your Elo. If we then minimize the exponent in equation(10) with equation(11) we get

$$\sum_t r^t = N\beta_0 + \beta_1 \sum_t x^t$$
$$\sum_t r^t x^t = \beta_0 \sum_t x^t + \beta_1 \sum_t (x^t)^2$$

When used in the program to solve we put it in vector form,

$$\mathbf{A} = \begin{pmatrix} N & \sum_t x^t \\ \sum_t x^2 & \sum_t (x^t)^2 \end{pmatrix} \tag{12}$$

, this could be done in a more general way by doing the following,[9]

$$\mathbf{A} = \begin{pmatrix} 1 & x^1 \\ 1 & x^2 \\ 1 & x^3 \\ . & . \\ . & . \\ . & . \end{pmatrix} \tag{13}$$

And then solving the equation;

$$\mathbf{w} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{r} \tag{14}$$

where r is the data to be predicted. This is what is used in the code, and the estimators obtained is,

$$\hat{\beta}_1 = 27.6$$
$$\hat{\beta}_0 = 1382.8$$

The intercept($\hat{\beta}_0$) tells how much the predicted Elo is if the rank is zero, and the slope coefficient ($\hat{\beta}_1$) tells how much the Elo is the predicted increase according to rank.

1.8 h)

Plotting the Elo data on y-axis and ranking on the x-axis gave the plot in figure(6) below.

The linear model seems to have a reasonable fit, although there are some fair amount of error terms.

[9]Introduction to machine learning, Alpaydin, Ethem p.77-80

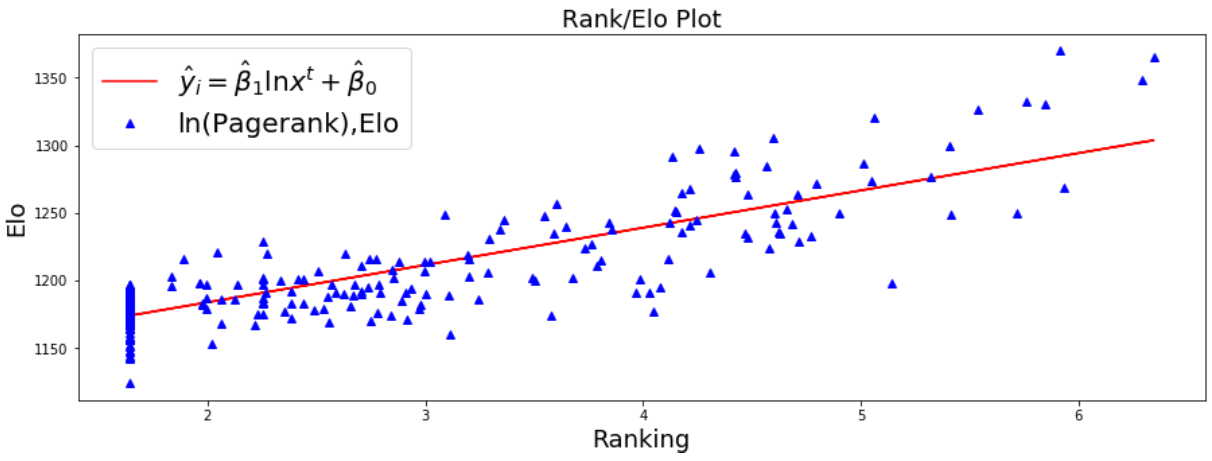


Figure 6: Linear regression model of the Google ranking with Elo data.

1.9 i)

The coefficient of determination for the Google steady state vector regression was calculated to $R^2 = 0.72$ this means that the correlation between the predicted and observed is about 72% accounted for. So this model is not the best, but it is reasonable. Al tough the highest R^2 is 1 its not necessarily the best because we could have over fitted our model. [10]

2 Problem 2

2.1 a)

We have the gamma distribution function given by,

$$P(x|C_0) = \frac{1}{\beta^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{-x/\beta} \tag{15}$$

If we now take the likelihood by taking the product of the probability function and assume independence,

$$\mathcal{L}P(x|C_0) = \prod_{i=1}^{N} P(x|C_0) \tag{16}$$

Expanding equation(16) and simplify we get,

[10] https://en.wikipedia.org/wiki/Coefficient_of_determination

$$\mathcal{L}P(x|C_0) = \left(\frac{1}{\beta^\alpha \Gamma(\alpha)}\right)^N x^{N(\alpha-1)} e^{-\sum_i x_i/\beta}$$

Now we take the log to get the exponents down to simplify,

$$\log \mathcal{L}P(x|C_0) = N\ln\left(\frac{1}{\beta^\alpha \Gamma(\alpha)}\right) + N(\alpha-1)\ln(x) - \sum_i \frac{x_i}{\beta}$$

Now we want maximize the probability for the given parameter therefore we take the derivative with respect to β and set it equal to zero. The reason why we can log the function is because log is a monotonic so values of the minimums and maximums will be at the same on the x axis.

$$\frac{\partial}{\partial\beta} \log \mathcal{L}P(x|C_0) = \frac{\partial}{\partial\beta}(-N\ln(\beta^\alpha\Gamma(\alpha)) - \sum_i \frac{x_i}{\beta}) = 0$$

$$(-N\frac{1}{\beta^\alpha\Gamma(\alpha)}(\alpha\Gamma(\alpha)\beta^{\alpha-1}) + \sum_i \frac{x_i}{\beta^2}) = 0$$

Here used the chain rule on the logarithmic part. Simplifying;

$$\sum_i \frac{x_i}{\beta^2} = N\frac{1}{\beta^\alpha}(\alpha\beta^{\alpha-1})$$

taking the inverse on both sides;

$$\frac{\beta^2}{\sum_i x_i} = \frac{\beta^\alpha}{N(\alpha\beta^{\alpha-1})}$$

Removing the α on both β and dividing the last β and multiplying by the sum of x

$$\hat{\beta} = \frac{\sum_i x_i}{N\alpha} \tag{17}$$

Which is our estimator for parameter in the gamma distribution.

For the normal distribution we have,

$$P(x|C_1) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{18}$$

Here we take the likelihood and logarithm and get,

$$\log \mathcal{L}P(x|C_1) = \frac{N}{2}\log\left(\frac{1}{2\pi\sigma^2}\right) - \sum_i \frac{(x_i-\mu)^2}{2\sigma^2}$$

taking the derivative with respect to μ gives,

$$\sum_i \frac{(x_i - \mu)}{\sigma^2} = 0$$

Here the minus got removed because of chain rule and the fact that μ had a minus sign, the first term was zero because there were no μ dependence. Solving for μ,

$$\mu N = \sum_i x_i$$

$$\mu N = \sum_i x_i$$

and finally,

$$\hat{\mu} = \frac{\sum_i x_i}{N} \tag{19}$$

Which is the estimator for μ in the normal distribution.

For the variance we do the same, taking the likelihood and log;

$$\log \mathcal{L}P(x|C_1) = \frac{N}{2} \log\left(\frac{1}{2\pi\sigma^2}\right) - \sum_i \frac{(x_i - \mu)^2}{2\sigma^2}$$

Now we take the derivative with respect to σ instead and set it equal to zero,

$$\frac{\partial}{\partial\sigma} \log \mathcal{L}P(x|C_1) = -\frac{N}{\sigma} + \sum_i \frac{(x_i - \mu)^2}{\sigma^3} = 0$$

here we used the fact that $log\frac{1}{x} = log(1) - log(x)$ and $log(1) = 0$ and the chain rule on the logarithmic term.

Rearranging the terms we get,

$$\sum_i \frac{(x_i - \mu)^2}{\sigma^3} = \frac{N}{\sigma}$$

$$\hat{\sigma}^2 = \sum_i \frac{(x_i - \mu)^2}{N}$$

Now we have the estimator for μ so we use that,

$$\hat{\sigma}^2 = \sum_i \frac{(x_i - \hat{\mu})^2}{N}$$

2.2 b)

The histogram and distributions have been plotted in figure(7) below. The distribution seem to be very reasonable because the peaks of both the data and distributions seem to fit and drop of at the same time.

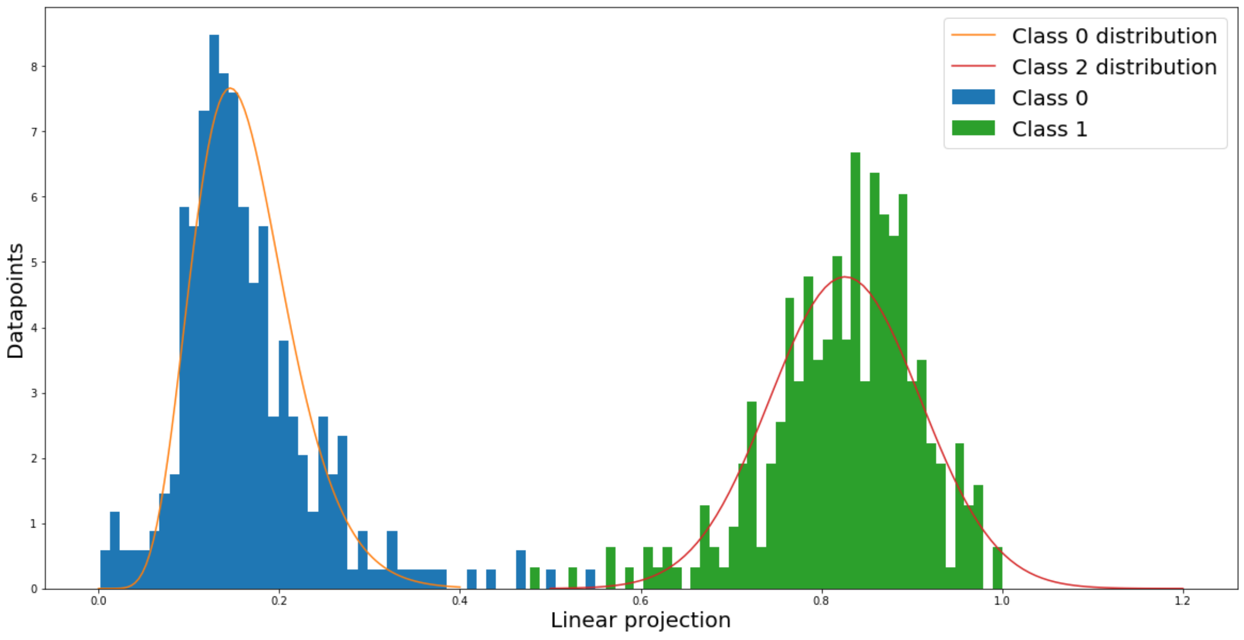


Figure 7: Histogram of both classes and their distributions

Their priors have been calculated by taking the total of one class and divide by the total amount of data (population data). This is done for both classes. Their prior probability is,

$$P(C_0) = 0.51$$
$$P(C_1) = 0.49$$

The prior probability is the probability we know like if there is a basket of 10 apples and nine of them are green, then i know the probability of picking a green apple is 9/10. This probability is usually used as an extra guide for the overall probability and makes the model more accurate. This could also help tune our model by knowing knew information over time.

2.3 c)

The confusion matrix is given in figure(8) below.

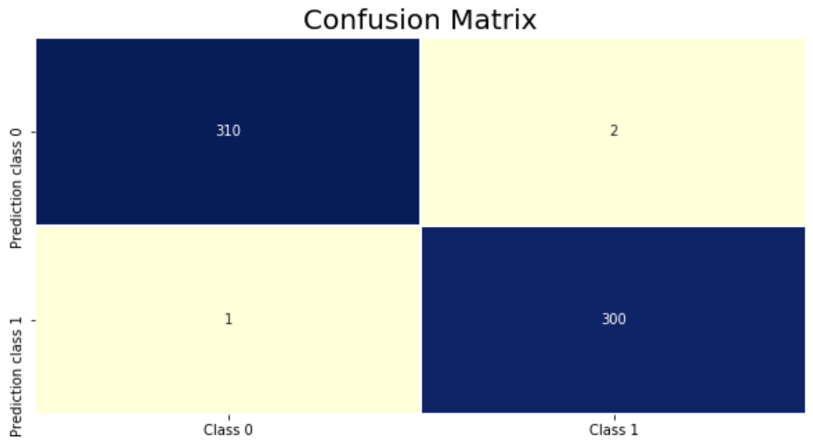


Figure 8: Confusion matrix from Bayes classifier

Here we have 2 false positives and 1 false negative, and the rest have been classified the right way. We continue to find the accuracy,
Accuracy of the prediction: 0.995 or 99.5% accurate.

This is found by taking the sum of the true positives and negatives divided by the total population. The reason why not everything is correctly classified is that in some extreme cases the data is crossing the threshold of each distribution making it hard to predict the real class.

In this task we used the Bayes Theorem,

$$P(C|x) = \frac{P(x|C)P(C)}{P(x)} \tag{20}$$

When using it as a discriminant function we can remove the equal terms (P(x)) so our discriminant will be,

$$g(C_0|x) = P(x|C_0)P(C_0)$$
$$g(C_1|x) = P(x|C_1)P(C_1)$$

where $P(C_i)$ is the prior for the class C_i. Here we then choose the class with highest probability and assign it to a list as either 0 or 1 if its classified as class 1 or class 2 respectively.

2.4 d)

Here the secret message is ”nevergonnagiveymuup”. I am guessing it should be ”nevergonnagiveyouup” There reason why there is one miss classification is that the projections are from real life examples and some have been written so badly that its hard to classify (the projection value is closer to the distribution of the other class). This can be seen in the histogram.

References

Ethem Alpaydin. *Introduction to machine learning.* The MIT Press, third edition, 2014.

Howard Anton and Chris Rorres. *Elementary linear algebra with supplemental applications.* Wiley, 11 edition, 2015.

Cornell University. PageRank Algorithm - The Mathematics of Google Search, 2009. URL `https://pi.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture3/lecture3.html`. [Online; accessed 21.09.2018].

Amy N. Langville and Carl D. Meyer. A survey of eigenvector methods for web information retrieval. *SIAM Review*, 47(1):135161, 2005. doi: 10.1137/s0036144503424786.

Weisstein, Eric W. Transition matrix — MathWorld–A Wolfram Web Resource, 2018. URL `http://mathworld.wolfram.com/TransitionMatrix.html`. [Online; accessed 21.09.2018].

Wikipedia contributors. Confusion matrix — Wikipedia, the free encyclopedia, 2018a. URL `https://en.wikipedia.org/w/index.php?title=Confusion_matrix&oldid=849067221`. [Online; accessed 21.09.2018].

Wikipedia contributors. Coefficient of determination — Wikipedia, the free encyclopedia, 2018b. URL `https://en.wikipedia.org/wiki/Coefficient_of_determination`. [Online; accessed 21.09.2018].

Task 1 Code

In [1]:

```
#Import needed modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


#Importing data to variables to dataframe and numpy arrays.
data = np.genfromtxt('data\chess-games.csv')
datanames = pd.read_csv('data\chess-games-names.csv', header=None)
chessname = np.copy(datanames)
chessdata = np.copy(data)
```

In [2]:

```
#Grapping names only from original data
name_lst = []
for name in chessname:
    name_lst.append(name[1])
```

In [3]:

```
def make_transitionmat(chessdata):
    """
    make_transitionmat function takes in the chessdata as argument the function will then
    sort the dataframe into a NxN transition matrix and return it.
    """

    #Initate empty 291X291 matrix (all players)
    A = np.zeros((291,291))
    #each row containing player id white, black and if win/tie/loss
    for match in chessdata:
        #If white won then black lost, holding row(black id) and column (white id)
        if match[2] == 1:
            #Typecasting to int because data is taken as string.
            row = int(match[1])
            column = int(match[0])
        #If black won then white lost, holding row(white id) and column (black id)
        elif match[2] == 0:
            row = int(match[0])
            column = int(match[1])

            #Filter out ties.
        if match[2] != 0.5:
            #Adding +1 on all lost matches on rows and columns.
            A[row,column] += 1

    #Normalizing each row.
    for i in range(291):
        A[i,:] *= 1/A[i,:].sum()

    return A
A = make_transitionmat(chessdata)
```

In [4]:

```
def power_method(A, iterations, init_vec = None):
    """
    power_method takes the transition matrix, the amount of iterations and a initial vector as arg
uments,
    if there is no initial vector given the function will assign a fair initial vector as 1/N on
each row.
    The function will return the steady state vector if it converges.
    """
    A = A.T
    if init_vec == None:
        init_vec = np.ones_like(A[0])/len(A)
    for i in range(iterations):
        AA = np.linalg.matrix_power(A,i)
        AAA = AA@init_vec
    return AAA


#Making a new pandas dataframe with the steady state vector. Then adding chess player names in fir
st column and the rank given from the power method in the second column
#Then sorting from highest to lowest rank.
df_new = pd.DataFrame(power_method(A, 100))
df_new[0] = name_lst
df_new['Rank'] = power_method(A, 100)
df_new.columns = ['Player','Rank']
df_new1 = np.copy(df_new)
df_new1 = pd.DataFrame(df_new1)
df_new1.columns = ['Player','Rank']
df_new1 = df_new1.sort_values(by=['Rank'],ascending=False)
df_new1.head(n = 10)
```

Out[4]:

	Player	Rank
1	Karjakin Sergey	0.0770343
17	Svidler Peter	0.0644529
73	Andreikin Dmitry	0.0606796
128	Aronian Levon	0.0581414
178	Matlakov Maxim	0.0563554
2	Ivanchuk Vassily	0.0447092
131	Kramnik Vladimir	0.0411574
29	Vachier-Lagrave Maxime	0.0409471
54	Tomashevsky Evgeny	0.0335337
157	Areshchenko Alexander	0.0318338

In [5]:

```
def google_mat(alpha, S):
    """
    google_mat takes a tuning scalar with values from 0 to 1 and the stochastic transition matrix
as arguments.
    The function will take the google algorithm and return the google transition matrix.
    """

    e = np.ones(len(S))
    E = 1/len(S)*np.outer(e,e)
    G = alpha*S + (1-alpha)*E
    return G


def check_stochasticity(A):
    """
    check_stochasticity checks if the matrix is stochastic by summing all rows and checks if this
equals one.
    Since this is numeric i have a threshold of 0.1 because some values might be 0.99 etc,
    but want it to fail if it is too low like 0.8 etc.
    """

    column = 1
    testcheck = 0
    threshold = 1e-1
    for row in A:
        threshold_test = abs(1-row.sum())
        if threshold_test > threshold:
            print('Sum of row in column %d does not equal one!'%column)
            testcheck = 1
    column += 1
    if testcheck == 0:
        print('Test finished: Matrix is stochastic')

#Checking if the matrix is stochastic before i use it in google alogrithm function.
check_stochasticity(A)

#Make new variable with the google transition matrix with tuning 0.85 and 100 iteration. Then assi
gn new pandas dataframe,
#with the steady state google vector. The dataframe is then given the column names "Players" and r
ank. The dataframe is then sorted from highest to lowest rank.
df_make_google_matrix = google_mat(0.85, A)
df_new_google = pd.DataFrame(power_method(df_make_google_matrix, 100))
df_new_google[0] = name_lst
df_new_google['Rank'] = power_method(df_make_google_matrix, 100)
df_new_google.columns = ['Player','Rank']
#Changing name of dataframe to avoid using the ordered dataframe in regression model.
df_new_google1 = np.copy(df_new_google)
df_new_google1 = pd.DataFrame(df_new_google1)
df_new_google1.columns = ['Player','Rank']
df_new_google1 = df_new_google1.sort_values(by=['Rank'],ascending=False)
df_new_google1.head(n = 10)
```

```
Test finished: Matrix is stochastic
```

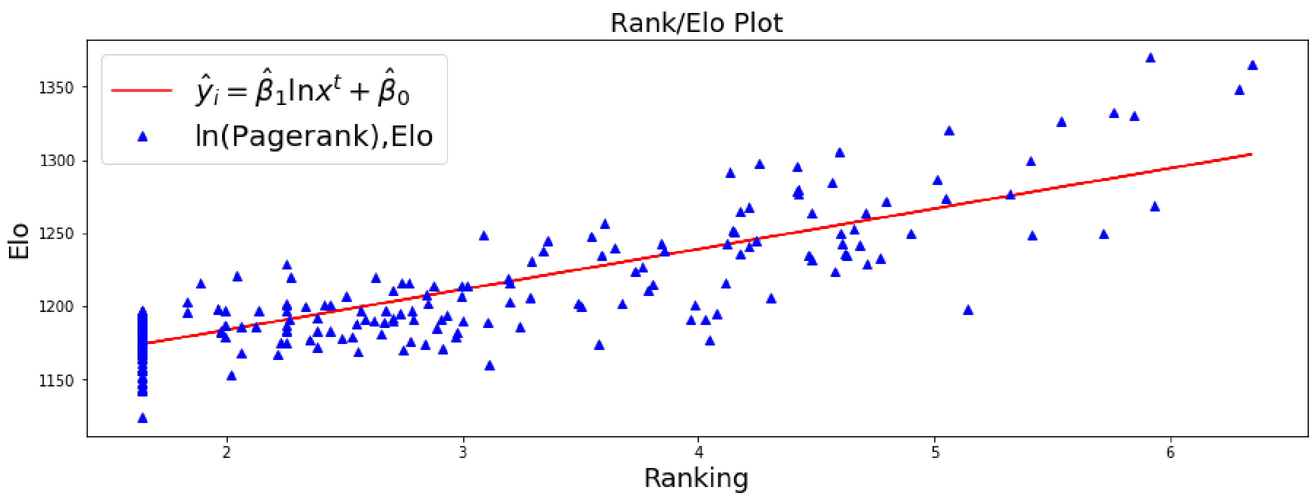
Out[5]:

	Player	Rank
1	Karjakin Sergey	0.0569849
17	Svidler Peter	0.0539471
73	Andreikin Dmitry	0.0376787
128	Aronian Levon	0.0369885
2	Ivanchuk Vassily	0.0345631
29	Vachier-Lagrave Maxime	0.0317558
178	Matlakov Maxim	0.0303571
131	Kramnik Vladimir	0.025497
54	Tomashevsky Evgeny	0.0224247
10	Grischuk Alexander	0.0222977

In [6]:

```
#Importing matplotlib module for plotting
import matplotlib.pyplot as plt

#Importing the chess elo data assigning the chess data to a pandas dataframe and renaming columns
to Players and Elo rank.
chess_elo = pd.read_csv('data\chess-games-elo.csv', header = None)
df_chess_elo = pd.DataFrame(chess_elo)
df_chess_elo.columns = ['Player', 'Elo']
# Performing np.copy is to avoid changing the original data.
X = np.copy(df_chess_elo)

#Taking logarithm of the data and multiplying with
Rank_temp = np.log(np.copy(df_new_google['Rank'])*10**4)

def linearfit_estimate(X):
    """
    linearfit_estimate takes the matrix with players and elo as argument, the function then sorts
the matrix for
    vectorized regression model, it will then return the whole regression model y as a array.
    Here we assume gaussian distributed error so that we can minimize the error and use the follow
ing multiplications."""

    player_temp = np.copy(df_chess_elo['Player'])
    #Assigning the r vector with the steady vector of the google transition matrix and the regular
one.
```

```
    r = np.copy(df_chess_elo['Elo'])
    ones = np.ones_like(r)
    #Rearranging the matrix to have ones in first column and elo ranks i second (using the logarit
hm of the data in the regression line).
    A = np.array((ones,Rank_temp)).T

    w = (np.linalg.inv(A.T@A))@A.T@r
#     Returning the linear regression model

    return w



# Performing the linear regression on the chess elo data.
# The regression model is plotted with the logarithmic paramater, then the google steady state val
ues are plotted with the regression model on a logarithmic y-scale.
reg_coeffs = linearfit_estimate(X)
regline_goog = reg_coeffs[1]*Rank_temp + reg_coeffs[0]
plt.figure(figsize = (15,5))
plt.plot(Rank_temp,regline_goog, label = r'$\hat{y}_{i} = \hat{\beta}_{1}\lnx^{t} +
\hat{\beta}_{0} $', color = 'red')
plt.plot(Rank_temp,df_chess_elo['Elo'],'^', label = 'ln(Pagerank),Elo', color = 'blue')
plt.legend(loc = 'best', prop={'size': 20})
# plt.xscale('log')
plt.title('Rank/Elo Plot',fontsize=18)
plt.ylabel('Elo',fontsize=18)
plt.xlabel('Ranking',fontsize=18)
# axes = plt.gca()
# axes.set_ylim([0.0003,1])
plt.show()

np.mean(df_chess_elo['Elo'].values)
```



Out[6]:

```
1199.5257731958764
```

In [7]:

```
def residual(regline):
    """
    residual function takes in the steady state vector and the regression model as arguments
    the function then finds the residual by find the difference between the model and the data.
    Then it find the residual sum of squares and the total sum of squares and then finding the R^2
.

    #https://en.wikipedia.org/wiki/Residual_sum_of_squares
    #https://en.wikipedia.org/wiki/Coefficient_of_determination
    """

    r = np.copy(df_chess_elo['Elo'])
    resid = []
    for i,j in zip(r,regline):
        resid.append(j-i)
    resid = np.array(resid)
    SSres = sum(resid**2)
    SStot = sum((r - np.mean(r))**2)
    #Relative square error used to find the R^2
    E_RSE = SSres/SStot
    R_Sqr = 1 - E_RSE

    return R_Sqr


#print out the coefficient of determination R^2 and the estimators of the regression model for the
google vector
print('R^2 Google',residual(regline_goog))
print('Beta_0_hat = ',reg_coeffs[0])
print('Beta_1_hat = ',reg_coeffs[1])
```

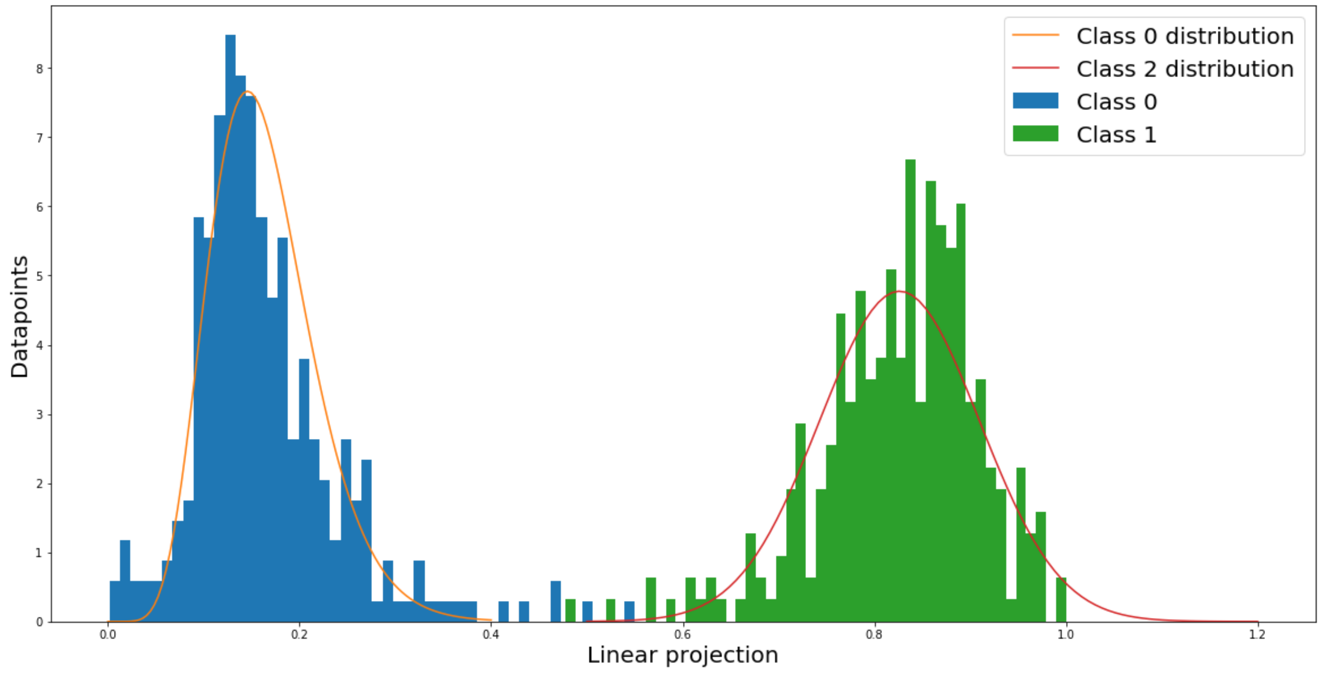
```
R^2 Google 0.721446886994
Beta_0_hat =  1128.80571481
Beta_1_hat =  27.5725796107
```

Task 2 Code

In [8]:

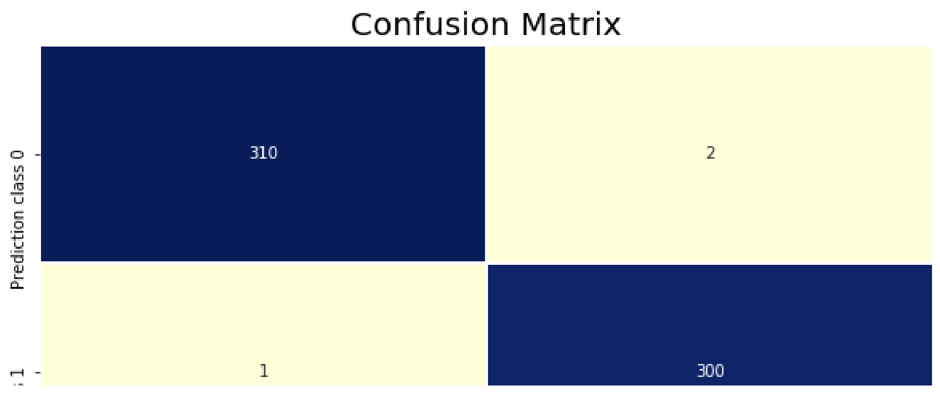
```
import numpy as np
import matplotlib.pyplot as plt
import math

#Import data
data = np.genfromtxt('data\optdigits-1d-train.csv')
alpha = 9

#putting each class in its own list for finding priors and for training
C0 = []
C1 = []
#sorting classes
for i in data:
    if i[0] == 0:
        C0.append(i[1])

    elif i[0] == 1:
        C1.append(i[1])

C0 = np.array(C0)
C1 = np.array(C1)
#Priors for each class
C0_prior = len(C0)/(len(C0)+len(C1))
C1_prior = len(C1)/(len(C0)+len(C1))

#The estimated parameters for gamma distribution and normal distribution
beta = (1/(len(C0)*alpha))*sum(C0)
mu = (1/len(C1))*sum(C1)
variance = (1/len(C1))*sum((C1-mu)**2)


def gammadistr(x,alpha,beta):
    """
    The gammadistr function takes in the datapoint or an numpy array, alpha parameter and the beta
parameter as arguments
    the result of the gamma distribution is then returned.
    """
    #as long as alpha is bigger then 0 and integer
    Tau_alpha = math.factorial(alpha - 1)
    result = (1/(beta**alpha*Tau_alpha))*x**(alpha-1)*np.exp(-x/beta)
    return result



def normaldistr(x,mu,variance):
    """
    The normaldistr function takes in the data point or an numpy array, mu and variance parameters
    the function the returns the values of the normal distribution.
    """
    result = (1/(np.sqrt(2*np.pi*variance)))*np.exp(-((x - mu)**2)/(2*variance))
    return result
```

```
#values to plot the 'continues' distributions
x1 = np.linspace(0.5,1.2,100)
x = np.linspace(0,0.4,100)


#values to plot the 'continues' distributions
P_C0 = gammadistr(x,alpha,beta)
P_C1 = normaldistr(x1,mu,variance)


#Plotting the histogram and distributions together in one plot
plt.figure(figsize = (20,10))
plt.hist(C0, bins = 50, label = 'Class 0',density = True)
plt.plot(x,P_C0, label = 'Class 0 distribution')
plt.hist(C1,bins = 50, label = 'Class 1',density = True)
plt.plot(x1,P_C1, label = 'Class 2 distribution')
plt.legend(loc = 'best', prop={'size': 20})
plt.ylabel('Datapoints',fontsize = 20)
plt.xlabel('Linear projection',fontsize = 20)
plt.show()
```



In [9]:

```
#Import seaborn to make nice confusion matrix
import seaborn as sns

#Importing training data
data = np.genfromtxt('data\optdigits-1d-train.csv')


def bayes_class(data,alpha,beta,mu,variance,C0_prior,C1_prior):
    """
    The bayes_class function takes in the data, alpha and beta parameters for the gamma
distribution
    and the mu and variance of the normal distribution, and the prior probability for class 0 and
for class1.
    The data is then classified using the bayes theorem without the equal terms.
    Here the results is added into a result list where the output is zero for class 0 and one for
class 1.
    """

    result = []
    for x in data:
        P_C0 = gammadistr(x,alpha,beta)
        P_C1 = normaldistr(x,mu,variance)

        if C0_prior*P_C0 > C1_prior*P_C1:
            result.append(0)
        else:
            result.append(1)
    return result

#Adding training data to the classifier
C0_bayes = bayes_class(C0,alpha,beta,mu,variance,C0_prior,C1_prior)
C1_bayes = bayes_class(C1,alpha,beta,mu,variance,C0_prior,C1_prior)


def confusion_matrix(data0_train,data1_train):
    """
    The confustion_matrix function takes in the training data for class 0 and training data for cl
ass 1
    then the data is training with the classifier and the results are sorted in a 2x2 matrix with
the True positives and True negatives
    are given on the diagonal axis.
    """
    data0_train = bayes_class(data0_train,alpha,beta,mu,variance,C0_prior,C1_prior)
    data1_train = bayes_class(data1_train,alpha,beta,mu,variance,C0_prior,C1_prior)

    confusion_mat = np.zeros((2,2))
    class0_0 = 0
    class0_1 = 0
    class1_1 = 0
    class1_0 = 0
    #Sorting the results
    for i in data0_train:
        if i == 0:
            confusion_mat[0,0] += 1
        else:
            confusion_mat[0,1] += 1
    for i in data1_train:
        if i == 1:
            confusion_mat[1,1] += 1
        else:
            confusion_mat[1,0] += 1

    return confusion_mat



#Setting up the confusion matrix
df_cm = pd.DataFrame(confusion_matrix(C0,C1), index = [i for i in ["Prediction class 0","Prediction
class 1"]],
                  columns = [i for i in ["Class 0","Class 1"]])

#plotting the confustion matrix with seaborn heatmap
plt.figure(figsize = (10,5))
sns.heatmap(df_cm, linewidths=1 , cmap="YlGnBu",cbar=False, annot=True,fmt='g', annot_kws={"size":
10})
plt.title('Confusion Matrix',fontsize = 20)
plt.show()


#Finding the accuracy of the prediction
#https://en.wikipedia.org/wiki/Confusion_matrix
#Taking the sum of the true positives and negatives divided by the total population
Population = sum(sum(confusion_matrix(C0,C1)))
Accuracy = sum(np.diag(confusion_matrix(C0,C1)))/Population

print('Accuracy of the prediction: ',Accuracy)
```



```
Accuracy of the prediction:  0.995106035889
```

In [10]:

```
#Importing the utility python module.
from he1_util import get_msg_for_labels

#Importing the test data
data_test = np.genfromtxt('data\optdigits-1d-test.csv')

#Printing out the secret message
print('The secret message is,',get_msg_for_labels(bayes_class(data_test,alpha,beta,mu,variance,C0_
prior,C1_prior)))
```

```
The secret message is, nevergonnagiveymuup
```