Group Assignment Report 2DRR00, Group 36

Xiuqi Shi 1935526 ; Yuan Mo 1934783 ; Shiyun Kong 1920847 ; Alex Xu 1920456 ; Pieter Zhu 2008254

FALL 2023

(1) Vector angles and movie ratings (recommender systems)

(a) A matrix G representing chosen movies and coordinate ratings

Tabel 1 is the 8 discourses (movies or anime) we chose to rate, and each person's randomly chosen half of those can be disclosed at this stage.

movie name	Tim	Alex	Yuan	Alan	Pieter
spirited away		1	1	2	
love letter	1	0			0
titane				0	
thelma & Louise			0		-2
sousou no Frieren	2	2	0		
Youkoso Jitsuryoku Shijou Shugi no Kyoushitsu				0	2
schindler's list	2	1		1	0
parasite	0		2		

Table 1: matrix G containing movie ratings, with secret ratings hidden.

(b) What could be a reason to scale -2,...,2 instead of the usual 1,...,5

Compared with the rating scale of 1-5, the rating range of -2 to 2 will make the difference in angles after vectorization of each person's taste more obvious.

According to Cosine Similarity Formula: $\text{cosine\_similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \cdot \|\mathbf{B}\|}$

Case 1, $\mathbf{A} = [a_1 a_2]^T, \quad \mathbf{B} = [b_1 b_2]^T, \quad \forall (a_i, b_i) \in \{1, 2, 3, 4, 5\}$

Vector Entries from 1 to 5 (Positive Range): The $\mathbf{A} \cdot \mathbf{B}$ is positive, leading to acute angles. The magnitude resulting from the operation of norms can be relatively large because the vectors can take greater values e.g. {3, 4, 5} into the squares.

Case 2, $\mathbf{A} = [a_1 a_2]^T, \quad \mathbf{B} = [b_1 b_2]^T, \quad \forall (a_i, b_i) \in \{-2, -1, 0, 1, 2\}$

Vector Entries from -2 to 2 (Mixed Range): The dot product can be negative, potentially resulting in a broader range of angles, including obtuse angles. The magnitude resulting from the operation of norms may be smaller compared to Case 1 because the vectors can take values in a smaller range.

This indicates that in case 2, rating range of -2 to 2, will yield a cosine similarity difference between vectorization of each person's taste, hence make obvious of each person 's taste.

(c) Who has the 'average' taste, and who has the most extreme taste?

movie name	Tim	Alex	Yuan	Alan	Pieter	Typical norm of movies
spirited away	1.29	1	1	2	1.17	1.33
love letter	1	0	0.54	0.54	0	0.33
titane	0.625	0.5	0.375	0	0.5	0
thelma & Louise	1.13	1	0	0.88	-2	1
sousou no Frieren	2	2	0	1.04	1.17	1.33
Youkoso Jitsuryoku Shijou Shugi no Kyoushitsu	1.13	1	0.88	0	2	1
schindler's list	2	1	0.88	1	0	1
parasite	0	1	2	0.88	1	1
Typical norm of people	1.25	1	0.75	0.75	1	1

Table 2: Matrix of movie ratings. Secret ratings filled with prediction, using ∞-norms and 1-norms.

We used averaged $\|\mathbf{A}\|_1$ to answer this question, it is the average of summation of magnitude of a single person's rating, which indicates who is likely to give extreme |points| to the movies assuming missing entries has minor effect in altering a person's taste, hence, typical. $\|\mathbf{A}\|_1$ for {Tim, Alex, Yuan, Alan, Pieter}, is respectively, {1.25, 1, 0.75, 0.75, 1}. The conclusion after calculation is that Tim has the most extreme taste (with average $\|\mathbf{A}\|_1$ of 1.25), Alex and Pieter have relatively average taste (average $\|\mathbf{A}\|_1$ of 1) compare to Tim's extreme taste and Yuan & Alan's conservative taste (average $\|\mathbf{A}\|_1$ of 0.75). See Table 1 and Table 2.

(d) Try to predict the missing ratings of your group's member, based on the other ratings of your group

We, again, use norms to make predictions. We first reused averaged $\|\mathbf{A}\|_1$ to calculate the typical value of each person's rating preference. After calculation, average (typical) norms implies a person's general taste. Then using $\|\mathbf{A}\|_\infty$ to calculate the typical norms of each movie's score, this will show how many magnitude of points people tend to rate this movie. Combining these two indicators, we can predict the movies that each person has not publicly rated (take the average of the two data). See Table 1 and Table 2. We realise that, by doing this, we will never have a negative value being predicted. Thus this is a drawback of our method, hopefully, we can resolve this issue when doing Truncated Singular Values Decomposition.

movie name	Tim	Alex	Yuan	Alan	Pieter
spirited away	1	1	1	2	2
love letter	1	0	2	2	0
titane	-2	2	0	0	-1
thelma & Louise	-1	1	0	0	-2
sousou no Frieren	2	2	0	2	1
Youkoso Jitsuryoku Shijou Shugi no Kyoushitsu	2	-1	0	0	2
schindler's list	2	1	1	1	0
parasite	0	1	2	1	1

Table 3: Compare with original matrix, with secret ratings revealed.

Comparing the form filled with our predictions in Figure 2 with the complete original form in Table 3, the conclusion is that the difference is significantly large. Just using norms for prediction does not work well. Words written after the entire part 1 is done: We tried to quantify the difference in prediction of entire rating matrix with the original matrix for various methods (including Truncated Singular Values Decomposition later). The measure we decided to choose is taking the **MSE** $\forall$ elements between the matrices ($\frac{1}{8\times 5}\sum_{ij=(1,1)}^{(8,5)}(y_{ij}-\hat{y}_{ij})^2$) because our rating is $\forall(a_i,b_i)\in\{-2,-1,0,1,2\}$, the decimal point differences we do not really care, as they would even be rounded up into integers if we convert the predicted value to real ratings; what we want to emphasize is the magnitude of differences bigger than 1. It turns out, the MSE is 3.54, which indeed demonstrate the norm completion does not work well

(e) Now do the same as (d), but use the more advanced concept of low-rank matrices, via a TSVD (for instance, with k = 2). Does this give better results than in (d)?

	Tim	Alex	Yuan	Alan	Pieter
spirited away	1.553795	1.315325	0.964609	1.129171	0.886973
love letter	0.585223	0.495405	0.363311	0.425292	0.334070
titane	0.433380	0.366866	0.269046	0.314945	0.247392
thelma & Louise	-0.374672	-0.317169	-0.232600	-0.272281	-0.213879
sousou no Frieren	1.634178	1.383371	1.014511	1.187587	0.932859
Youkoso Jitsuryoku Shijou Shugi no Kyoushitsu	1.236953	1.047111	0.767911	0.898916	0.706106
schindler's list	1.392580	1.178853	0.864526	1.012013	0.794945
parasite	1.018083	0.861832	0.632035	0.739860	0.581166

Table 4: TSVD Rank 1

	Tim	Alex	Yuan	Alan	Peter
spirited away	1.422141	1.277173	1.319924	1.335240	0.525426
love letter	0.680386	0.511317	0.403842	0.554991	-0.065426
titane	0.536029	0.388497	0.198896	0.348142	0.069525
thelma & Louise	0.098226	-0.232997	-0.161216	0.250388	-1.910142
sousou no Frieren	2.298250	1.539893	0.137854	1.006123	0.721833
Youkoso Jitsuryoku Shijou Shugi no Kyoushitsu	0.923962	0.995685	0.611449	0.450640	2.071503
schindler's list	1.754730	1.252179	0.693157	1.200466	-0.001748
parasite	0.113627	0.643621	1.954259	1.107174	0.583608

Table 5: TSVD Rank 3

From comparing Table 4 and 5 with Table 3 and 2, it can be seen that using Truncated Singular Values Decomposition (TSVD) is more accurate in predicting film ratings than using norms only. Or, using numerical mesures: MSE of TSVD rank 1: 0.95; MSE of TSVD rank 3: 0.72, indeed this is with average imputations: Before we do the SVD, we imputed NaN with something similar to what we did in part d, instead, we just take the average of every column and every row, then average to fill into the elements, forget about the norms. Because firstly SVD cannot be done with NaN values. We think it is a better choice than putting zeros in the NaNs' places, because our methods incorporates some of the wisdom we used in part (c) and (d), and we indeed verified that our method has better MSE compared to just filling 0s in NaNs' place. Even without the average imputation, we would have MSE of TSVD rank 1: 1.2; MSE of TSVD rank 3: 0.93. Our singular values are {5.51, 2.61, 2.05, 1.21, 1.0} respectively, we can see the first 3 eigenvalues are more than 2, we think this could be a big change in eigenvalues from the bottom 2. We have the SVD of the original matrix $\mathbf{A}=\mathbf{U\Sigma V}^T$. We then truncate the SVD by keeping only the first $\mathbf{k}$ singular values. $\mathbf{\Sigma}_k$ becomes the diagonal matrix with the three singular values. $\mathbf{U}_k$ and $\mathbf{V}_k$ can be obtained by truncating the corresponding columns of $\mathbf{U}$ and $\mathbf{V}$. The rank-k approximation of the matrix $\mathbf{A}$ is $\mathbf{A_k}=\mathbf{U_k\Sigma_k V_k}^T$. Substituting $\mathbf{k}=3$ we get $\mathbf{A}_3$. Similarly, by taking only 1 singular values, which is more justifiable with the significant gap between singular value of 5.51 and the rest, the rank-1 approximation $\mathbf{A}_1$ can be obtained with $\mathbf{k} = 1$. But lower the rank, lower the amount of important features retailed in the predicted matrix, so the rank 3 TSVD is definitely better than rank 1 TSVD, the question is, in what extent though. When setting $\mathbf{k} = 3$ improves the accuracy a bit (MSE of 0.72) compared to when setting $\mathbf{k} = 1$ (MSE of 0.95), which is in line with our expectations, because 5.51 is significantly larger than the rest, meaning that there obviously a single most important/representative feature in the matrix, which turns out compressed the data into rank 1 and can be recreated at the cost of 0.23 MSE more than rank 3 with the significant reduction in storage spaces.

(2) Google PageRank type ranking

(a) Create a matrix G representing the graph

For part (2), we used the euro−21 dataset. We created a set of unique team names from euro−21. We created an empty adjacency matrix of 24×24. Then, populated the adjacency matrix with wins for an arbitrary Team $\mathbf{i}$ against arbitrary Team $\mathbf{j}$ at index $\mathbf{A_{ij}}$ with 1; lose for an arbitrary Team $\mathbf{i}$ against arbitrary Team $\mathbf{j}$ at index $\mathbf{A_{ij}}$ with 0. In case of tie, we populate 0 at both position of $\mathbf{A_{ij}}$ and $\mathbf{A_{ji}}$

(b) Compute (using the computer) the eigenvector x corresponding to eigenvalue 1

The reason for finding the eigenvector corresponding to eigenvalue 1 is rooted in the fact that the Markov chain (including Google PageRank) should have a stationary distribution, and the principal eigenvector of the modified random walk matrix $\mathbf{A}$ is associated with probability distribution of a random surfer's visit to different web pages. In other words, we do not want to see the enlargement of probabilities (extension of eigenvector) during transitions of latent space during random walks. That is the special solution $\mathbf{x}$ to $\mathbf{Ax} = \mathbf{x}$.

(c) List the ordered items. Can you explain the ranking? Is it logical?

Table 6: Based on the results of the tournament, we have listed the logical order of the strongest teams (as in the second column of the table) according to the winner, final participants, top four, top eight and etc. The order of the strongest teams given by the computer using our algorithm is the fourth column of the table. We weight each win or lose/tie by 1 or 0/0 (no matter what it is final or group level), so the team that wins more and defeats stronger opponents (more wins counts as stronger) gets a higher ranking, or, higher relative importance.

As the Table 6 shows, this algorithm gives a logical ranking of teams that is not too far from reality. Taking into account factors such as luck of the draw, uneven exhibition of strength of teams when against opponent teams, probability of random walks, it is acceptable to have some variation. As far we we are aware, sp has the most difference in actual ranking of the real world and predicted ranking because what pagerank algorithm does, is also considering the opponents lost and win against sp. sp only wins weaker opponents, not winning any stronger opponents, and hence can be concluded more reasonably as get into the best-4 by chance. From this case, we can inductively think only assigning 1s to the win, not giving more weights to stronger stages of match can possibly

Ranking	Team Name (Real Ranking)	Full names vs.	Team Name (Produced Ranking)	Eigenvector component
1	it	Italy	it	0.14
2	en	England	en	0.11
3	sp	Spain	be	0.08
4	dk	Denmark	dk	0.07
5	be	Belgium	cz	0.07
6	cz	Czech Republic	nl	0.05
7	ch	Switzerland	ua	0.05
8	ua	Ukraine	fi	0.04
9	nl	Netherlands	sp	0.04
10	se	Sweden	at	0.04
11	at	Austria	se	0.03
12	fr	France	ru	0.03
13	pt	Portugal	pt	0.03
14	hr	Croatia	de	0.03
15	de	Germany	fr	0.03
16	wa	Wales	hr	0.02
17	fi	Finland	sk	0.02
18	sk	Slovakia	ch	0.02
19	ru	Russia	wa	0.02
20	hu	Hungary	tr	0.02
21	pl	Poland	sc	0.02
22	sc	Scotland	ma	0.02
23	tr	Turkey	hu	0.02
24	ma	North Macedonia	pl	0.02

Table 6: truth values vs. $0.85\widehat{G}$

lead to a deviation from what real life happened. But we still indeed think our logic of assigning 1 to the win 0 to lose/duce is a better measure of real ability of teams.

(d) Take another p-value, for instance p=0.99 or p=0.50. What is the influence on the PageRank?

We experimented with p=0.99. As can be seen from comparing Table 6 and Table 7, the change in p-value from 0.85 to 0.99 had little effect on the rankings, only changing the order of 9 & 10^{th} places, and 11 & 12^{th} places. The effect on the calculated value is that the standard deviation of the eigenvector components which becomes larger for higher p-values. And the gap between the maximum and minimum values becomes larger as well, i.e., a larger p-value makes the numerical differences between the teams more evident. Min-max difference is more obvious when comparing p=0.99 with p=0.5, but the rankings are mostly the same, as in previous case .

(e) Now assume you are the owner of the first node (i.e., the first site, or club, or...). You may change 1 element in the graph to maximize your PageRank: adding a link or, removing a link to your liking. What can you try to do this? Try some ideas and see how they turn out.

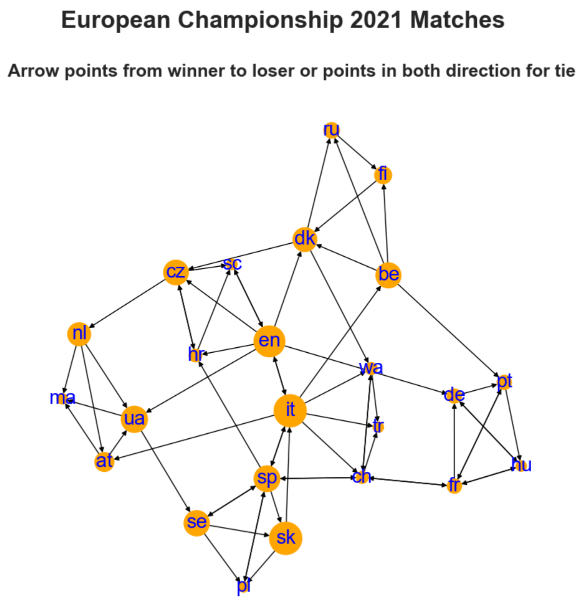
In order to achieve the maximum impact of changing only one element, that is, in our case, adding a winning result of one game to maximize the change in the ranking of one team, we choose to assume that we are the sk, as in Figure 1(b) sk is the first node viewing from bottom, and choose to add results of the game between sk and it. Our logic is to select the strongest (that will be adjacent

Ranking	Team Name	Eigenvector component	vs.	Team Name	Eigenvector component
1	it	0.15		it	0.10
2	en	0.11		en	0.09
3	be	0.09		be	0.07
4	dk	0.08		dk	0.06
5	cz	0.07		cz	0.06
6	nl	0.05		nl	0.05
7	ua	0.05		ua	0.05
8	fi	0.04		sp	0.04
9	at	0.03		at	0.04
10	sp	0.03		se	0.04
11	ru	0.03		fi	0.04
12	se	0.03		ru	0.03
13	pt	0.03		fr	0.03
14	de	0.03		pt	0.03
15	fr	0.03		de	0.03
16	hr	0.02		hr	0.03
17	sk	0.02		sk	0.03
18	ch	0.02		ch	0.03
19	wa	0.02		wa	0.03
20	tr	0.01		ma	0.02
21	sc	0.01		sc	0.02
22	ma	0.01		pl	0.02
23	hu	0.01		tr	0.02
24	pl	0.01		hu	0.02

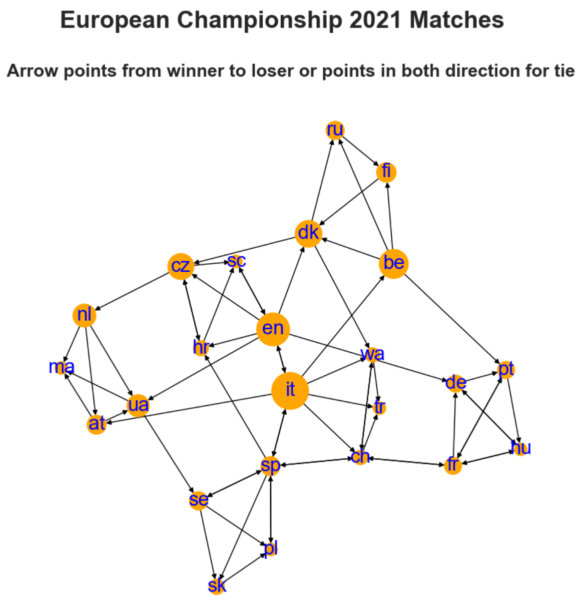
Table 7: $0.99\widehat{G}$ vs. $0.5\widehat{G}$

Ranking	Team Name	Eigenvector component	vs.	Team Name	Eigenvector component
1	sk	0.11		it	0.14
2	it	0.10		en	0.11
3	en	0.10		be	0.08
4	ua	0.07		dk	0.07
5	sp	0.07		cz	0.07
6	se	0.06		nl	0.05
7	be	0.06		ua	0.05
8	cz	0.06		fi	0.04
9	dk	0.06		sp	0.04
10	nl	0.05		at	0.04
11	at	0.04		se	0.03
12	fi	0.03		ru	0.03
13	ru	0.02		pt	0.03
14	pt	0.02		de	0.03
15	fr	0.02		fr	0.03
16	de	0.02		hr	0.02
17	hr	0.02		sk	0.02

Table 8: Team sk being raised vs. $0.85\widehat{G}$


European Championship 2021 Matches
Arrow points from winner to loser or points in both direction for tie
ru
fi
dk
be
cz
sc
nl
en
hr
wa
pt
ma
de
ua
it
tr
at
hu
sp
ch
fr
se
sk
pl

(a)


European Championship 2021 Matches
Arrow points from winner to loser or points in both direction for tie
ru
fi
dk
be
cz
sc
en
nl
hr
wa
ma
ua
it
de
pt
tr
at
hu
ch
sp
fr
se
pl
sk

(b)

Figure 1: Sizes of nodes in the graph represent the relative importance of the node; the larger the node, the higher the ranking

to sk, with the highest ranking) opponent from all possible opponents and write the game results to be the victory of sk, which should have the greatest increase in ranking, see Figure 1(a). Before the change, sk's ranking among the 24 teams was 17, and the value was 0.02 (Table 8); after adding the results of the game between sk and it, sk's ranking rose to 1 (you can immediately observe that on the size of node in Figure 1(a)), and the value became 0.14. Compared with the previous ranking, it has risen by 16 places to the winner of championship, which is a very significant change (both with p=0.85).

(3) Eigenvector-based clustering

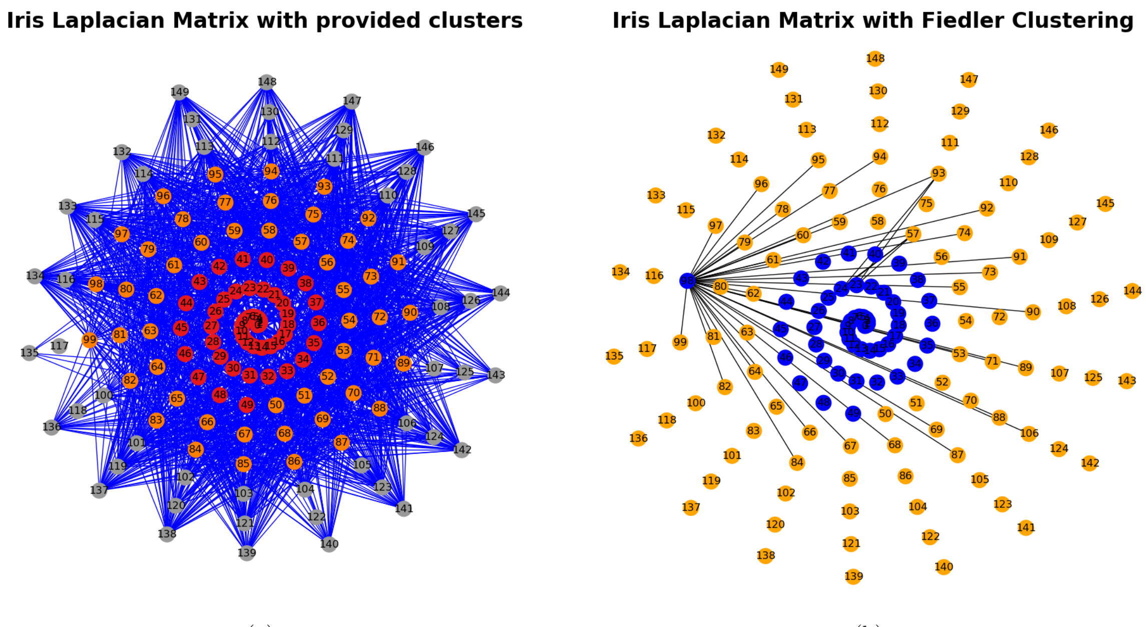
(a) Briefly explain our items and graph

We are using an external dataset, a small classic dataset from Fisher, 1936. One of the earliest known datasets used for evaluating classification methods. This is one of the earliest datasets used in the literature on classification methods and widely used in statistics and machine learning. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are not linearly separable from each other. Predicted attribute: class of iris plant. This is an exceedingly simple domain. This data differs from the data presented in Fishers article[1]. The 35th sample should be: 4.9,3.1,1.5,0.2,"Iris-setosa" where the error is in the fourth feature. The 38th sample: 4.9,3.6,1.4,0.1,"Iris-setosa" where the errors are in the second and third features.

[1]identified by Steve Chadwick, spchadwick@espeedaz.net

(b) Plot the graph by computer

The dataset we choose includes the data of 150 species of Iris in 3 categories, which is a 3 categories $\times$ 50 rows $\times$ 4 columns data. The first figure is Default clustered data, you can see that there are a lot of complex connections in the Figure 2(a) below, but you can still see that the three types of nodes with different colours (Red, Orange, Grey respectively) show a regular ring distribution.


Iris Laplacian Matrix with provided clusters
Iris Laplacian Matrix with Fiedler Clustering

(a) (b)

Figure 2:

(c) Explain how Iris Dataset becomes an Laplacian Matrix

We created a matrix that represents the pairwise similarities or distances between data points. Then, We used distance metrics of Euclidean to created an adjacency matrix represents the connections between data points. Next, we converted the similarity matrix to an adjacency matrix by applying a threshold of 2.0. The degree matrix is a diagonal matrix where each diagonal element represents the degree (number of connections) of a node construct from adjacency matrix. The Laplacian matrix is, finally, computed as the difference between the degree matrix and the adjacency matrix. This adjacency matrix is in dimension 150 $\times$ 150

(d) Compute (using the computer) the Fiedler vector

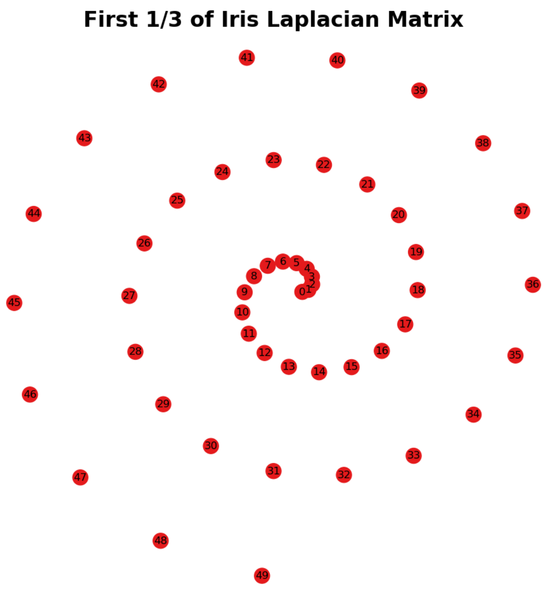
The Fiedler vector and Fiedler value are associated with spectral clustering and graph partitioning. The Fiedler vector captures the "betweenness" or "connectivity" of nodes in a graph. The Fiedler vector corresponds to the second smallest eigenvalue of the Laplacian matrix (by solving for the second diagonal element in upper triangular matrix $\mathbf{Q^T}$ in QR Decomposition $\mathbf{Rx} = \mathbf{Q^T b}$), in our case, is 0.50, which means the graph is disconnected, see Figure 2(b).

(e) Color the nodes corresponding to the sign of the Fiedler vector, and display the result. Is it as expected? Can you explain the result?

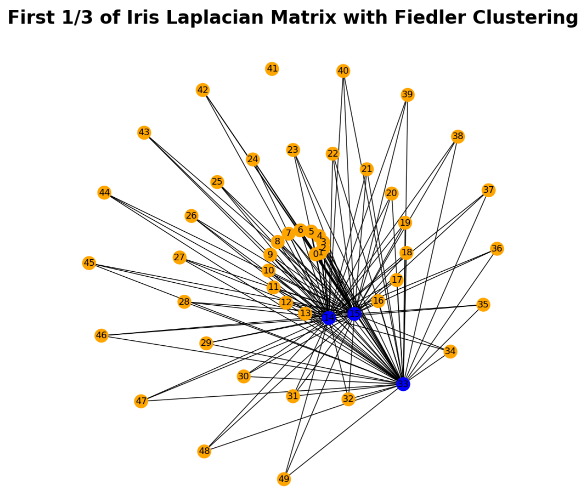
See Figure 2(b). With the exception of one blue dot on the left side, into the yellow region of the graph, the rest of the dots are very clearly distributed in a ring. Given, the excuse of coloring three semantically meaningful cluster into two, this is very close to what we expected from the Fiedler Clustering, showing two very distinct clusters.

(f) For this last part, take another graph, with more, or fewer edges. Which Fiedler value is smaller: that for the graph with many or few edges? Which graph seems easier to partition: the graph with many or few edges? Do you see a relation?

Since our first plot had few edges, we made the second plot by taking the first 50 rows $\times$ 50 columns of the Laplacian Matrix (to make a chart with more edges because we had few edges in the previous part). Then we made a new plot from the new Laplacian Matrix, as Figure 3(a) show, which indeed does not have any edges because it is one of the categories comes with the iris dataset, not connected to other clusters. Then, we computed (using the computer) the Fiedler vector and color the nodes corresponding to the sign of the Fiedler vector again, and in Figure 3(b) shown below.



(a)



(b)

Figure 3:

It can be clearly observed that the new graph has more edges than the original graph with complete data. And the result of the clustering for the new Laplacian Matrix is not as obvious as that for the original complete dataset. The graph with few edges seems easier to partition. We believe that the relation here is that the smaller the Fiedler value (the second smallest eigenvalue) of a data set is, the fewer the edges are, and the easier it is to partition, and the better the clustering result.

Keyword \ Venue	CAMAD	EUNICE	HAISA	HPCC-ICESS	IJESMA	ISCA	KMIS	NMR	SPRINGL	SSV
algorithm	2	8	0	24	0	5	0	2	1	1
cellular	2	1	0	1	0	0	0	0	0	0
game	1	1	0	1	0	0	1	1	0	0
hardwar	1	0	1	4	0	18	0	0	1	0
internet	2	6	2	0	2	0	0	0	0	0
mobil	10	8	0	6	17	5	2	0	2	0
network	58	60	4	38	2	25	12	0	3	0
search	0	1	0	1	2	4	1	0	0	0
secur	4	4	29	5	1	12	3	0	4	0
web	0	2	0	3	3	1	13	0	2	0

Figure 4: Venue-Keyword Matrix of the pre-processed titles of all published papers in venues in recent years

(4) Data mining: term–document matrix

(a) Briefly explain your terms and documents, or customers and products

Our fourth part uses a dataset of 10 rows × 10 columns. Figure 4. Each column represents one of the venues to be ranked, and terms are the keywords of the pre-processed titles of all published papers in those venues in recent years. Each matrix entry indicates the frequency of the corresponding term in the corresponding venue. The matrix formed by this dataset contains 58% of non-zero values.

(b) Give the matrix

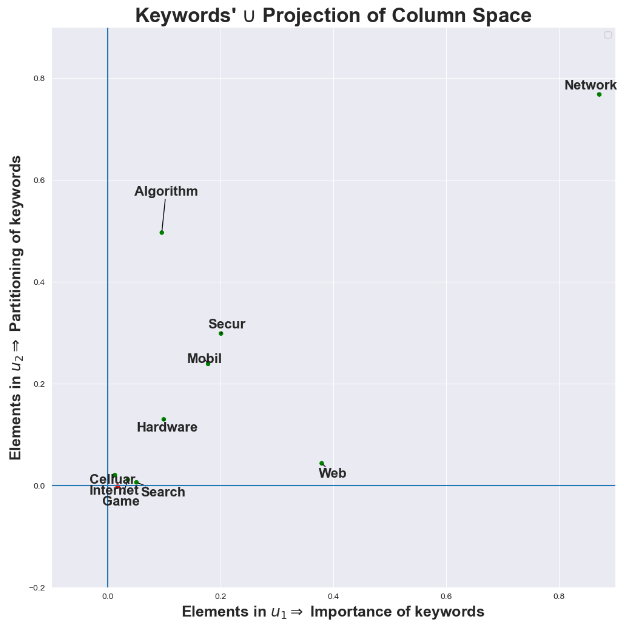
See Figure 4

(c) Computation

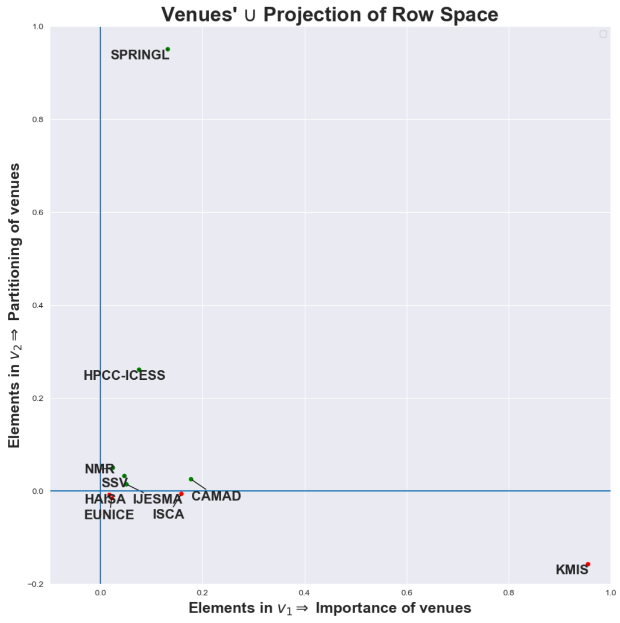
Singular Value Decomposition (SVD) extracts the dominant features in columns and rows of a matrix and stores the information in $\mathbf{U}$ and $\mathbf{V}^T$ accordingly. The $\mathbf{\Sigma}$ matrix stores information of relative importance of those features. In this particular case, we only select first two most predominant orthogonal direction in column space or row space as a input basis to the linear transformation ($\mathbf{A} \cdot [\mathbf{u_1}\mathbf{u_2}]^T$, then, $\mathbf{A} \cdot [\mathbf{v_1}\mathbf{v_2}]^T$). This orthogonality guarantees that we have established a linear transformation of entire latent subspace as the linear combinations of those two basis vectors span the whole subspace. The transformed subspaces are indeed in $\mathbb{R}^2$, which not only indicates we extracted the dominant features but also reduced the dimensionality of data.

(d) The intention of the previous part is that we now can visualize the c_j. Plot the c_j in $\mathbb{R}^2$ as points, with the terms as labels. Likewise, in a separate figure, plot the d_j , with the documents as labels.

See Figure 5(a) and 5(b).



(a)



(b)

Figure 5: Feature Projection of the Venue-Keyword matrix

(e) **Are the results as expected? Can you explain it? Are related terms (or products) more or less in the same direction? Are related documents (or customers) more or less in the same direction? Do you see nice unexpected connections?**

The two graphs (Figure 5(a) and 5(b)) generated from the calculation of this dataset show results that are more in line with expectations. In the keyword graph, the keywords that are related to each other, such as web, internet, network, and other keywords related to information network, are all in the same direction in the graph, while cellular, a biological term, is not in the same direction with other keywords.

4.5.1 Explanation of Keywords (Column Space):

The more often a keyword appears and the more often it appears in important magazines, the more important keyword it is and appear to the right on the x-axis; the more concentrated a keyword is, the more to the top it is on the y-axis. "Network" has high importance and partitioning, indicating a high frequency of mentioning across all venues, and an extremely concentrated occurrences. More concentration here means that the bigger gap between some most-posted venues and the majority of other less posted venues , i.e., IN THIS CASE, CAMAD, EUNICE, HPVV-ICESSS, and ISCA combined network appeared 181 times, and the remaining six avenues combined network appeared a total of 21 times. The same situation can be used to explain Algorithm and Secur. while keywords such as Game and search appear very rarely and scattered, so clustered around the origin point.

4.5.2 Explanation of Venues (Row Space):

The Partitioning of Venue is undertaken by determining whether there exists a extreme allocations of one of the keywords out of all keywords in its venue in terms of proportions: The more extreme distribution of (especially more important, if there exists) keywords (see importance of keywords in Figure 5(a)'s x-axis), the more negative the partitioning vector components rates it.

The Importance of Venue checks for whether there exists a extreme distribution of important keywords {Network, Web, Secur, Mobil} (under criteria vector component in $\mathbf{u}_1$ greater than 0.1) relative to the rest of keywords in terms of proportion and checks if there exist a similar amount of papers (remember the keywords are the keywords of the pre-processed titles of all published papers in those venues in recent years) under two of the most populated keywords.

(5b) Programming language competition

(a) Introduction to the Problem

The three languages we chose to compare are Python (Numpy + Pytorch-GPU), Julia (LinearAlgebra) and Java (Math3.linear). The three tasks we chose to program and compare the time spent on the programs are:

- Task1. 100 epoch of $\mathbb{R}^{3000\times3000}$ones matrix multiplication operation;
- Task2. Solving a least-squares system of random $\mathbf{A} = \mathbb{R}^{3000\times3000}$, $\mathbf{b} = \mathbb{R}^{3000\times1}$;
- Task3. Computing the eigenvalues/vectors of random matrix size $\mathbb{R}^{3000\times3000}$.

Device Usages:

Numpy - Python $\Rightarrow$ 31% CPU usages, uses the Intel's rubbish efficient cores, 100 watts+ power withdraw (first 6 core has hyper-threading) (CPU throttling is a limiting factor).

Pytorch-GPU - Python $\Rightarrow$ Spikes of 50% usages, hard to measure instant power withdraw, too easy task for GPU. Math3.linear - Java $\Rightarrow$ very low of 1.4% CPU usages, 20.8 watts power withdraw (first 6 core has hyper-threading), not far from idle power withdraw (CPU throttling is NOT a limiting factor).

LinearAlgebra - Julia $\Rightarrow$ 5% CPU usages, only uses single thread (first 6 core has hyper-threading), but average around 75 watts power withdraw (CPU throttling is a limiting factor)

The Table 9 records in seconds (reserve three decimal places for enough accuracy to visualize the speediness of GPU computations) the time used by the above three programming languages to perform three distinct tasks. And there are indications of computing complexity in Floating Point Operations (FLO) and average computing speeds of methods in Tera Floating Point Operations per Second (TFLOPs: 10^{12} FLOPs).

Task 1 is 100 matrix multiplication operations of 3000 $\times$ 3000 matrices. For each row vector in 300 rows in matrix $\mathbf{A}$, we need to dot product with each column vector in matrix $\mathbf{B}$, each time of the row vector times the column vector takes n dot products and $n-1$ additions. For n number

Time / Package	Numpy - Python	Pytorch-GPU - Python	LinearAlgebra - Julia	Math3.linear - Java	FLO needed
Task 1	16.33	0.002	25.76	15658.15	5.4×10^{12}
Task 2	9.77	0.03	18.51	100.74	5.4×10^{10}
Task 3	16.79	1.18	20.63	1509.54	8.3×10^{11}
Average	0.13 TFLOPs	900.95 TFLOPs	0.08 TFLOPs	0.00 TFLOPs	2.1×10^{12}

Table 9: Time taken for packages to run the 3 tasks, and average TFLOPs

of rows in matrix $\mathbf{A}$, we need to dot product it by n columns in matrix $\mathbf{B}$, that is n^2. Each of the operations takes $n^2(2n-1)$ floating point operations. By substituting 3000 for n and multiply the result by 100, it is estimated that Task 1 takes 5.4×10^{12} floating point operations.

Task 2 is solving linear system with a square matrix. One method is QR factorization by first rewriting normal equations $A^TAx = A^Tb$ using QR factorization $A = QR$:

$$A^TAx = A^Tb$$
$$R^TQ^TQRx = R^TQ^Tb$$
$$R^TRx = R^TQ^Tb$$
$$Rx = Q^Tb$$

For $m \times n$ matrix QR this takes $2mn^2$ flops. Then forming $d = Q^Tb$ takes $2mn$ flops and solving $Rx = d$ by backward substitution takes another n^2 flops. The total complexity is the summation of the operations above, and for large m, n is approximately $2mn^2$ flops . By substituting 3000 for n and m, it is estimated that Task 2 takes approximately 5.4×10^{10} floating point operations.

Task 3 is computing the eigenvalue and eigenvector of a matrix of size 3000×3000. Since a Symmetric QR algorithm is used and the original matrix is automatically ensured to be symmetric, and hence the upper Hessenberg matrix is also symmetric, thus tridiagonal; this procedure requires approximately $\frac{4}{3}\times n^3\times\log_2(n)$ floating point operations per iteration. We also need to do a second iteration to make it robust and efficient. Therefore, we get $2 \times \frac{4}{3}\times n^3\times\log_2(n) = 8.3\times10^{11}$ floating point operations for a real matrix of size 3000×3000 by substituting n for 3000.

(b) Which do you consider to be the winner in running time?

In Table 9, it is clear that Pytorch-GPU took the least amount of time for each task, using GPU calculating numbers are simply fast, especially with some optimization in Nvidia's Compute Unified Device Architecture (CUDA) to boost Matrix Multiplication speed. Numpy and Julia on CPU also took relatively less time, very fast (0.0843 and 0.1285 TFLOPs). And, especially Julia is only running in single thread, compare to Numpy uses lots of threads, in more advanced packages and systems, Julia have a true potential to be faster. Java took a shockingly long time (only 0.0005 TFLOPs), especially in Task 1, which is almost a 1000 times as long as numpy, and 7.83 billion times slower than Pytorch-GPU. It is clear to see that Java is very slow in matrix multiplications, compare to CUDA with optimizations.

(c) Which do you consider to be the winner in programming convenience? What is your final recommendation? Are there any 'DOs' or 'DON'Ts' ?

In programming in each of the three languages, we found both Python and Julia to be more convenient than Java for programming, with clean syntax and high ease of use as well as outstanding speed. Python also has a more mature development environment and a richer ecosystem.

The final recommendation we can give after all these experiments and data processing is that when very tedious computational tasks are required, both Julia and python are good choices, well suited for scientific computing. On the other hand, try not to program in java. And as you can see from the fact that we used two different libraries, python may has more possibilities than its peers.

(6b) Linear data fitting

(a) Discuss: to what extent have you been able to reconstruct the polynomial p from the noisy data?

We generated the dataset in Jupyter Notebook and plotted the scatter-plot. Then the cubic, quadratic, and primary functions were used to fit them respectively by solving for solution $\mathbf{x}$ in QR Decomposition $\mathbf{Rx} = \mathbf{Q^T b}$ by back-substitution. The degree of fit was indicated by calculating the size of Mean Squared Error (MSE), which is the average squared deviations of predicted value from the truth value (Punishes large deviations, reward small deviations). The smaller the MSE, the better the fit. As the figure 6 shows. MSE $= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

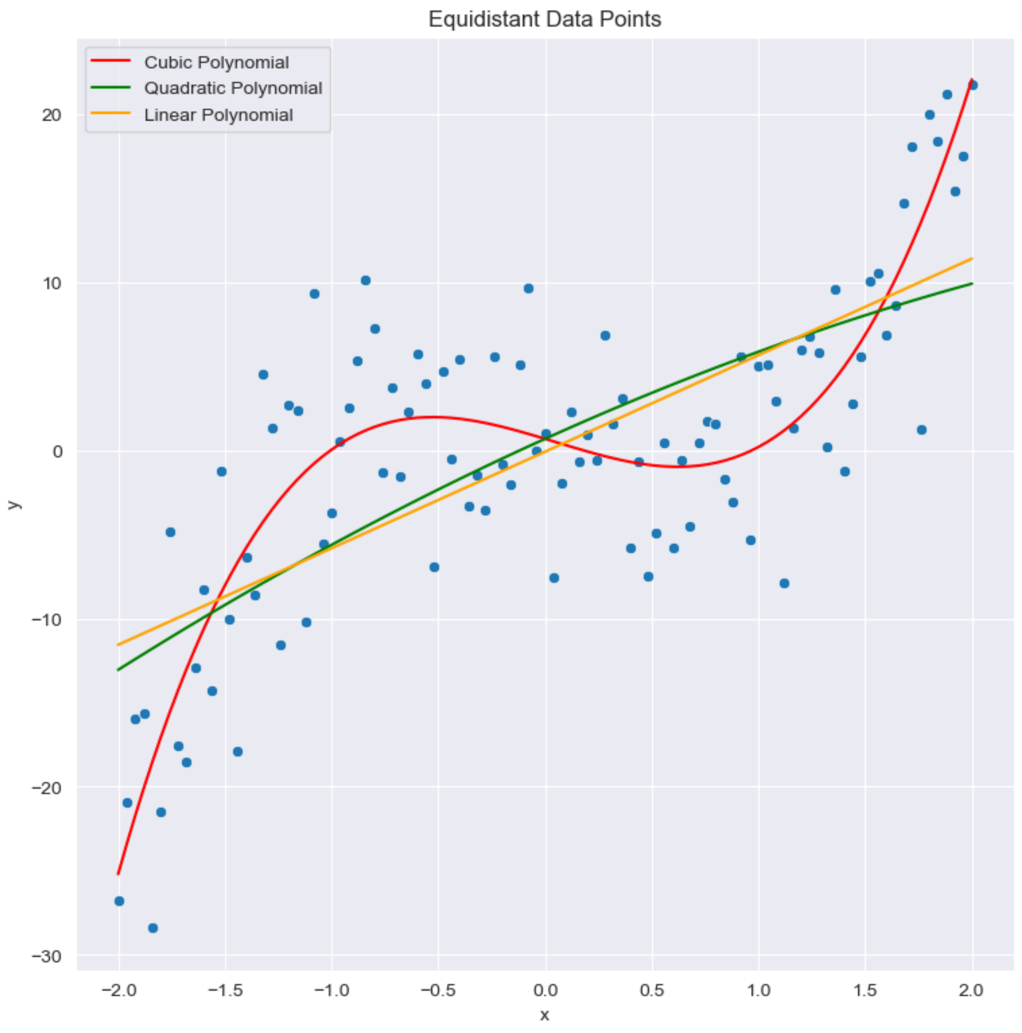


Figure 6:

Due to the presence of random constants, we executed the code ten times and recorded the (MSE) of each of the three functions in order to increase the persuasiveness of the data and to reduce the effect of random constants. We obtained the following table in figure 8 for each of the three functions (reserve two decimals).

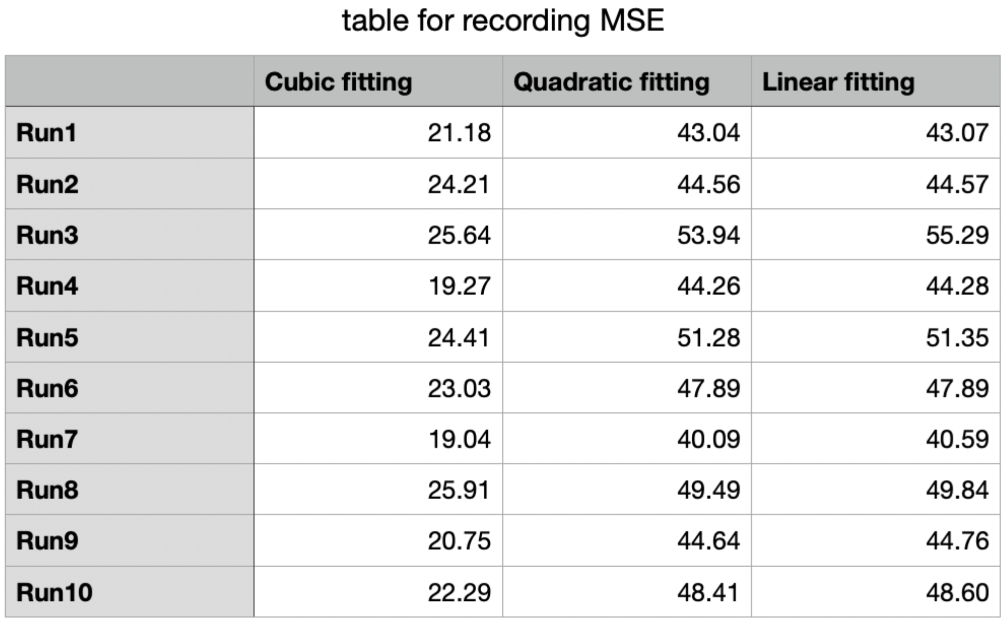


table for recording MSE

	Cubic fitting	Quadratic fitting	Linear fitting
Run1	21.18	43.04	43.07
Run2	24.21	44.56	44.57
Run3	25.64	53.94	55.29
Run4	19.27	44.26	44.28
Run5	24.41	51.28	51.35
Run6	23.03	47.89	47.89
Run7	19.04	40.09	40.59
Run8	25.91	49.49	49.84
Run9	20.75	44.64	44.76
Run10	22.29	48.41	48.60

Figure 7: MSE of 10 runs of 3 different fittings respectively

Figure 8:

The average value of the MSE for the ten fits of the cubic function is 22.573, which is the best fit of the three functions, but its MSE is still large. We think it is because the standard deviation of the random constant η_ι is 5, the randomness is larger, resulting in the generation of data points to be more dispersed. This randomness causes the direct summary of the linear pattern will be very inaccurate. It is expected to be much better, if we use of non-linear regression such as Random Forest/Decision Tree to fit the data points.

(b) Also try to fit with a quadratic and linear polynomial. Do you think the results are logical, and as expected?

On the other hand, as can be seen from the Figure 6, the effect of using a quadratic function as well as a primary function to fit is that the error is very significant, and the ten MSEs' average are 46.76 and 47.024 respectively (See Figure 8), which is also too large, but quite similar in between. This is because the criteria we generated the points has 3 degrees of freedom (4 independent variable - 1 dependent variable = 3), and the quadratic fit only has 2 degrees of freedom (3 - 1), the linear fit is one lesser with only 1 degree of freedom, but the cubic fit has the same degrees of freedom (4 - 1), so it can capture all underlying pattern very well except the random y-component generated. There does not exist enough degrees of freedom to capture all underlying patterns in the data, so the quadratic and linear had similar bad results, and quadratic is still almost always a little bit better (probably because the random generated y-component, quadratic is not strictly better than linear) with 1 extra degree of freedom. We can confirm the result using residual plot in Figure 9,

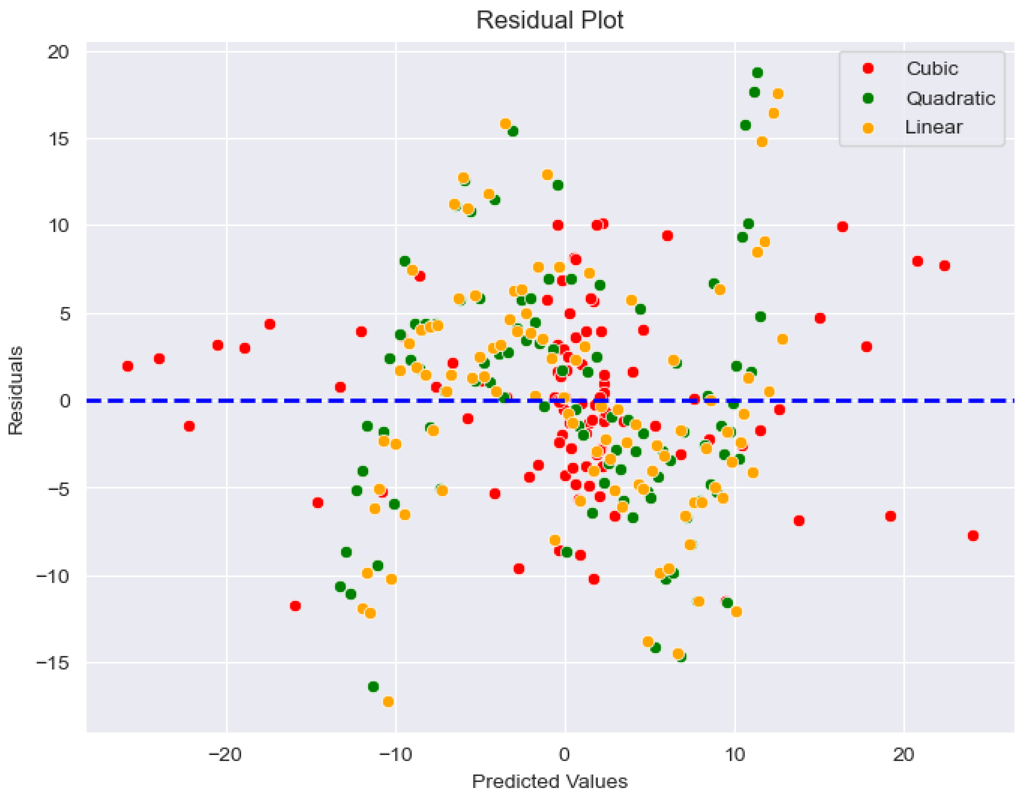


Figure 9: Residual Plot to visualize the ability of regressions to capture underlying pattern

the cubic residual is randomly scattered around both side of the axis, but quadratic and linear fit both demonstrate there exists a cubic oscillation in the result that is not being captured. These are both expected and logical results.