League of Legends Prediction, A Statistical Approach

Brian Zu

League of Legends Prediction, A Statistical Approach

League of Legends (LOL) is a competitive team game and it have been popular for more than 10 years. It contains 5 players per team, competing against another team of 5. The main goal is to destroy the enemy team's Nexus (the base of a team). Each player is in charge of one distinct character that's picked before the game start called the banning and picking section (BP). The standard game is played on a map called "Summoner's Rift," which has three lanes (top, middle, and bottom), a jungle area with neutral monsters, and bases for each team at opposite corners of the map. Each match generally takes from 30 to 40 minutes.

I first started playing this game in 2019, and mainly focused on playing the top lane. In competitive games, such as LOL, even though the game mechanism remains mainly the same in it's 14 years history, there are still significant differences in my gaming experience. This is mainly due to the competitive nature, which incentivizes players all around the world to dig in to subtle changes. For example, a slight change in damage or health points (HP) might affect the strength of a character and subsequently impacting the character choice in BP section. In fact, I have played 856 matches in total over the past 5 years.

Recalling to my gaming experience, I found out that many players tends to give up when their side is in an inferior position. In other times, other players held on straight to the end of the game, sometimes turning over. This prompts me to think: to what extent is the game result predictable at the middle of the game with so many variables?

In this work, we collected many rounds of matches that contains the first 10-minute game data. The data is 38 by 9879, where 38 is the amount of data per match, and 9879 is the number of match we collected. This is a very large dataset, and the data has a high dimension which is hard to handle. Thus, we need to compress (reduce the dimension) of the dataset so that it's easier for processing. The technique is Principal Component Analysis (PCA), where we reduce the dimension by finding a vector and map all the high dimension points onto it's direction, or lining up. Basically, for a 2D to 1D example, it means a series of point on 2D plane are all compressed onto a same line. An illustration is available later in fig 5. In this process, information loss exists because we are carrying out a compression task and the dataset is smaller. Here, the aim of PCA is to minimize this loss while compressing. Specifically, we want to know how dispersed the points are on the axis after mapping, and maximize it to show the maximum information maintained. The level of dispersion can be measured by variance. We used Lagrange multiplier method (Yadolah, 2008) for this optimization task. After this, the data is reduced from 38 dimension to 2 dimension. Lastly, we used logistic regression to find the relationship of the data with the game trend, trying to predict the game winning side with the first 10-min processed data. The python code is available in appendix B. Here is a brief illustration of this work:

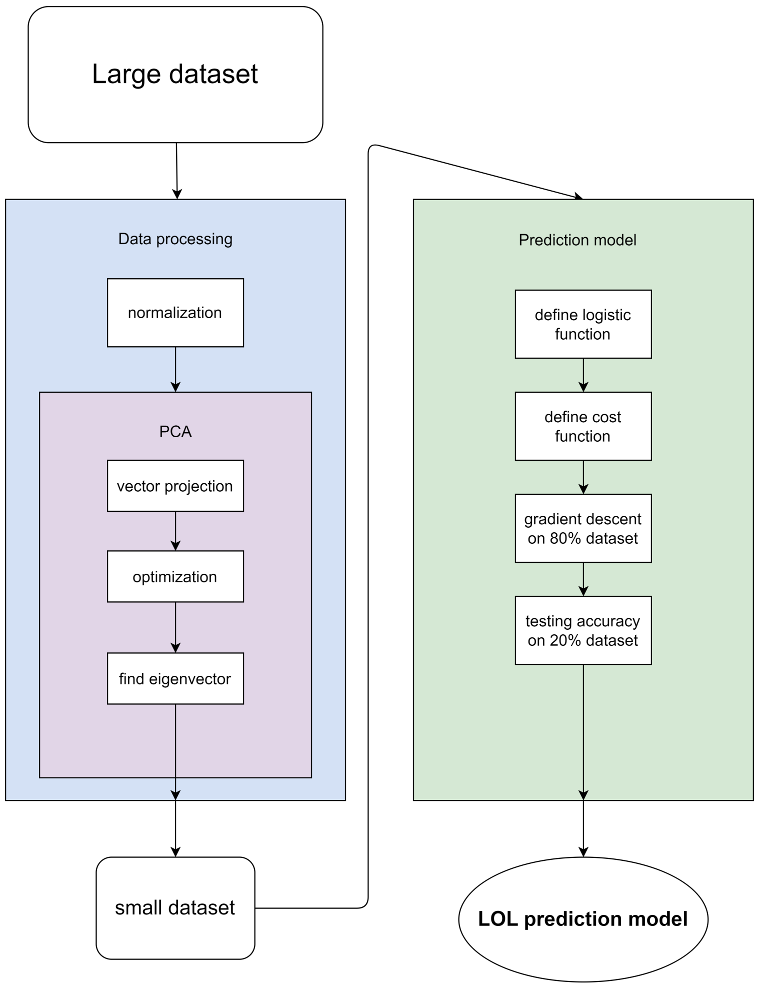

Figure 1

Overview flowchart

Data

Thank to the Riot API (Games, n.d.), I can access to many game data easily. The data I collected is from LOL ranked games at diamond stage (a stage where most advanced players are at). It contains 9879 games in total and each game with 38 features and one label. Since the whole dataset cannot fit for display, in Table 1, a 10×5 data snippet is selected from the dataset. The first column, titled "blue wins", contains binary values (only 0 and 1) that indicates whether blue team in this match has eventually won or not. This is called the label for the model that later will be used. Other columns, such as "blue wards placed", "blue first blood" describes the status of teams at the time of 10 minutes. These will be the features, or inputs, of the model.

Visualizing data

For better understanding of the data, it can be visualized before diving into modeling.

Currently, the dataset contains 38 features which is large to handle. I need to determine whether they are all useful to the prediction task. Plotting the relationship between features can reveal their usefulness.

First, let n be the size of a feature. The correlation between two features of length n can be calculated through the Pearson correlation coefficient (Pearson, 1895).

$$r = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum(X_i - \bar{X})^2 \sum(Y_i - \bar{Y})^2}} \tag{1}$$

The Pearson correlation coefficient is developed by Karl Pearson in 1895. It measures the linear correlation between two sets of data, and gives a normalized data, ranging from -1 (perfectly negative correlation) to 1 (perfectly positive correlation). By calculating for each pair of features, we can plot a correlation heat map indicating the correlation between each pair of data. In the formula, X_i one data point in column X, $\bar{X}$ is the mean and the same with column Y

Since a 38×38 grid is too large for display, I selected 9 features for plotting. In Figure 2, there is a strong negative correlation between "redFirstBlood" and "blueFirstBlood". This is because in each match there can be only one first blood, either from blue team or red, indicating one of the first blood column is redundant. The "blueGoldPerMin" has a strong positive

Table 1

A snippet of the match data collected

blue wins	blue wards placed	blue wards de-stroyed	blue first blood	blue kills
0	28	2	1	9
0	12	1	0	5
0	15	0	0	7
0	43	1	0	4
0	75	4	0	6
1	18	0	0	5
1	18	3	1	7
0	16	2	0	5
0	16	3	0	7
1	13	1	1	4
⋮	⋮	⋮	⋮	⋮

correlation with "blueKills". This is also reasonable because killing an enemy once will result in gold award. These results shows the dataset is unclean which needs further processing.

Reducing the dimension (the number of features) of the dataset is important for the modeling process because it makes the model more explainable.

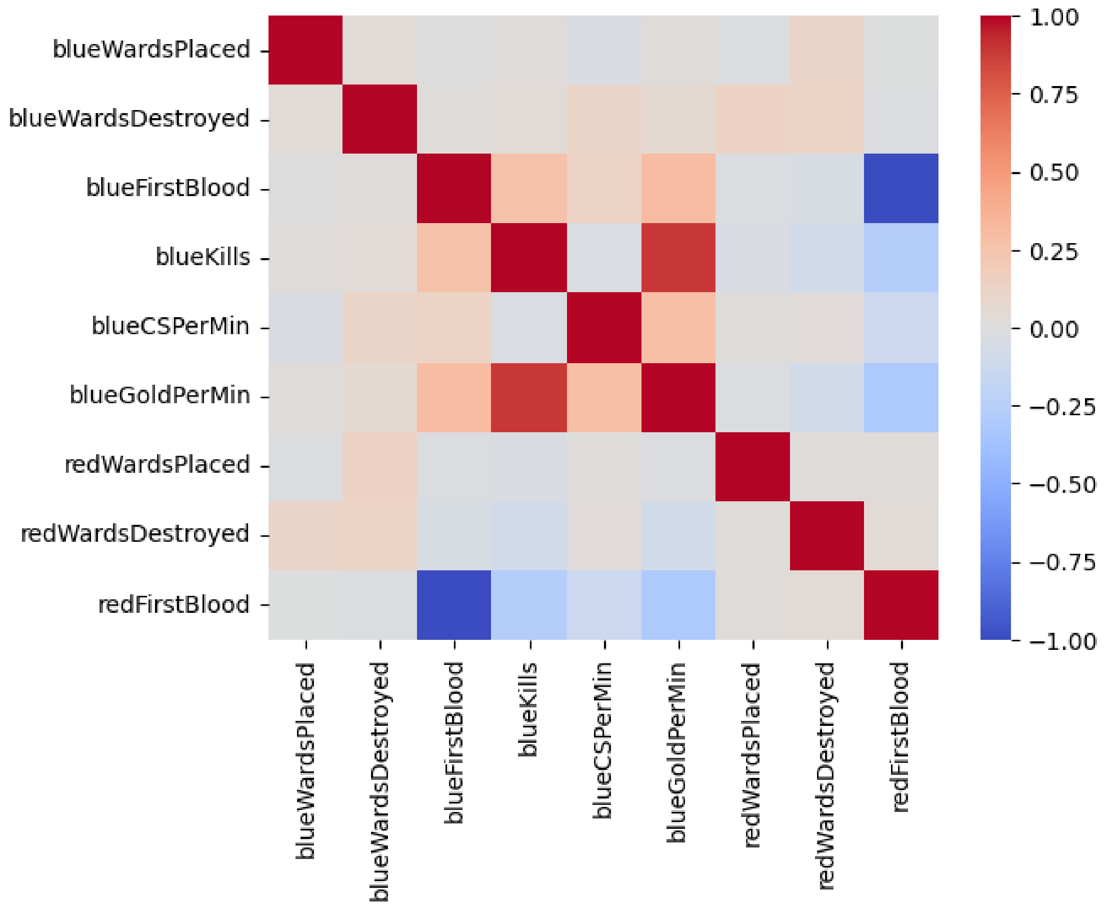

Figure 2

Feature correlation heat map

Data reprocessing

Normalization

Before reducing the dimension of data, it's necessary to normalize it first. It could be observed from Table 1 that each column are having different ranges. For example the "blue wards placed" has data ranging from 12 to 75 while "blue wards destroyed" only has data ranging from 0 to 4 in the snippet (the range isn't correct for the whole dataset but it's enough for showing the difference between columns). This potentially implies one column with larger values is more important than other columns, which might weaken the accuracy and robustness of the model. Thus we need to normalize the data.

The standard score of a column can be calculated by:

$$x_i = \frac{X_i - \bar{X}}{\sigma} \tag{2}$$

where x_i is the new data scaled, $\bar{X}$ is the mean of the column, and σ is the standard deviation of the column. This transforms the data to have a mean of 0 and a standard deviation of 1.

Principal Component Analysis

The most straightforward approach to reduce dimension of data is to remove one of the columns in the strongly correlated pairs. However, this cannot indicate the amount of information lost from our reduction. A more common approach is to use Principal Component Analysis (PCA).

PCA was developed in 1901, also by Karl Pearson (Pearson, 1901). It's a method to reduce the number of variables in a dataset while keeping the most information.

Vector projection

From fig 2 we can see the "blueGoldPerMin" and "blueKills" has strong positive correlation. Here, I'll take these 2 features from the dataset and select only 5 data points from it for illustrating the steps for PCA. For example the column "blue kills" and "blue gold per min":

We need to find a direction where all of the points will be projected on so that all the 2D points become on one single line, making it 1D. First, for the ease of computation, we express each data point using vectors:

$$x_i = \begin{bmatrix} 0.46 \\ 0.94 \end{bmatrix} \tag{3}$$

where x_i represent one data point, 1791 the "blueGoldPerMin" and 9 "blueKills".

Next, the projection of the vector x_i onto another direction, represented by a unit vector u with a length of 1 is illustrated in fig 4. It's mathematically defined:

$$\frac{|x_i| \cdot \cos\theta}{|u|} \tag{4}$$

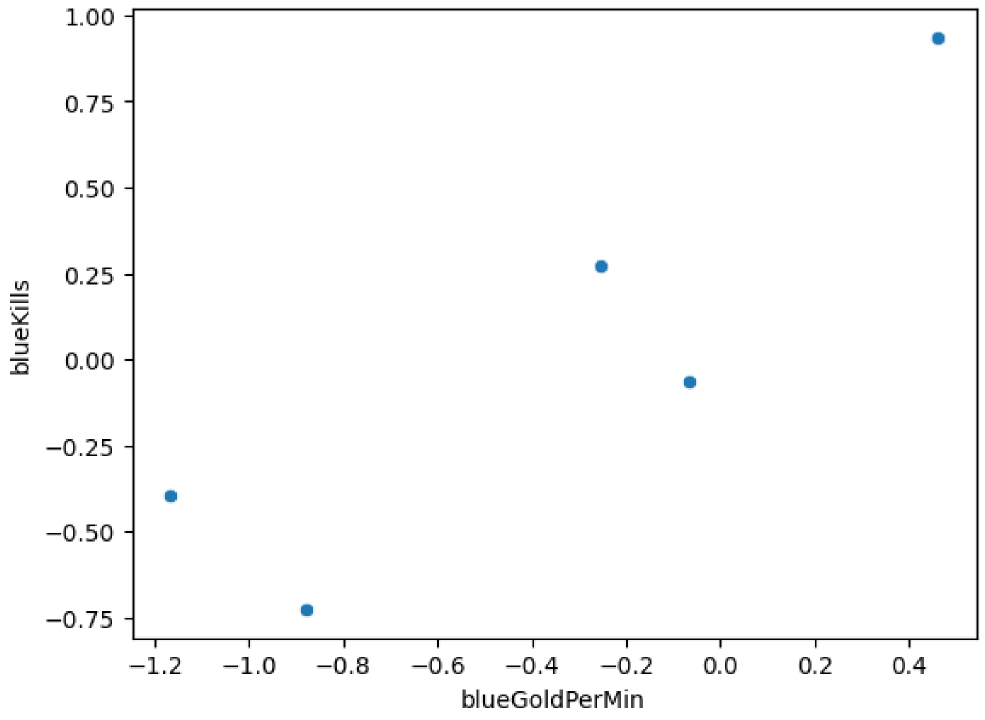

Figure 3

Scatter plot of 2 features

Because u is a unit vector of length 1, this means $|u| = 1$, so we can simplify the equation 4 into:

$$|x_i| \cdot \cos\theta = |x_i| \cdot 1 \cdot \cos\theta = |x_i||u| \cdot \cos\theta \tag{5}$$

The benefit of defining u a unit vector is that by equation 5, the projection is equal to the dot product of x_i and u.

$$|x_i||u| \cdot \cos\theta = x_i \cdot u = x_i^\top u \tag{6}$$

For every data point, we can apply this projection to make all the points lying on the unit vector's vector space, and thus making them 1D.

Fig 5 shows the original points (blue) have projected points (green) on the direction of the unit vector's vector space (red). With the process above, we can obtain a new dataset X_2

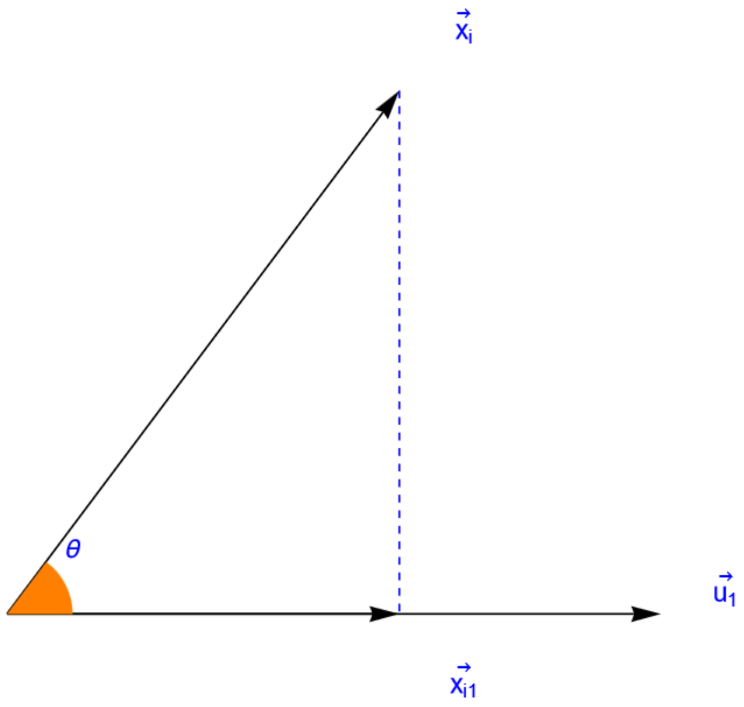

Figure 4

Vector projection

Vector matrix multiplication

In linear algebra, matrix A times vector v can be interpreted as the vector v undergoing a transformation defined by the matrix A.

For example, here we have two base vectors defined:

$$i = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \ j = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{7}$$

These two vectors spans the 2D Cartesian plane because for any vector v on it, it can be represented by a linear combination of the two base vectors:

$$v = \begin{bmatrix} 2 \\ 4 \end{bmatrix} = a \begin{bmatrix} 1 \\ 0 \end{bmatrix} + b \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 4 \end{bmatrix} \tag{8}$$

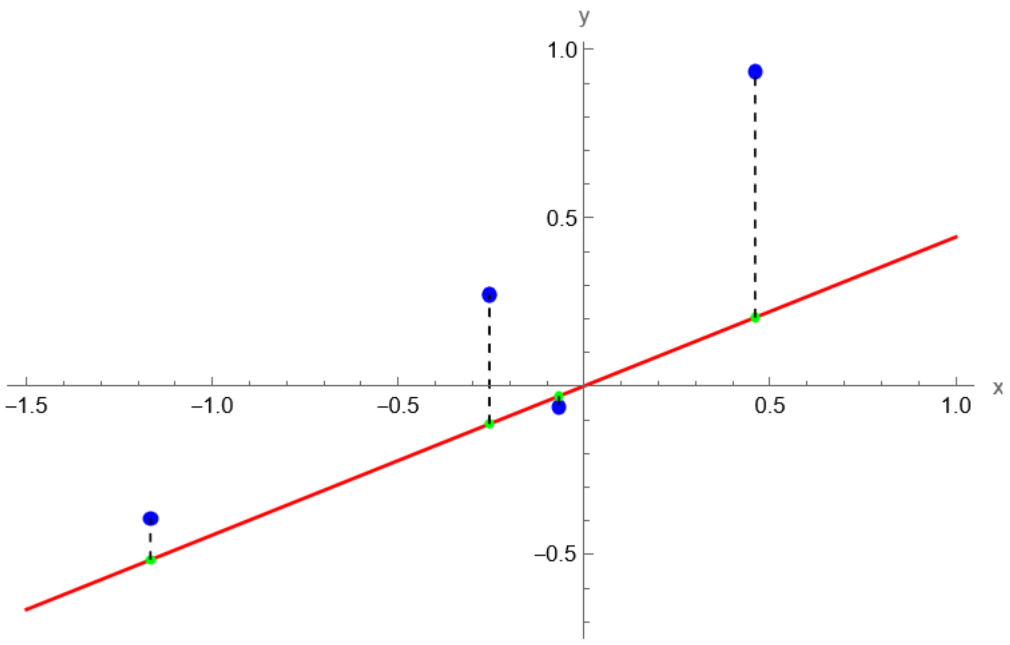

Figure 5

Projecting all the points

For the matrix A:

$$A = \begin{bmatrix} 2 & 4 \\ 1 & 3 \end{bmatrix} \tag{9}$$

It means that the base vectors have transformed from equation 7 to:

$$i = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \ j = \begin{bmatrix} 4 \\ 3 \end{bmatrix} \tag{10}$$

Similar to the combination in equation 8, the new vector after the base has changed is now becoming:

$$\hat{v} = 2 \begin{bmatrix} 2 \\ 1 \end{bmatrix} + 4 \begin{bmatrix} 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \cdot 2 + 4 \cdot 4 \\ 1 \cdot 2 + 3 \cdot 4 \end{bmatrix} = \begin{bmatrix} 8 \\ 13 \end{bmatrix} = Av \tag{11}$$

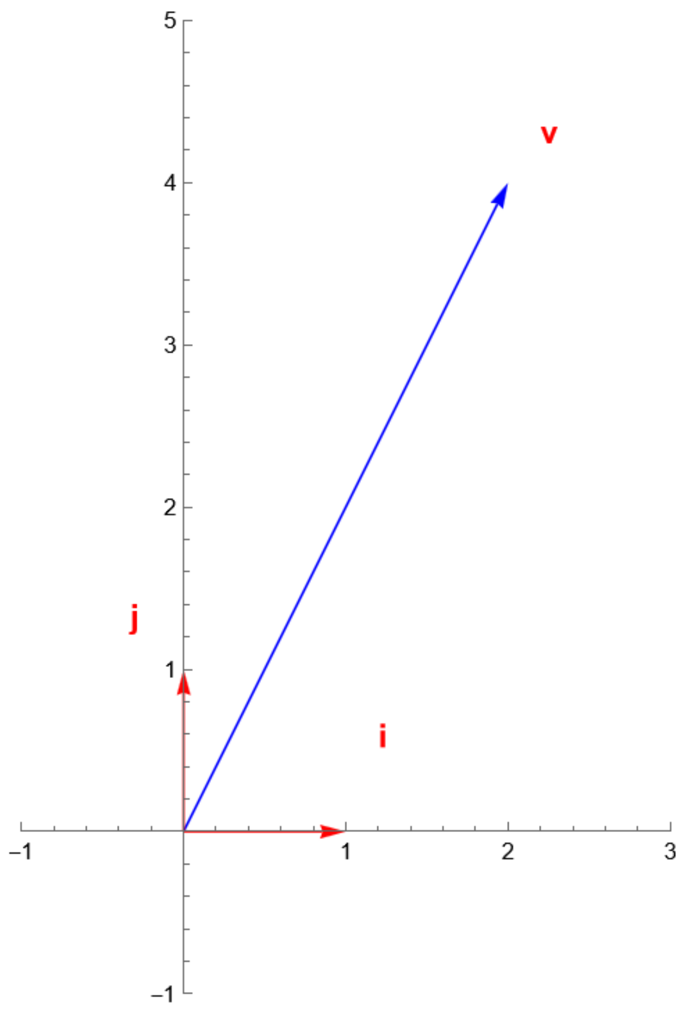

Figure 6

Vector v

Expressing variance

Since the aim of PCA is to reduce the dimension of data while maximising the remaining information, this means the green projected points on fig 5 should be as dispersed as possible on the new vector space. We can use variance to measure:

$$Var(X_2) = \frac{1}{n-1}\sum_{i=1}^{n}(x_{i2} - \bar{X_2})^2 \tag{12}$$

where $\bar{X_2}$ is the mean of X_2 and x_i2 is the element in X_2 to differentiate it with x_i in original data.

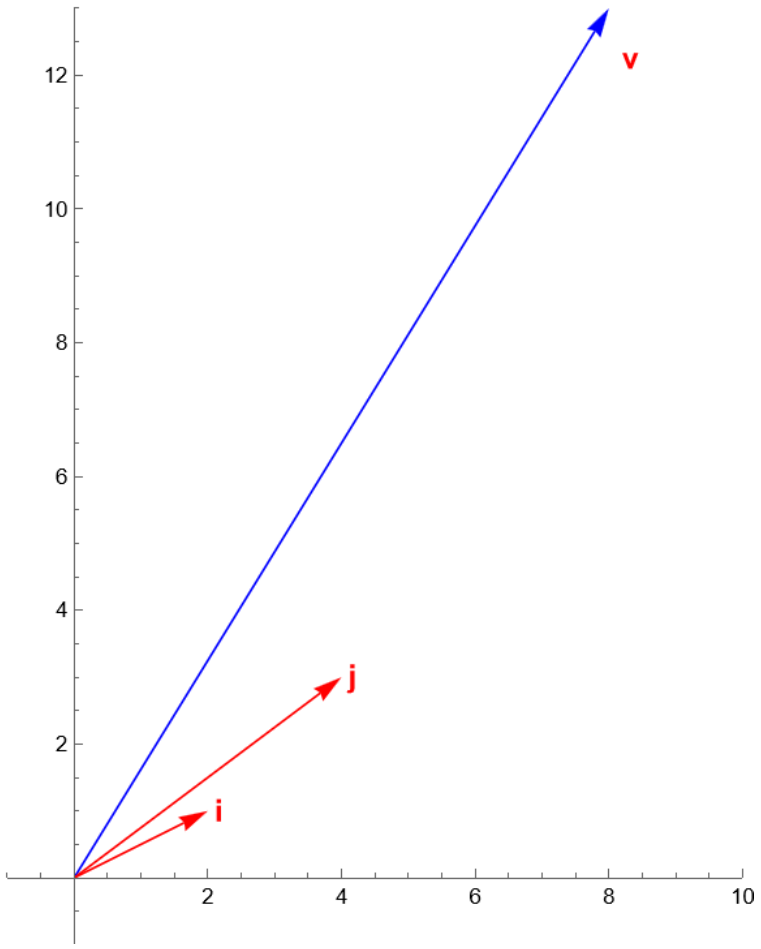

Figure 7

Vector $\hat{v}$

Because we have already normalized the data, and transforming it so that it's mean is 0, meaning $\bar{X}_2 = 0$. Given that the x_i2 in X_2 is actually calculated above by the projection formula 6. We can transform the equation 12 into:

$$Var(X_2) = \frac{1}{n-1}\sum_{i=1}^{n}(x_{i2})^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i^\top u)^2 \tag{13}$$

Notice that by expanding $x_i^\top$ it's actually i column vectors x_i that times u as a row vector. This is the same as the definition of matrix vector multiplication.

$$\sum (x_i^\top u)^2 = (\sum \begin{bmatrix} x_{i1} & x_{i2} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix})^2 \tag{14}$$

$$= (Xu)^\top Xu = X^\top u^\top Xu \tag{15}$$

Substituting this to the formula 13:

$$Var(X_2) = \frac{1}{n-1} X^\top u^\top Xu \tag{16}$$

We can group the matrix terms together:

$$Var(X_2) = u^\top (\frac{1}{n-1} X^\top X)u \tag{17}$$

Because the specific form of the terms in the brackets are unimportant for calculation later, we can simplify it as one term for simpler expression:

$$C := \frac{1}{n-1} X^\top X \tag{18}$$

Thus, the variation can be simplified:

$$Var(X_2) = u^\top Cu \tag{19}$$

We can express this in a function form:

$$f(u) = u^\top u \tag{20}$$

Maximum variance

To maximise the variance, we can find the gradient of function 20. Moreover, since we defined u as a unit vector, and equation 5 is based on this definition, the variable u for the function $f(u)$ has a restriction. The following equation describes the task:

$$\max_u u^\top Cu \tag{21}$$

$$\text{subject to } u^\top u = 1. \tag{22}$$

This is a standard optimization problem wit constraints. Without the constraint, we can simply find the u where the gradient of $f(u)$ is equal to 0. However, given the constraint, we need to use the Lagrange multiplier method.

Lagrange multiplier method

Given a function $f(x)$ that needs to be maximised, subject to a constraint $g(x) = c$, where c is a constant. The Lagrangian function is defined:

$$\mathscr{L}(x,\lambda) = f(x) - \lambda(c - g(x)) \tag{23}$$

where λ is a unknown scalar. The term $\lambda(c - g(x))$ can be considered a penalty for the function. In order to minimize the gradient, $c = g(x)$ will mean the constraint is satisfied.

Applying this to our situation, we can set:

$$\mathscr{L}(u,\lambda) = f(u) - \lambda(u^\top u - 1) \tag{24}$$

$$= \mathscr{L}(u,\lambda) = u^\top Cu - \lambda(u^\top u - 1) \tag{25}$$

$$= \mathscr{L}(u,\lambda) = u^\top Cu + \lambda u^\top u + \lambda \tag{26}$$

The function becomes without any constraints, so we can simply maximize it by finding the gradient. For this 2 variable function, the gradient is partial derivative, Because we are aiming to find the u, which is the direction where the vectors are mapped are having the greatest variance, we need to find the partial derivative of $\mathscr{L}$ with respect to u.

$$\frac{\partial f}{\partial u} = 2Cu - 2\lambda u = 0 \tag{27}$$

$$Cu = \lambda u \tag{28}$$

However, the derivative only tells the rate of change, when set to 0, it's a stationary point. We aim to maximize this function, so this stationary point needs to be the maximum point, and thus the function needs to be convex. The detailed proof is in appendix section A.

Simplifying eigenvalue equation

Looking back at equation 28, the left hand side means the base vector has changed according to matrix *C* on vector *u*. This is equal to the right hand side with a scalar λ multiplied by *u*. This means the transformation on the base vector has not impact the direction where vector *u* is pointing, because scalar vector multiplication only change the magnitude of u. Here the vector *u* is the eigenvector, and the scalar multiplier λ is the eigenvalue. We have *C* already known, now trying to find vector *u*.

Here we need to construct a identity matrix *I* where the top-left to bottom-right diagonal is 1 and the rest of the elements are 0.

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{29}$$

Then it multiplies λ will make the diagonal filled with λ

$$\lambda I = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \tag{30}$$

$$\lambda I u = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} u = \begin{bmatrix} \lambda u_1 \\ \lambda u_2 \end{bmatrix} = \lambda u \tag{31}$$

Substituting it into equation 28, we get:

$$Cu = \lambda I u \tag{32}$$

$$(C - \lambda I)u = 0 \tag{33}$$

$$\tag{34}$$

Here $C - \lambda I$ gives a matrix with unknown variable λ, so it's in the form of matrix vector multiplication. Notice that it's special because the result of multiplication is 0, which could be written as a vector of zeros:

$$(C - \lambda I)u = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{35}$$

This indicates that after the transformation represented by $C - \lambda I$, the eigenvector u shrinks to the origin, have a length of 0. To solve this equation 35, we need to have a look at what shrinking the eigenvector to the origin mean.

Using determinants

Determinant for a matrix is, geometrically, a scalar value that gives the area of the parallelogram formed by its column vectors (or the matrix transformed base vectors). We simply denote determinant for a matrix K:

$$K = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \tag{36}$$

$$\det(K) = \det \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \tag{37}$$

Fig 8 illustrate the determinant of K. The parallelogram $ABCD$'s area is the determinant of K where u is the first column $[1,2]$ in K and v is the second column $[2,1]$.

Then the area can be calculated. Draw a perpendicular line h from B to AC which represents the triangle ABC's height. It can be found by using formula for point to a line. By finding the length of AC or the length $|v|$ we can thus calculate the area of triangle ABC. Two times the area of ABC is the area of the parallelogram $ABCD$.

$$h = \frac{|2 \cdot 2 - 1 \cdot 1|}{\sqrt{1^2 + 2^2}} \tag{38}$$

$$= \frac{3}{\sqrt{5}} \tag{39}$$

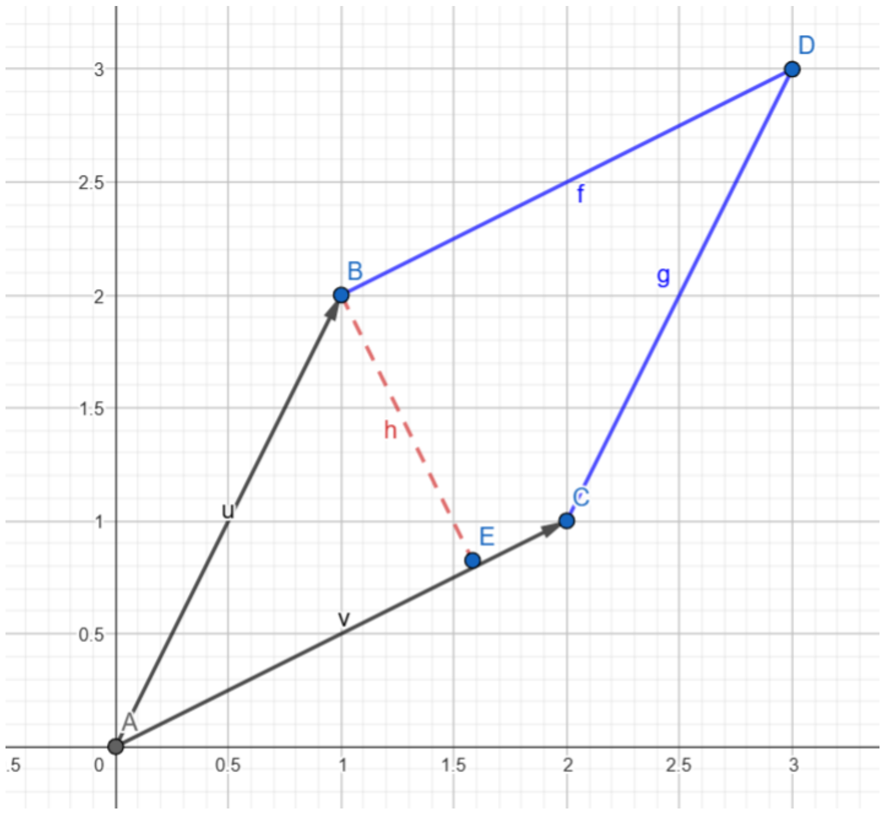

Figure 8

Illustration of determinant of A

Finding the length of vector v:

$$|v| = \sqrt{1^2+2^2} = \sqrt{5} \tag{40}$$

Lastly, find the determinant or the area of the parallelogram:

$$S_{ABCD} = 2 \cdot S_{ABC} = 2 \cdot \frac{1}{2}h|v| \tag{41}$$

$$= h|v| = \frac{3}{\sqrt{5}} \cdot \sqrt{5} = 3 \tag{42}$$

We can generalize this process by making $B = (c,d)$, $C = (a,b)$. This represent a matrix K:

$$K = \begin{bmatrix} c & a \\ d & b \end{bmatrix} \tag{43}$$

Then the determinant of K can be calculated in the same process:

$$h = \frac{|bc - ad|}{\sqrt{b^2 + a^2}} \tag{44}$$

$$|v| = \sqrt{b^2 + a^2} \tag{45}$$

$$S_{ABCD} = |bc - ad| \tag{46}$$

However, noting that the area in the determinant is signed, so the absolute value needs to be removed, giving a conclusion when K is 2 by 2:

$$\det(K) = bc - ad \tag{47}$$

This concept also applies to other dimensions of the matrix, but for simplicity in illustrating this process, I chose the dimension of 2.

Finding eigenvalue

Finally, we can find the eigenvalue. Starting with the conclusion in section , where the transformed eigenvalue shrinks to 0. Expanding the formula 35:

$$\left(\begin{bmatrix} c_1 & c_2 \\ c_3 & c_4 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{48}$$

Multiplying the scalar λ into the identity matrix I gives:

$$\left(\begin{bmatrix} c_1 & c_2 \\ c_3 & c_4 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \right) \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{49}$$

Matrix subtraction is simply subtracting each element with corresponding position. Thus the terms in the bracket can be simplified:

$$\begin{bmatrix} c_1 - \lambda & c_2 \\ c_3 & c_4 - \lambda \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{50}$$

From the conclusion in sub section , we can conclude that the determinant of the matrix applied to the eigenvector is 0. This is because in section , we concluded that the eigenvector shrunk to the origin. This means the matrix must have a determinant of 0 because a zero determinant means the transformation collapses the space into a lower dimension, like flattening a 2D object into a point.

$$\det\left(\begin{bmatrix} c_1-\lambda & c_2 \\ c_3 & c_4-\lambda \end{bmatrix}\right)=0 \tag{51}$$

Applying the conclusion from sub subsection :

$$(c_1-\lambda)(c_4-\lambda)-c_2c_3=0 \tag{52}$$

$$\lambda^2-(c_4+c_1)\lambda+c_1c_4-c_2c_3=0 \tag{53}$$

Given that C is a matrix with known numbers, the only unknown variable is λ, which is the eigenvalue, and it can be simply found by solving this quadratic equation.

$$\lambda=\frac{(c_4+c_1)\pm\sqrt{(c_4-c_1)^2+4(c_1c_4-c_2c_3)}}{2} \tag{54}$$

Then we can know the matrix in equation 50. By matrix vector multiplication, we can construct a linear equation:

$$\begin{cases} (c_1-\lambda)u_1+c_2u_2=0 \\ c_3u_1+(c_4-\lambda)u_2=0 \end{cases} \tag{55}$$

By finding the u_1 and u_2, we can find the eigenvector u, which is the direction where all of the data points will be projected to maximise the variance and to minimise the loss of information. This is the whole process of PCA.

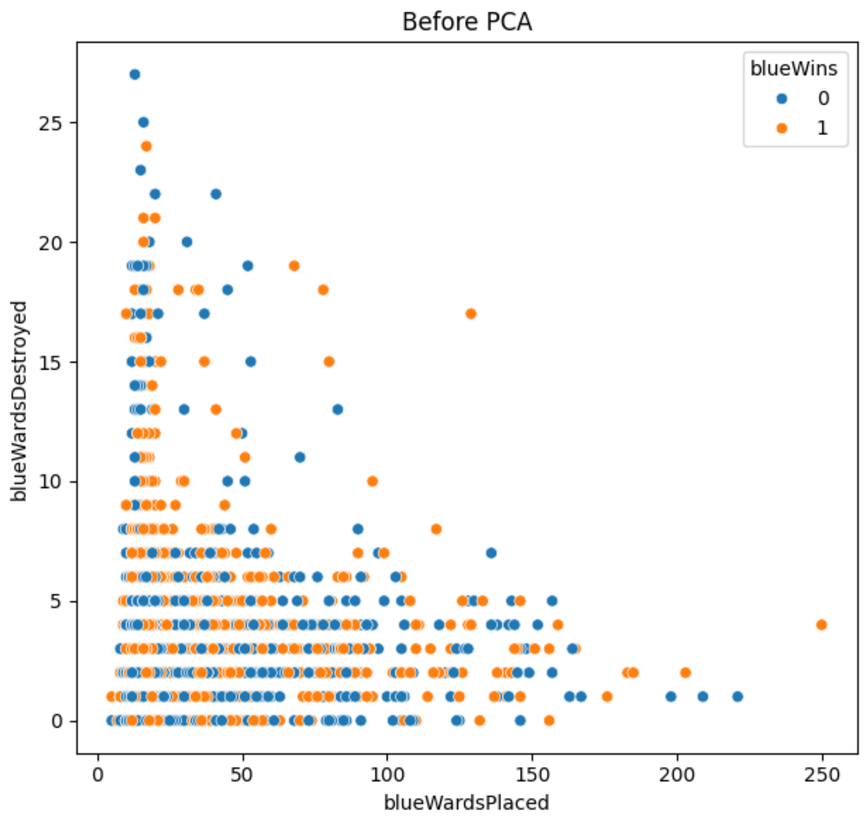

Figure 9

Scatter plot of 2 features before PCA

Applying PCA

Now, we can apply PCA to the original LOL dataset that has 38 features.

Fig 9 shows the scatter plot where orange points suggest blue wins and blue points suggest blue lost. We can compare the scatter plot of 2 features "blueWardsPlaced" and "blueWardsDestroyed" with the scatter plot of 2 features after PCA:

We can obviously see that the distribution of points is more spacial distinguishable with a general trend of orange points on the left and blue points on the right, much more than fig 9 before applying PCA. This shows that the useful information is more condensed in only 2 features, derived from the dataset with 38 features. This allows us to apply our prediction model simpler.

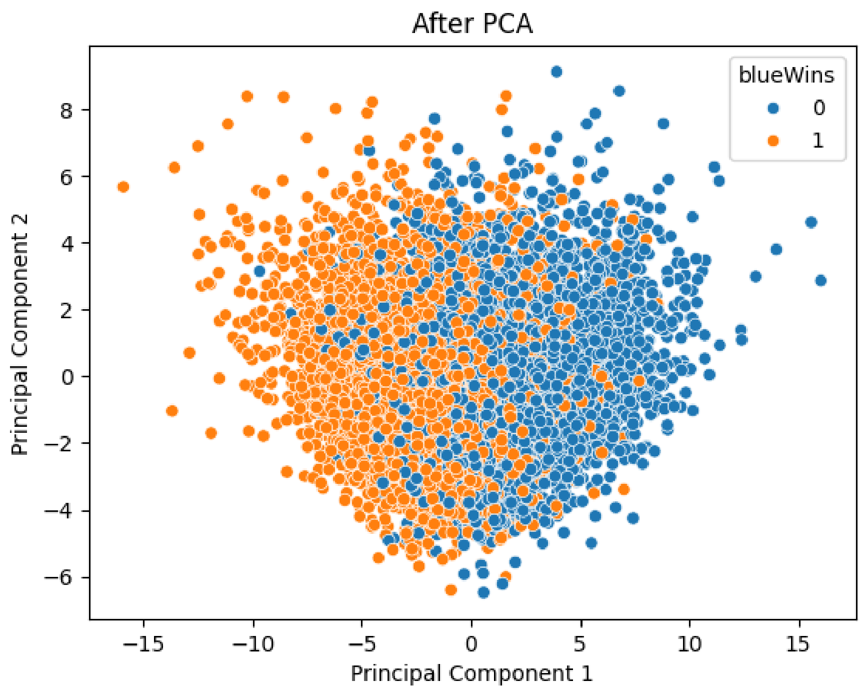

Figure 10

Scatter plot of 2 features after PCA

Prediction

After the data is being processed, we can use models that has the processed features as input and output the prediction. For a binary classification task, logistic regression is well suited.

Logistic function

Logistic regression (Cox, 1958) aims to output a prediction given a set of data. It uses the logistic function in the form:

$$z = Wx + b \tag{56}$$

$$p(z) = \frac{1}{e^{-z}} \tag{57}$$

$$p(x) = \frac{1}{e^{-(Wx+b)}} \tag{58}$$

with W being the matrix of coefficients, x the vector of input and b a vector of bias.

it looks like this:

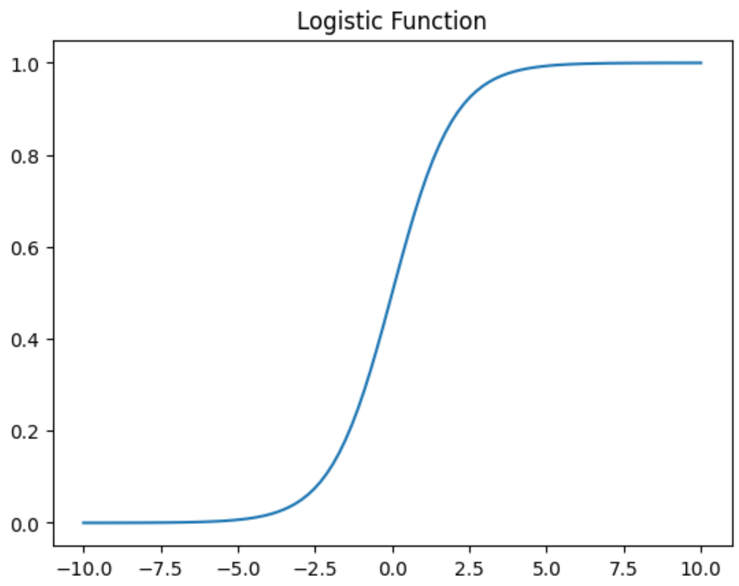

Figure 11

Logistic function

There are some properties of this function which explains why it's well fitted for our classification task:

$$\lim_{z\to\infty}\frac{1}{1+e^{-z}}=1 \tag{59}$$

$$\lim_{z\to-\infty}\frac{1}{1+e^{-z}}=0 \tag{60}$$

This gives the range where $0 < f(x) < 1$, meaning a probability between blue wins (1) and red wins (0).

So we aim to fit this curve for our dataset and reduce its loss.

Evaluating the performance of the model

In order to train our model with the dataset, we need to first evaluate its current performance. We can define a loss function:

$$L(f(x),y) = \begin{cases} -log(f(x)) & y = 1 \\ -log(1-f(x)) & y = 0 \end{cases} \tag{61}$$

where y is the label. In this case y is the column "blueWins", which is the target output.

When $y = 1$, the loss function is $-log(f(x))$. The output of $f(x)$ should be the closer to 1 the possible. Thus the greater $f(x)$ is, the loss function $-log(f(x))$ is smaller, and vice versa as shown in fig 12.

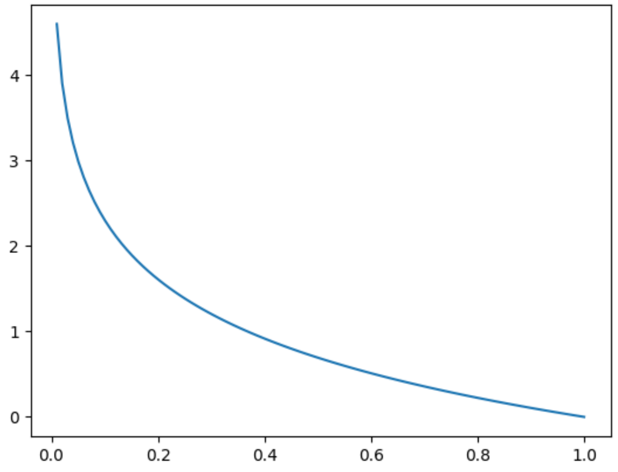

Figure 12

Loss function when $y = 1$

When $y = 0$, the loss function is $-log(1 - f(x))$. The output of $f(x)$ should be the close to 0 the possible. Thus the smaller $f(x)$ is, the loss function $-log(f(x))$ is smaller, and vice versa as shown in fig 13.

We can combine the two separate cost functions into one:

$$L(f(x),y) = -y\log(f(x)) - (1-y)\log(1-f(x)) \tag{62}$$

This is the same as equation 61 because when $y = 0$, the first term $-y\log(f(x)) = 0$, only leaving $-(1-y)\log(1-f(x))$, and when $y = 1$, the second term $-(1-y)\log(1-f(x)) = 0$, only leaving $-y\log(f(x))$

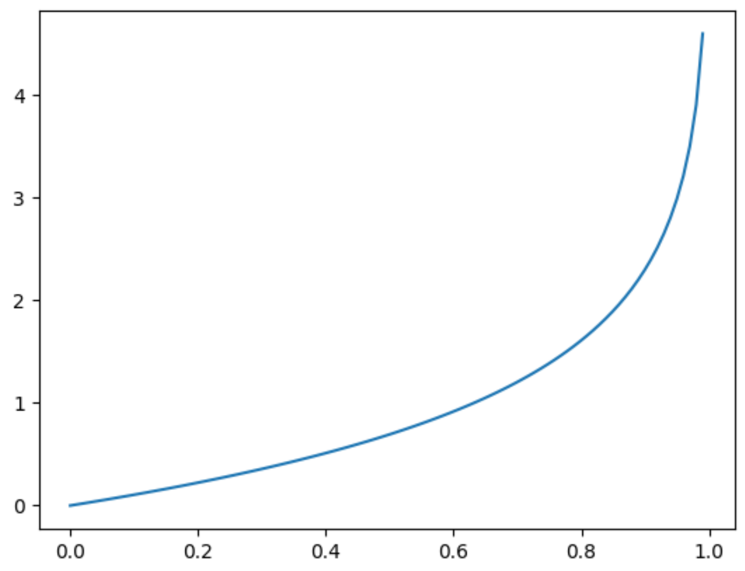

Figure 13

Loss function when $y = 0$

Using MSE (mean squared error), we can denote the total cost on the whole dataset:

$$C(W,b) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(f(x_i)) + (1-y_i)\log(1-f(x_i))] \tag{63}$$

We can simplify the expression by making the last column vector of W to be b

Here we need to optimize the coefficient matrix W to minimize the cost C. We can minimize the partial derivative of C with respect to W to show how the W should change to reduce C.

Gradient descent

The gradient gives the way to update the parameter W to reduce the cost function C. The gradient is:

$$\frac{\partial C}{\partial W} = \frac{1}{m}\sum_{i=1}^{m}(f(x_i) - y_i)x_i \tag{64}$$

Every time W reduce the gradient will result in a lower cost:

$$W_2 = W_1 - \alpha \frac{\partial C}{\partial W} \tag{65}$$

where α is the learning rate, a multiplier that's manually determined for the gradient so that the gradient process can be quicker.

By gradient descent, we can train the logistic model now. However, to accurately evaluate the precision of the model, only 80% of the data will be used, and the other 20% will be used for the validation dataset to ensure that the model is not only working with seen data.

Training model

During gradient descent, the cost of the model reduces in fig 14

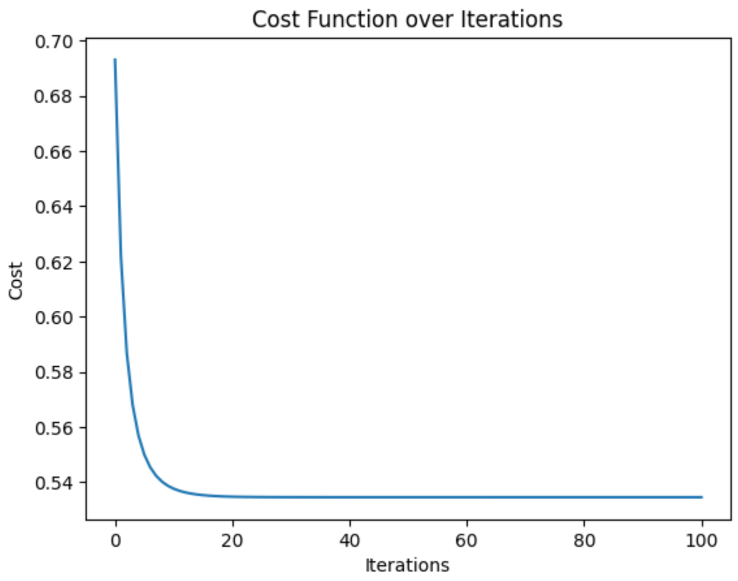

Figure 14

Gradient descent loss

As we can see, after 100 iterations, the cost dropped from about 0.7 to around 0.54, proving the effectiveness of this process. Moreover, the logistic model now can be visualized on the scatter plot:

where the green line represents the decision boundary that the logistic model draws. It thus categorize the points by their position: on the left of the decision boundary the data will be

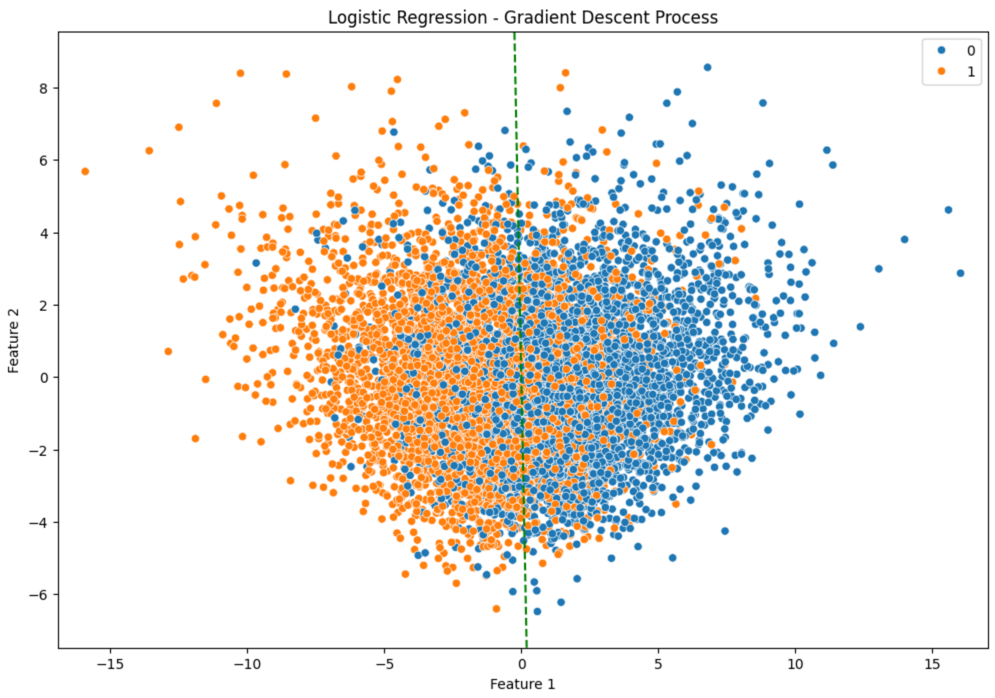

Figure 15

Decision boundary on dataset

categorized as blue wins and vice versa.

Evaluating model accuracy

For binary classification task, we can simply use a probability where the prediction value is equal to the real "blueWins" value. By using the model to predict the validation dataset, we can find its accuracy on unseen dataset:

$$Acc = 72.8744939271255\% \tag{66}$$

Evaluation

In this essay, we explored a statistical approach to predicting the outcomes of League of Legends matches using data collected from the Riot API. The methodology employed involved several key steps: data collection, preprocessing, visualization, dimension reduction using Principal Component Analysis (PCA), and building a logistic regression model for prediction.

Overall, the result is, to a large extent, acceptable because competitive games often have many changes and thousands of variables to consider, including the player's mindset that's not quantifiable. Moreover, the dataset only contains the first 10 minute data to predict the winning side, which only gives a trend and ignores everything that happens after 10 minutes. Maybe it's this unpredictability that gives LOL as a competitive game its charm and can remain popular for over 10 years.

However, there are some limitations in the methodology. First, the dataset focused on games at the diamond skill level, which represents a specific subset of League of Legends players. Expanding the analysis to include a wider range of skill levels could provide insights into how the predictive factors vary across different player segments.

Secondly, although PCA condenses the data so that we can visualize and easily operate it, in there exist information lost when reducing the dimension of the data and though PCA allows for minimum information loss.

Lastly, the PCA compresses the data. Similar to compressing files, the compressed data cannot be interpreted by human. For example, if applying logistic regression, we can find out what game data, or feature, can impact the result the most and thus explain the result. By compression, we only have access to two feature that have no interpretable relation with LOL data, making the task in this paper specific to purely prediction.

References

Yadolah, D. (2008). Lagrange multiplier. In *The concise encyclopedia of statistics* (pp. 292–294). Springer New York. https://doi.org/10.1007/978-0-387-32833-1_218

Games, R. (n.d.). Riot developer portal [Accessed: 2024-09-04].

Pearson, K. (1895). Note on Regression and Inheritance in the Case of Two Parents. *Proceedings of the Royal Society of London Series I*, *58*, 240–242.

Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *2*(11), 559–572. https://doi.org/10.1080/14786440109462720

Cox, D. R. (1958). The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, *20*(2), 215–232.

Appendix A

Additional proofs

Convexity

In single-variable calculus, we use the second derivative test: If $f'(x) = 0$ and $f''(x) < 0$, then x is a local maximum.

Similarly, here we need to find the secondary partial derivative. The secondary partial derivative of formula 27 is:

$$H = \frac{\partial^2 \mathscr{L}}{\partial u^2} = 2C - 2\lambda I \tag{A1}$$

Where H is a new matrix called the Hessian matrix. For the function to be convex, it needs to be negative definite, which means for any non-zero vector v, we should have:

$$v^\top H v < 0 \tag{A2}$$

Recalling the eigen form in equation 28, the matrix C has several eigenvalue, we are typically interested in the largest. The eigenvalue of C are denoted $\lambda_1, \lambda_2, ..., \lambda_n$.

Next, for any non-zero vector v, we can express it as a linear combination of eigenvectors of C, calling them $k_1, k_2, ..., k_n$. We take the largest, making $\lambda = \lambda_1$

$$v = c_1 u_1 + c_2 u_2 + ... + c_n u_n \tag{A3}$$

Now, substituting into the equation A2:

$$v^T H v = 2u^\top C u - 2\lambda_1 u^\top u \tag{A4}$$

$$= 2(c_1 u_1 + ... + c_n u_n)^T C(c_1 u_1 + ... + c_n u_n) - 2\lambda_1 u^\top u \tag{A5}$$

Because $Cu_i = \lambda_i u_i$,

$$= 2(c_1^2\lambda_1 + c_2^2\lambda_2 + \ldots + c_n^2\lambda_n) - 2\lambda_1 u^\top u \tag{A6}$$

$$= 2(c_1^2\lambda_1 + c_2^2\lambda_2 + \ldots + c_n^2\lambda_n) - 2\lambda_1(c_1^2 + c_2^2 + \ldots + c_n^2) \tag{A7}$$

$$= 2(c_2^2(\lambda_2 - \lambda_1) + c_3^2(\lambda_3 - \lambda_1) + \ldots + c_n^2(\lambda_n - \lambda_1)) \tag{A8}$$

As we can see, for each term, there is one $\lambda_i - \lambda_1$, because λ_1 is the largest, so $\lambda_i - \lambda_1$ is negative, and making the whole expression negative, in2dicating that the Hessian matrix H is negative definite. Thus, we had proven the convexity of the function $\mathscr{L}$.

Appendix B

Python Codes

Setup

```
# imports
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# import csv
raw_data=pd.read_csv('rank-data/high_diamond_ranked_10min.csv').dropna()
raw_data.info()

raw_data.head()
```

Plotting heatmap

```
raw_data.drop('gameId',axis=1,inplace=True)
raw_data.head()
raw_data

# plot heat map
selected_columns =
    raw_data.columns[1:5].tolist()+raw_data.columns[18:23].tolist()
selected_df = raw_data[selected_columns]
corr_matrix=selected_df.corr()
sns.heatmap(corr_matrix,
                # annot=True,
                cmap='coolwarm')
plt.figure(figsize=(20,10))
plt.show()
```

Normalization

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

scaler = StandardScaler()
scaled_features=scaler.fit_transform(features)

pca=PCA(n_components=2)
principal_components=pca.fit_transform(scaled_features)
feature_pca=pd.DataFrame(data=principal_components)
feature_pca.head()
#convert scaled features to dataframe
scaled_features=pd.DataFrame(data=scaled_features,columns=features.columns)
```

Comparison scatter plot

```
plt.figure(figsize=(12, 6))

# Before PCA
plt.subplot(1, 2, 1)
sns.scatterplot(x=features.iloc[:, 0], y=features.iloc[:, 1],
   hue=raw_data['blueWins'])
plt.title('Before PCA')
plt.xlabel('blueWardsPlaced')
plt.ylabel('blueWardsDestroyed')

# plt.subplot(1, 2, 2)
# sns.scatterplot(x=selected_features.iloc[:, 0],
    y=selected_features.iloc[:, 1], hue=raw_data['blueWins'])
# plt.title('Before PCA')
# plt.xlabel('blueDeaths')
# plt.ylabel('redTotalGold')

plt.tight_layout()
plt.show()

# After PCA
sns.scatterplot(x=feature_pca[0], y=feature_pca[1],
    hue=raw_data['blueWins'])
plt.title('After PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
```

Logistic regression

```
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(feature_pca,
    label, test_size=0.2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(feature_pca,
    label, test_size=0.2, random_state=42)

X_train.shape,X_test.shape,y_train.shape,y_test.shape

from sklearn.metrics import accuracy_score, confusion_matrix,
    classification_report

from sklearn.linear_model import LogisticRegression
```

```
# fit the model
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
log_reg_pred = log_reg.predict(X_test)
# accuracy
log_reg_accuracy = accuracy_score(y_test, log_reg_pred)
log_reg_accuracy
```

Plotting decision boundary on scatterplot

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

X=X_train.values
y=y_train.values

# Sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Logistic regression with gradient descent
def logistic_regression(X, y, learning_rate=0.1,
   num_iterations=1000):
    m, n = X.shape
    theta = np.zeros(n)
    theta_history = []

    for i in range(num_iterations):
        z = np.dot(X, theta)
        h = sigmoid(z)
        gradient = np.dot(X.T, (h - y)) / m
        theta -= learning_rate * gradient

        if i % 100 == 0:  # Store theta every 100 iterations
            theta_history.append(theta.copy())

    return theta, theta_history

# Run logistic regression
theta, theta_history = logistic_regression(X, y)

# Plotting
plt.figure(figsize=(12, 8))

# Plot the data points
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=y)
# sns.scatterplot(x=[:, 0], y=[:, 1],hue=y_train)
# plt.scatter(X[y==1][:, 0], X[y==1][:, 1], c='r', label='Class 1')

# Plot decision boundaries at different iterations
x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, 0.01),
                       np.arange(x2_min, x2_max, 0.01))

for i, theta in enumerate(theta_history):
    Z = sigmoid(np.c_[xx1.ravel(), xx2.ravel()].dot(theta))
    Z = Z.reshape(xx1.shape)

    # plot if on last iterations
    if i == len(theta_history) - 1:
        plt.contour(xx1, xx2, Z, [0.5], colors=['g'],
                    linestyles=['--'],
                    label=f'Iteration {i*100}')

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Logistic Regression - Gradient Descent Process')
plt.legend()
plt.show()
```