Predicting League of Legends Game Outcomes Using Binary Classification Techniques

Thomas Lee
Boston University
tlee03@bu.edu

Abstract

League of legends (LoL) is a popular online video game with millions of active players. Much work has been done to predict the outcomes of individual matches given various features of the games. To this end, most of these experiments conducted have been focused on binary classification models, the results of which I have attempted to reproduce in my own project. I use logistic regression, decision trees, random forest, and naive Bayes classifier to predict the outcome of professional match games in League of Legends. Features are taken from the Kaggle dataset which include player statistics and champions picked. I use different techniques to increase the testing accuracy such as PCA, one-hot encoding, and regularization. I found that naive Bayes classifier, random forest, and logistic regression performed relatively the same. I also found that the most important features pertained to the stats of the blue team, specifically the middle lane player.

1 Introduction and Motivation

League of Legends (LoL) is a 5 versus 5 online battle arena video game published by Riot Games, where one team is labeled blue and the other red. Here, each person picks a different role/position: top, middle, bottom (ADC), support, and jungle. In LoL, each person chooses one champion to play as and fight as a team in order to complete objectives to beat the other team. This includes eliminating the enemy champions, destroying turrets, and killing monsters. The game boasts of 150 million players and a thriving competitive scene with millions in tournament prize pools as well as online viewers. With such a large following, it would be valuable to gain more insights on predicting match outcomes as well as understanding which features of a game are the most important for determining the outcome. In this project, I will be predicting if a team is going to win against another team in a competitive game in League of Legends based on pre-game data.

In order to accomplish this objective, I utilized a dataset [1] of professional LoL matches from 2021 on Kaggle. This dataset includes features such as the outcome of the fight, champions picked of each player, the number of games the specific player has with that champion, and their average win rate and kill death ratio with that champion. Kill-death ratio is defined as the ratio of the number of eliminations a player has to their number of deaths. The outcome of the fight is either a 0 or 1, with 1 encoding a win for the blue team and 0 for a loss. In total, this dataset includes 41 features.

In this research paper, I primarily explored the logistic regression model's performance on predicting match outcomes. In addition, I also compared the performance with a few other predictive models such as random forest, categorical naive Bayes and decision trees. While experimenting, I found that logistic regression, random forest, and categorical naive Bayes had the highest testing accuracies. I was also able to conclude that the most important features were those pertaining to the blue team.

Code to all experiments can be found in the following Github repo:

`https://github.com/ThomasLee03/CS365-Project/tree/main`

2 Related Work

As it is a highly relevant topic, there has been a plethora of research done on predicting the outcome of matches. Some of the popular models explored in the literature include different tree-based models [2 3], naive Bayes [5], and logistic regression [3, 4, 5]. In my project, I was able to reproduce some of the results found in other experiments using these predictive models. I will give a high-level overview of these models and touch on how I adapted them to my own work.

2.1 Decision trees

Decision trees [10] are a tree structure where the internal nodes denote a feature, branches denote the rule of split, and the leaf nodes denote the result of the algorithm.

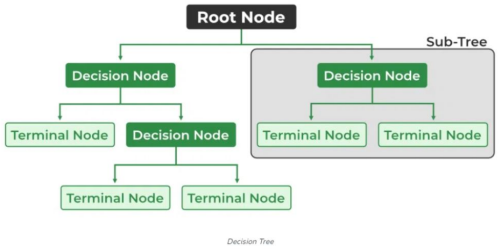

Figure 1: Decision Tree Diagram

I utilized the Gini index as the criterion for creating new branches in the tree and set the max depth to 7 to prevent overfitting.

2.2 Random forest

Random forest [8] is an algorithm which creates several decision trees during the training phase. Each decision tree then is created using a random subset of the original data set to measure a random subset of feathers for each partition. During prediction, random forest precedes to aggregate the results of the trees, commonly by voting for classification tasks. This is also another powerful model as it utilizes multiple decision trees, making it more accurate than a single decision tree. Here, I followed the default procedure using 100 trees in the forest, again using the Gini index as a branching criterion.

2.3 Categorical Naive Bayes

Categorical Naive Bayes [4] is a classification algorithm for categorically distributed data. It assumes that each feature has its own categorical distribution. For each feature in the training set, Categorical Naive Bayes estimates a distribution for each feature of the training set conditioned on the target value. This might be helpful when attempting to model my data as some features such as the champions picked are categorical.

2.4 Logistic regression

The logistic regression model [7] is a linear function composed with the sigmoid function to models probabilities directly. In my implementation of logistic regression, I attempted with one hot encoding, with and without ℓ_2 regularization, and optimization solvers such as Newton-Cholesky and L-BFGS. I also attempted to apply PCA to the dataset with varying the number of components kept to use as features. Regularization is a method that aids in counteracting overfitting data. Solvers are numerical algorithms in order to solve the optimization problem.

3 Resources

Each algorithm was performed with 10 trials each through a LAPTOP-V8C2A4N5 with 3201 Mhz, 8 Core(s), 16 Logical Processor(s), and 15.4 GB of Total Physical Memory. I utilized the

`scikit-learn` and `pandas` packages in Python to train my models. I also consulted ChatGPT the syntax of these packages.

4 Method

As mentioned previously, I utilized decision trees [11], naive Bayes classifier [4], random forest [14], and logistic regression [12] with the `scikit-learn` package. I will briefly summarize decision trees and naive Bayes classifiers, and go more into depth with logistic regression as it was my focal point for this research paper.

4.1 Decision trees

A decision tree [10] can be learned by splitting based on certain criteria. These criteria are used to identify the attribute that creates the most homogeneous subsets of data after each split, thus maximizing the information gain. Trees tend to terminate once splitting no longer yields added value to predictions, when the subset at a node all have the same value of the target variable, or once it has reached a depth limit.

Going more into depth of the criterion, I utilized the Gini index. The Gini index is a score that determines the accuracy of a split among the classified groups. It ranges from 0 to 1, where 0 occurs if all observations belong to one class and 1 if it is a random distribution of the elements within the classes. In my testing, we want the score to be as low as possible. The Gini index (g) can be represented as:

$$g = 1 - \Sigma p_i^2$$

where p_i is the portion of elements belonging to the i'th category.

4.2 Categorical naive Bayes

Categorical naive Bayes [4] is similar to regular naive Bayes in the fact that it assumes that the features are conditionally independent given the output. However, since we assume that each feature has it's own categorical distribution, we obtain a slight variation of the probability estimation.

The probability of category t in feature i given class c is estimated as:

$$P(x_i = t|y = c; \alpha) = \frac{N_{tic} + \alpha}{N_c + \alpha n_i},$$

where $N_{tic} = |\{j \in J|x_{ij} = t, y_j = c\}|$ is the number of times category t appears in the samples x_i, which belong to class c, $N_c = |\{j \in J|y_j = c\}|$ is the number of samples with class c, α is a smoothing parameter, and n_i is the number of available categorical features i.

4.3 Logistic Regression

Logistic regression [7] models the input-output relationship probabilistically via a parametrized model f_θ. That is, given a set of input features x which is assumed to follow the distribution of some random variable X, we can compute the probability that the output Y is 1. The parameters of the model f is given by θ, so the relationship we would like to capture is given by

$$\mathbb{P}(Y = 1|X = x) = f_\theta(x).$$

Consequently,

$$\mathbb{P}(Y = 0|X = x) = 1 - f_\theta(x).$$

Since the outputs y can only take on the values of zero or one, this can be written more compactly as:

$$\mathbb{P}(Y = y|X = x) = f_\theta(x)^y(1 - f_\theta(x))^{1-y}$$

Afterwards, to convert our probabilities to binary outputs, we will round the probabilities up or down, e.g. if $f_\theta(x) = 0.76$, we will count that as a win.

Recall that the basic form for the linear classifier h_θ is

$$h_\theta(x) = \theta^T x.$$

This, however is not amenable to modeling probabilities as the range of f is all of $\mathbf{R}$. To restrict the range of the model to $[0, 1]$, pass the output of $h_\theta(x)$ into the sigmoid function g which is defined as

$$g(z) = \frac{1}{1 - e^{-z}}.$$

Hence, the full logistic regression model is of the form

$$f_\theta(x) = g(\theta^T x) = \frac{1}{1 - e^{-\theta^T x}}.$$

Now, given a dataset of n input-output pairs $\{(x_i, y_i)\}_{i=1}^n$, we can find the optimal θ fit to this data set via maximum likelihood estimation. We define the likelihood function as

$$L(\theta) = \prod_{i=1}^{n} \mathbb{P}(Y = y_i | X = x_i)$$

Using the assumption that the distribution $\mathbb{P}$ can be parametrized with f_θ, this can be further expanded as

$$L(\theta) = \prod_{i=1}^{n} f_\theta(x_i)^{y_i} (1 - f_\theta(x_i))^{1-y_i}$$

Taking the natural logarithm of both sides, we get the log likelihood function

$$\ell(\theta) = \sum_{i=1}^{n} y_i \log(f_\theta(x_i)) + (1 - y_i) \log(1 - f_\theta(x_i)).$$

Maximum likelihood estimation is thus the problem of maximizing the above quantity. However, it is more amenable to instead minimize the negative of $\ell(\theta)$, i.e. letting $J(\theta) = -\ell(\theta)$. We can equivalently solve the problem:

$$\min_\theta J(\theta)$$

to compute the maximum likelihood solution $\theta^\star$. Here, J is known as the binary cross-entropy loss.

4.4 Regularization

To help prevent overfitting, it is often helpful to include a regularization term [6] in the objective as:

$$\min_\theta J(\theta) + R(\theta)$$

where $R(\theta)$ is a penalty term on the size of θ. Common choices for R include $R(\theta) = \lambda\|\theta\|_2^2$ and $R(\theta) = \lambda\|\theta\|_1$ known as ℓ_2 and ℓ_1 regularization respectively. Here, $\lambda \geq 0$ is the regularization parameter.

The purpose for regularization is to prevent the model from overfitting to the data. The larger λ is, the more weight we put on the regularization term, which discourages large coefficients. For this project, we have opted to use ℓ_2 regularization as it is twice differentiable and hence amenable to (quasi-)Newton methods for optimization. A more detailed dicsussion on regularization and numerical optimization methods is deferred to the appendix.

4.5 PCA

Another commmon form of data preprocessing that I also tried in my project is Principal Component Analysis. Given a standardized data matrix $X \in \mathbf{R}^{n \times p}$, the principal components are easily computed by the SVD of X and the data projected onto the principal components are used as the new features to fit on. Mathematically, the reduced SVD of X keeping only k singular values is given by

$$X \approx U_k \Sigma_k V_k^T$$

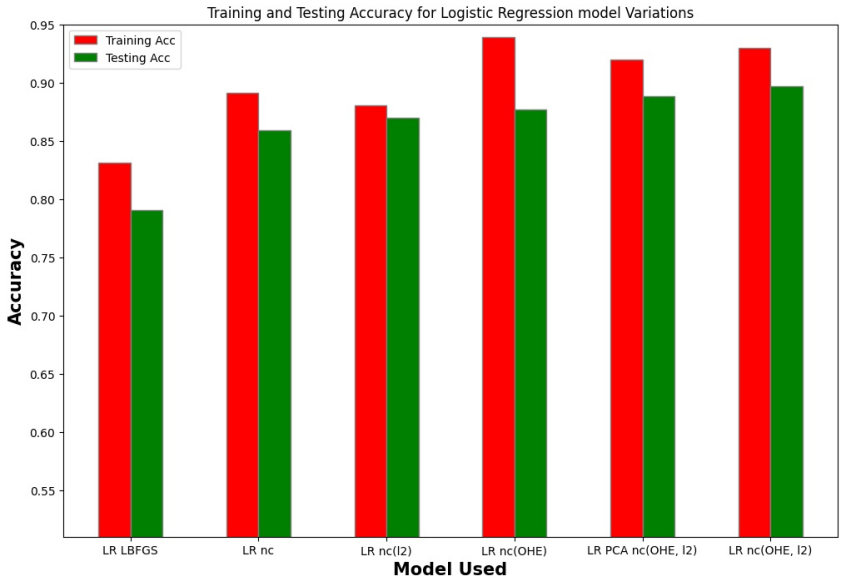

Figure 2: Comparison of logistic regression variations

where the columns of V_k contain the relevant principal components. The idea behind PCA preprocessing is then to use the data matrix $W = XV_k$ as the transformed features and to fit onto W instead of X.

PCA extracts the most relevant parts of the data and acts as an implicit form of regularization. Primarily, PCA extracts the parts of the data with maximal variance and discards high-frequency data (noise). This can be seen as a denoising procedure and can thus help prevent overfitting to noise in the predictive model.

5 Experimental Results:

I will now discuss the practical steps of my experiments which include data collection, cleaning, and modeling.

To reproduce the results, one can follow this procedure:

1. Download `dataset_picked_champions_players_statistics` from the dataset as csv files.
2. Split the dataset randomly into a training and testing dataset 80/20 for cross-validation.
3. Use `scikit-learn` to train the logistic regression model.
4. Use the trained model to predict outcomes on the test set.
5. Repeat steps 2 and 4 (with modifications to the Python file to include the different variations of logistic regression) 10 times and record the average of the training and testing accuracy.

5.1 Comparison with different logistic regression models

We notice that in figure 3, L-BFGS had a poor performance compared to the Newton-Cholesky solver. This is due to the fact that it could not converge. I predict this is due to the data being poorly conditioned. Because of this, I focused mainly on utilizing the Newton-Cholesky solver.

We also notice that without one-hot encoding, we can see that using ℓ_2 regularization did decrease our training accuracy and increase our testing accuracy (if only slightly), as we predicted it would. We also saw an increase in both after utilizing one-hot encoding, which makes sense as the features (our champion id numbers) are not ordered from "worst" to "best". For the implementation with PCA, I decided to pick the first 201 components, as I picked components 1 through 50 with increments of

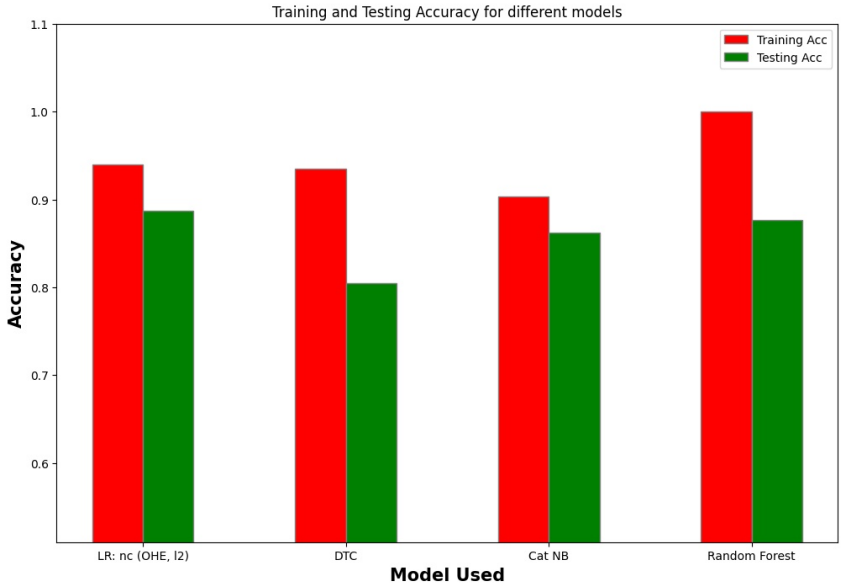

Figure 3: Comparison of all models.

50 and found 201 to have the highest results. However, we saw comparable performance with and without PCA applied to the dataset.

5.2 Comparison with other models

To better understand how logistic regression stands against other common models, I ran 10 trials each on decision trees, categorical Naive Bayes, and random forest. Using the same procedure as before except with these other models, we see:

Overall, we saw a relatively similar performance with categorical Naive Bayes (cat NB), random forest, and Logistic regression utilizing the solver Newton-cholesky (nc) with ℓ_2 regularization. I tried depths between 3-8 and no specified depth for decision tree classifier (DTC), yet it still had relatively poor performance. This is to be expected due to the fact that decision trees are prone to overfitting data.

5.3 Feature importance

I also analyzed feature importance in the logistic regression model by plotting the absolute value of the coefficients.

In figure 4, we see that in general, the features pertaining to the blue team had higher importance (larger absolute value coefficients). This correlates to the well known inherent asymmetry of win rates between the blue and red team in League of Legends. Looking more closely at these features, we see that the features pertaining to the positions of middle laners were the most important. This is attributed to the fact that this role is able to maintain control of their lane while also helping take down other objectives and aid other players, making it the most impactful role in the game.

What was interesting to find was that the champion picked for the middle lane was considered to have the most impact. In other studies, it has been found that the champions picked alone [3, 5] are not very good predictors. This was also seen in my exploration, where I fit my model with features only pertaining to champions picked and saw a decrease of testing accuracy of more than 20%. A possible explanation to this is that a champion's power can dramatically increase due to synergies with other champions. Since the middle lane has the most impact, having good combinations with the champions other players pick plays a key role in victory.

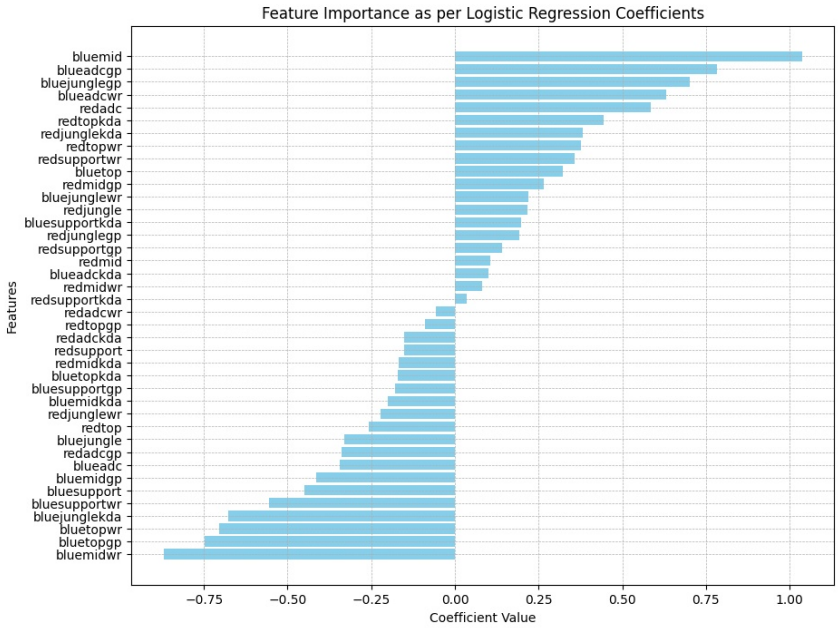

Figure 4: Feature importance as per logistic regression coefficients.

6 Conclusion

Predicting matches of professional League of Legends professional teams is a valuable and insightful problem. My models achieved ∼0.88 testing accuracy using Random Forest, Naive Bayes classifier, and Logistic regression algorithms using the dataset with pre-game player statistics. This indicates that one can create reliable predictions of match results with only pre-game player statistics. In such a competitive scene such as League of Legends with millions on bets, there is much motivation to conduct further research. Nevertheless, my research displays three excellent models one can use. One can also observe that the features relating to the blue team, specifically the mid lane player are the most important features.

During the process of my research, I struggled with obtaining high training and testing accuracies with my original dataset [15] as this dataset did not contain any information about previous match history. In almost all studies done [2, 3, 5] it was shown that simply having the characters and abilities picked were not enough to obtain high accuracy readings. I then decided to train my models on the new dataset [1] and observed much higher results. After visualizing the feature importance and noticing the unexpected outcome that the middle laner's champion had the most importance, further research can be done on why this is the case.

References

[1] Costa, L. (2021, May 24). League of legends pre-match data 2021. Kaggle. https://www.kaggle.com/datasets/tekpixo/leagueoflegendsprematch2021

[2] Decision tree application for choosing champions in league ... (n.d.). https://informatika.stei.itb.ac.id/ rinaldi.munir/Matdis/2019-2020/Makalah2019/13518003.pdf

[3] Feature analysis to league of legends victory prediction on the picks and bans phase | IEEE conference publication | IEEE Xplore. (n.d.-c). https://ieeexplore.ieee.org/document/9619019

[4] 1.9. naive Bayes. scikit. (n.d.). https://scikit-learn.org/stable/modules/naive$_bayes.html$: : $text = CategoricalNB\%20implements\%20the\%20categorical\%20naive\%20Bayes\%20algorithm\%20for, i\%20of\%20X\%20conditioned\%20on\%20the\%20class\%20y.$

[5] Jonduke. (2020a, March 6). Attempting to predict league of legends match outcomes. Medium. https://medium.com/@jonduke90/attempting-to-predict-league-of-legends-match-outcomes-9b92814a0215

[6] Boyd, S. P., & Vandenberghe, L. (2023). Convex optimization. Cambridge University Press.

[7] CS229 lecture notes. (n.d.-a). https://cs229.stanford.edu/lectures-spring2022/main_notes.pdf

[8] Random Forest algorithm in machine learning. GeeksforGeeks. (2024b, February 22). https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/

[9] GeeksforGeeks. (2024, March 1). Naive Bayes classifiers. https://www.geeksforgeeks.org/naive-bayes-classifiers/

[10] GeeksforGeeks. (2023, August 20). Decision tree. https://www.geeksforgeeks.org/decision-tree/

[11] Sklearn.tree.decisiontreeclassifier. scikit. (n.d.-e). https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.htmlsklearn.tree.DecisionTreeClassifier

[12] Sklearn.linear$_m odel.logisticregression.scikit.(n.d. - d).https : //scikit - learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html$

[13] SKLEARN.DECOMPOSITION.PCA. scikit. (n.d.-b). https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[14] Sklearn.ensemble.randomforestclassifier. scikit. (n.d.-b). https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[15] Campanelli, P. (2017, October 26). League of Legends ranked matches. Kaggle. https://www.kaggle.com/datasets/paololol/league-of-legends-ranked-matches

7 Supplementary Material

In this appendix, I will discuss in greater depth the details of second-order optimization methods as well as the role of regularization. We follow the treatment of optimization and regularization as presented in [6].

7.1 Optimization

The (regularized) maximum likelihood problem is an unconstrained convex problem and can be thus solved via Newton's method. Moreover, because the objective is convex, the solution that Newton's method produces is globally optimal.

To recap Newton's method for general convex minimization, we assume our objective function $f : \mathbf{R}^n \to \mathbf{R}$ is twice differentiable and form a second order Taylor expansion about a point $x \in \mathbf{R}^n$.

$$f(x + \Delta x) \approx f(x) + \nabla f(x)\Delta x + \frac{1}{2}\Delta x^T \nabla^2 f(x)\Delta x.$$

The minimization of this quadratic function (over Δx) is thus given by

$$\nabla^2 f(x)\Delta x = -\nabla f(x),$$

and hence,

$$\Delta x = -(\nabla^2 f(x))^{-1}\nabla f(x)$$

Newton's method iteratively minimizes f by minimizing successive quadratic approximations of f. In particular, each iteration of the Newton method solves for a new Δx and updates our current iterate as $x \leftarrow x + \Delta x$.

As an aside, we note that Newton's method is a descent method that converges faster than gradient descent. Gradient descent only incorporates first-order derivative information and thus may exhibit erratic convergence, zig-zagging back and forth. This is known to be the case when the level sets of the objective function are anisotropic, i.e. stretched out ellipses. By contrast, the Newton method incorporates second-order derivative information and thus produces a descent step that converges better than gradient descent.

To solve the linear system formed at each step, note that since f is convex, $\nabla^2 f$ is positive semidefinite. Thus, a Cholesky factorization of $\nabla^2 f$ exists and can be performed to solve the linear system. This is the basis of the `newton-cholesky` option I have opted to use in solving our logistic regression model.

Note that for each iteration of the Newton method, we are required to form the full Hessian as part of the solution process. In memory-limited settings however, this may not be feasible, so we also explore a quasi-Newton method known as Limited-Memory Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS). Essentially, L-BFGS forms an approximation to $(\nabla^2 f(x))^{-1}$ using only a few vectors at a time to iteratively update the inverse Hessian at each iteration. While suitable for large-scale settings, note that when the data is poorly conditioned (and hence, so is $\nabla^2 f(x)$), L-BFGS may exhibit poor numerical behavior and thus, converge poorly if at all.

7.2 Regularization and Curvature

Let f be a twice-differentiable convex function and $\lambda > 0$. We consider the regularized optimization problem:

$$\min_x f(x) + \lambda\|x\|_2^2.$$

This is exactly the form of the regularized logistic regression model. The curvature of the function is given by the definiteness of the Hessian of the objective. Now we note that

$$\nabla^2(f(x) + \lambda\|x\|_2^2) = \nabla^2 f(x) + \lambda I.$$

Since f is convex, the eigenvalues of $\nabla^2 f(x)$ are nonnegative for any $x \in \mathbf{R}^n$. Adding the λI term ensures that the eigenvalues of the Hessian of the regularized objective are strictly positive, and hence invertible.