Final Project

Hanxiong Wang RUID: 149002562 NetID: hw277
Ruofan Yan RUID:147003209 NetID:ry91
Ruiheng Yin RUID: 152000034 NetID:ry99

May 2, 2016

Contents

1 Introduction

The National Basketball Association (NBA) is one of the largest sports leagues and it has millions of fans across the world. NBA players are the world's best paid sportsmen. There is a large market related to NBA and the outstanding NBA players have huge business values.

Finding those genius players is thus a valuable thing to do. Machine learning methods can be applied to achieve this goal. NBA is suitable for this application of machine learning methods because it contains a significant amount of easily accessed data and the performance of NBA players at different positions can be measured using the same set of statistics.

Here in our project, we chose three classification methods to predict the All Star players and All League Team players. All Star players and All League players are the outstanding players of each season. For each season, there are 24-29 players that are chosen to form two All Star team and play the NBA All-Star Game. Also, for each season, there are 15 best players who are given the honor to form the All-NBA Team (All League Team). By predicting All Star players and All League players respectively using supervised machine learning: three classification methods, we can achieve our goal of finding the outstanding NBA players. Same procedure can be applied in other sports area.

2 Data selection and preprocessing

The data of NBA players are collected from the website: http://www.basketball-reference.com/. The player records of 35 seasons (1980-2015 seasons) are collected and for each season, the data contains the record of around 480 players, which gives 16116 NBA players' data in total.

There are 23 features of each player record that are used for all star team player and all league team player prediction. These features are: field goals, field goal attempts, 2 point field goals, 2 point field goal attempts, 3 point field goals, 3 point field goal attempts, free throws, free throws attempts, offensive rebounds, defensive rebounds, total rebounds, assists, steals, blocks, turnovers, personal fouls, points, field goal percentage, 2-point field goal percentage, 3-point field goal percentage, effective field goal percentage, free throw percentage and true shooting percentage.

For All Star dataset, we labeled 1 to All Star team players and labeled 0 to others. Similarly, for all League dataset, All League Team players are labeled as 1 and 0 for others. We split the dataset into two parts: data of seasons 1980-2007 are the training dataset, and data of seasons 2008-2015 are the testing dataset. Since the dataset is pretty unbalanced, for example, only 28/15 players out of 480 players a season have label 1 for All Star/All League team players and the rest all have label 0. To solve this problem, we duplicated the player records with label 1 to obtain the dataset with balanced 0/1 labels. For All Star dataset, we duplicated player records whose label is 1 for 16 times and for All League dataset, we duplicated 32 times. Finally we resorted the dataset into a random order, so that duplicated sample do not appear consecutively in out dataset.

3 Classification method

3.1 Perceptron/Kernel Perceptron

3.1.1 Perceptron

Perceptron is an algorithm for learning a binary classifier: a function that maps its input $\vec{x}$ to an output binary value.

$$Y = sgn(b + \sum_i w_i X_i)$$

Initialize the weights $\vec{w}$ and b to be zero, pick a learning rate m (this is a number between 0 and 1). Loop over the training samples for T iterations: for each training sample $\vec{x}$, let its true label be y_i (+1 or -1). Change the players' label 0 to -1. When there is mistake, update the weights $\vec{w}$ and b:

$$\vec{w}_{t+1} = \vec{w}_t + y_i \vec{x}_i, \quad \vec{b}_{t+1} = \vec{b}_t + m \times y_i$$

Pick a T such that after learning for T iterations, the test accuracy is good.

3.1.2 Kernel Perceptron

From the implementation of Perceptron algorithm, we conclude that the data is not linearly separable. So we introduced a kernel function into the classification method to map the lower dimension space into higher dimension space. The Kernel used is Gaussian kernel:

$$K(x, x') = exp(-\frac{||x - x'||^2}{\sigma^2})$$

In comparison with perceptron, kernel perceptron algorithm stores a subset of its training examples $\vec{x_i}$, associates with each a weight α_i. For each training sample $\vec{x_j}, y_j$,

$$\hat{y} = sgn(\sum_i \alpha_i y_i K(x_i, x_j))$$

If $\hat{y} \neq y_j$, update α_j: $\alpha_j = \alpha_j + 1$. Loop over training examples for T iterations.

3.2 Support Vector Machine

3.2.1 Overview

The main idea in the Support Vector Machine(SVM) is not only finding a hyper plane $\vec{w} \cdot \vec{x} + b = 0$ which can classify the labels, but also making this hyper plane has a maximum margin over all the possible hyper plane so that we could own a maximum classifier confidence. Mathatically speaking, we are trying to solve the following optimization problem

$$\max \frac{1}{||\vec{w}||} \qquad s.t., y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1, i = 1, 2, ..., n \tag{1}$$

Once we solve the problem (1), the support vector are those satisfied constrain

$$y(w^T x + b) = 1 \tag{2}$$

otherwise the vectors are not support vectors and wont have effect on the optimal hyper plane. Thus, the hyper plane determined by the support vectors can be given

$$\vec{w} = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}, \qquad b = -\frac{\max_{i:y^i=-1} \vec{w} x^{(i)} + \min_{i:y^i=1} \vec{w} x^{(i)}}{2} \tag{3}$$

3.2.2 Dual problem

Reconsider the problem (1), it is equivalently to the minimization problem

$$\min \frac{1}{2}||\vec{w}||^2 \qquad s.t., y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1, i = 1, 2, ..., n \tag{4}$$

Because of the properties of the problem, we can consider Lagrange duality and try to solve the dual problem. The advantages of solving a dual prolem instead of the origion one includes the simplicity of solving a dual problem and introducing the kernel method to deal with some linearly non-seperable cases.

By adding Lagrange multipliers α_i on each constrains, we could define Lagrange funtion as

$$L(\vec{w}, b, \alpha) = \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{n} \alpha_i (y_i(\vec{w} \cdot \vec{x_i} + b) - 1) \tag{5}$$

Therefore the minimization problem (4) is converted into a minimax problem

$$\min_{\vec{w},b} \max_{\alpha_i \geq 0} L(\vec{w}, b, \alpha) \qquad s.t., y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1, i = 1, 2, ..., n \tag{6}$$

The dual form of the minimax problem is

$$\max_{\alpha_i \geq 0} \min_{\vec{w},b} L(\vec{w}, b, \alpha) \tag{7}$$

The core idea here is that in the minimization problem (6) is satisfied by the **Karush Kuhn Tucker** (KKT) conditions. In mathematical optimization theory, this guarantees that the minimax problem (6) and the maximin problem (7) are completely equivalent. Based on this fact, we just need to solve the dual problem (7).

The first step to solve (7) is trying to eliminate the parameter $\vec{w}$ and b. By minizing L with fixed α, we take the derivative of L with respect to $\vec{w}$ and b and set them as zero.

$$\begin{cases} \frac{\partial L}{\partial w} = 0 \Rightarrow \ \vec{w} = \sum_{i=1}^{n} \alpha_i y_i \vec{x_i}, \\ \frac{\partial L}{\partial b} = 0 \Rightarrow \ \sum_{i=1}^{n} \alpha_i y_i . = 0 \end{cases} \tag{8}$$

Substitute this result into (7), we finally obtain a concise form of the optimization problem

$$\begin{aligned} &\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \vec{x_i} \cdot \vec{x_j} \\ &s.t., \alpha_i \geq 0, \ \sum_{i=1}^{n} \alpha_i y_i = 0 \quad i = 1, ..., n \end{aligned} \tag{9}$$

3.2.3 Extensions to outliers and kernel function

For those datasets which have unavoidable noises, or say outliers, one method to deal with thess cases is applying an penalty term, changing the contrains in problem (6) as

$$y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1 - \xi_i, \ i = 1, 2, ..., n \tag{10}$$

where the $\xi_i \geq 0$ are called slack variables, which allow data points $\vec{x_i}$ to have some tolerance to the noise. In principle, larger slack variables ξ will produce more permissible hyper plane. Thus, we shall add a penalty term in the function (4), the problem with the contrains now is given

$$\min \frac{1}{2}||\vec{w}||^2 + C \sum_{i=1}^{n} \xi_i \tag{11}$$

$$s.t., \qquad y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1 - \xi_i, \qquad \xi_i \geq 0, i = 1, 2, ..., n \tag{12}$$

Generally, we call this SVM method as C-SVM.

Therefore new Langrange function is

$$L(\vec{w}, b, \vec{\xi}, \vec{r}, \vec{\alpha}) = \frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i(y_i(\vec{w}\cdot\vec{x_i}+b) - 1 + \xi_i) - \sum_{i=1}^{n}r_i\xi_i. \tag{13}$$

By the same analysis, the dual problem is

$$\begin{aligned} &\max_{\alpha}\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j \vec{x_i}\cdot\vec{x_j}, \\ &s.t., 0 \le \alpha_i \le C,\ \sum_{i=1}^{n}\alpha_i y_i = 0, \quad i = 1, ..., n, \end{aligned} \tag{14}$$

where C is a prescribed parameter which is determined by the trade off between finding the maximum margin and the tolerance of noise. Usually, for larger C, the tolerance of noise is smaller.

Moreover, we believe the dataset in most cases is not linear seperable so that a kernel function shall be introduced to map the lower dimension space into higher dimension space. Luckily, we only need a small change in the dual problem when considering kernel function as follows

$$\begin{aligned} &\max_{\alpha}\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j K(x_i, x_j), \\ &s.t., 0 \le \alpha_i \le C,\ \sum_{i=1}^{n}\alpha_i y_i = 0, \quad i = 1, ..., n, \end{aligned} \tag{15}$$

In our prediction model here, we employ Radial Basis Function(RBF) kernel

$$K(\vec{x_i}, \vec{x_i}') = \exp(-\gamma||\vec{x} - \vec{x_i}'||^2) \tag{16}$$

This kernel function is spreadly used and can map into infinite dimensional space. Practically, we use grid search the find the best parameter γ and C. And the dual problem can be solved numerically by Sequential Minimal Optimization(SMO) algorithm.

3.3 Neural Network

3.3.1 Overview

The key thing in the Neural Network is to get the derivative of each parameter for the cost function. And then use the conjugate gradient descent to get the optimal solution and optimal parameter for our problem. The Forward Process gives us the estimation based on the current parameter set for any input data while the Backward Propagation gives us the derivative estimation "revise" based on it. When we loop over all the samples in the training set, we get the current derivative. Then we use conjugate gradient descent to update the Θ_1 and Θ_2. We iterate 50 times (max iteration time we set) to get the updated Θ_1 and Θ_2.

3.3.2 Forward Process

In our problem, we use 1 input layer, 1 hidden layer and 1 output layer in our Neural Network system. Our input is a vector of 23 neurons (number of features) and our hidden layer consisting of 92 neurons. The number of our output layer neurons is 2. (Actually we can use only 1 neuron,

but in order to easily generalize to multi-class (for example n classes) classification problem, we use 2 neurons and in this way, we only need to change 2 into n, then the whole framework stays the same)

For a given Θ_1 and Θ_2, for any input data, we run the forward process and we will finally get a vector of 2 neurons at the output layer (α_1, α_2), and α_1 and α_2 indicates the probability that the prediction for this input data is 0 or 1. We mention here that our size of Θ_1 and Θ_2 are 92×24 and 2×93 respectively. The reason we use 24 rather than 23 and 93 rather than 92 in $\Theta_!$ and Θ_2 is because we want to add 1 as a bias term for our forward process.

3.3.3 Backward Propagation

For each sample i, we calculate the difference between our estimation output layer $h(x)_i$ and the true label y (notice true label 0 is represented to be $(1, 0)$ and true label 1 is represented to be $(0, 1)$) and denote it by δ_3. Then we do it in the reverse way based on the Θ_1 and Θ_2 to calculate $delta_2$. With this, we can calculate the derivative update for each parameter in round i. When we go through all the samples, we will have our derivative estimation for each parameter.

However, the backward propagation may not be correct, in order to test its accuracy, we choose a small data set and use both backward propagation and also Finite Different Method to calculate the derivative. If the difference of the result between these two methods is small enough, we have confidence in our backward propagation. Since the time cost for using Finite Different Method is expensive, we only use it as back testing.

3.3.4 Optimization

When we have the derivative from backward propagation, we can do conjugate gradient descent to update the parameters Θ_1 and Θ_2. We then iterate 50 times to get the updated Θ_1 and Θ_2.

3.3.5 Training Algorithm

Cost function:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{2}[-y_k^i \log((h_\theta(x^i))_k) + (1 - y_k^i)\log(1 - (h_\theta(x^i))_k)] + \frac{\lambda}{2m}[\sum_{i,j}(\theta_1^{i,j})^2 + \sum_{i,j}(\theta_2^{i,j})^2]$$

- Initialize the parameter Θ_1 and Θ_2 first. I have checked some reference, and have found that a reasonable way to choose the bound of each item in the Θ_1 and Θ_2 is $[-\xi, \xi]$, where $\xi = \frac{\sqrt{6}}{\sqrt{in+out}}$, where in means the number of input neurons and out means the number of output neurons. Hence in our case, it should be $\xi = \frac{\sqrt{6}}{\sqrt{23+92}}$.
- Set the max iteration number to be 50. You can also try some larger number, but the accuracy does not increase much.
  - Remember that we start with the current Θ_1 and Θ_2.
  - Initialize Δ_1 and Δ_2 to be zero matrix. Theses matrix are used to record the "contribution" of each training sample j to the calculation of the parameters derivative.
    - For each sample j, we start to do the following:
    - We input our features as vector a_1 and we add 1 as our bias term so that $a_1 = (1, a_1)$

* We use Θ_1 to help to get the linear combination of the input neurons to get a vector $z_2 = \Theta_1 a_1$
* Then we can calculate a_2 by using sigmoid function g to z_2. $a_2 = g(z_2)$
* Similar as a_1, we add 1 as bias term to a_2. $a_2 = (1, a_2)$
* Similar as z_2, now we get z_3 with the help of Θ_2. $z_3 = \Theta_2 a_2$
* $a_3 = g(z_3)$, where a_3 is our output layer, a vector of 2 neurons.
* We compare our result a_3 with the true label y, and denote the difference by δ_3, which is the error by output layer. $\delta_3 = a_3 - y$
* With the help of δ_3, we would like to calculate δ_2, which can be regarded as the error caused by each neurons in hidden layer. $\delta_2 = \Theta_2^{\mathrm{T}} \delta_3 g'(z_2)$
* Drop the first column of δ_2, that is drop the bias term.
* Update Δ_1 and Δ_2 by $\Delta_i = \Delta_i + \delta_{i+1} a_i^{\mathrm{T}}$ where $\delta_{i+1} a_i^{\mathrm{T}}$ is the "contribution" of sample j to the $Delta_i$.

– Define the matrix of parameter derivatives D_1 and D_2 by $D_i = \frac{1}{m}(\Delta_i + \lambda \Theta_i)$

– Run Conjugate Gradient Descent to update Θ_1 and Θ_2.

4 Prediction procedure

We shall notice that the in the raw datasets, the labels are severely unbalanced. That means if we simply guess all the testing data are labeled as 0, the traditional concept for the accuracy could still be very high but meaningless. Therefore in order to demonstrate the goodness of our classification methods more clearly, we define two kinds of accuracy type in our work. First one is called "0/1 Accuracy" and the second one is called "All star/ All League Team Pick Accuracy(with a fixed pool)"

4.1 0/1 Accuracy

The"0/1 Accuracy" is simply the classic concept for accuracy. The denominator is the total number of testing data which we predict correctly, including label 0 and label 1 while the numerator is size of the testing data.

Ideally speaking, if we have no information of the dataset and only do a random guess, the accuracy shall be 50%. In the Perceptron and SVM method, the label of each testing data is determined by the sign of the decision value to the hyper plane. In Neural Network, for each testing data, the output is a vector of 2 neurons, if the first 1 is bigger, we predict 0, otherwise, we predict 1.

4.2 All Star/All League Team Pick Accuracy

"0/1 Accuracy" can not really reflect the goodnees of the model because of unbalance dataset as we mentions before. Hence we proposed an alternatively kind of accuracy called "All Star/All League Team Pick Accuracy". This accuracy is used as we hope to pick a list of the most possible All Star/All League players in a prescribed pool. For instance, there is generally 28 players could be voted as All Star in a season officially. Then we pick the pool size as 28 and find the most 28 possible players in that season. The ratio of the number of the correct prediction to the all star player number 28 is the "All Star Pick Accuracy". Usually there are 500 players in the league, thus for naive random guess, the accuracy can be computed as $\frac{\sum_{i=1}^{28} C_{28}^{i} C_{500-28}^{28-i}}{C_{500}^{28}} = 5.6\%$. When we double

the pool size to 56, the accuracy is the ratio of the number of true all star players in this pool to the all star player number. Same for the all league accuracy.

Same as "0/1 Accuracy", we could compute a random guess as a standard. The results are given in the Table.1 and Table.2. Now, in the Perceptron and SVM method, we no longer investigate the sign of the decision value of the hyper plane, but sort all the decision value in descending order. Picking those data with higher rank in the list into the pool even the sign of the decision value indicates it is labeled as 0 instead of 1. While in Neural Network, we remove all the data with label 0, and for the rest, we descend the the order of the second neurons, which is the probability that the real label is 1. Then we also pick the higher rank data as our prediction.

5 Results and discussion

Table 1: All Star Prediction

	Random Guess	Perceptron	Perceptron with kernel	SVM	SVM with kernel	Neural Network
0/1 Accuracy	50%	84.24%	87.34%	89.55%	89.64%	89.65%
All Star Pick Accuracy (pool = 28)	5.6%	55.49%	56.64%	54.91%	56.65%	57.23%
All Star Pick Accuracy (pool = 56)	11.2%	80.92%	79.19%	81.50%	82.08%	80.90%

Table 2: All League Team Prediction

	Random Guess	Perceptron	Perceptron with kernel	SVM	SVM with kernel	Neural Network
0/1 Accuracy	50%	94.38%	95.41%%	95.34%	94.64%	95.67%
All League Team Pick Accuracy (pool = 15)	3%	67.00%	65.00%	66.00%	70.00%	71%
All League Team Pick Accuracy(pool = 30)	6%	90.00%	92.00%	92.00%	92.00%	93%

Table. 1 and Table. 2 illustrate the testing results for All Star prediction and All League Team prediction. Obviously, the results of all three classification methods are acceptable. You can see easily that the result for All League Team is a lillte better than All Star Team. The reason is easy and intuitive to understand. The All League Team is voted by the specialists who focus more on the current data itself, while the All Star Team is voted by the fans, therefore, they may prefer their idols whose current data maybe not good. Hence, the result of All League is better than that of the All Star one. Furthermore, we can see that SVM and Neural Network method is slightly better then Perception method. This is easy to understand since SVM method is always trying to find the hyper plane with maximum margin while Perceptron does not. And Neural Network is good at the non linear data, so it make sense that it does better than perceptron in this dataset. Meanwhile, SVM method with RBF kernel function also has a better result then SVM without kernel function after choosing a proper parameter γ. This justifies that kernel method works in dealing with linearly non-separable cases.

The error of the accuracy of the prediction comes from the following reasons:

First, in our dataset, we only have data to measure the performance of each player themselves but have no performance of the player's team and the link between the player to his team. Also, it must be different to evaluate a player for different position he played in the game. In fact, these related features are important when evaluating if this player is eligible for being a All Star or All League Team player.

Secondly, some others significant factors can not be counted easily in the dataset. For example, the fondness by basketball fans is obviously a vital factor when voting for All Star player, but it is really hard to give a detail measurement for each player on this feature. Also, whether a player's team location is in big city or not shall also work in the prediction. Notice that intuitively, we guess these two features play more important roles in All Star prediction then in All League Team prediction.

Thirdly, there are some unavoidable outliers in the dataset. This is quite common in the sports world. For example injury is a quite crucial factor. Kevin Durant was the MVP for the year 2013, however he was injured in the 2014. Even his average data in 2014 is still top 3 in the league, he can not be in the All League Team. Therefore, his data in 2014 is a huge outlier and misleading in our dataset. There are also some other factors cause the outliers in our dataset, which is not easy to avoid.

Last but not least, as we mentioned several times before, the dataset we use is a severe unbalance dataset. We duplicate the data to eliminate the unbalance here. There are also some other technology methods to deal with this problem, like Synthetic Minority Over-sampling Technique (SMOTE). This maybe improve the prediction to some extent.

6 Summary

In summary, we use three classification methods: Perceptron, SVM and Neural Network to build classifier models and predict NBA All Star/All League Team players. The test results are satisfied. SVM and Neural Network shows their advantage over Perceptron method. In the future work, by collecting more related features and using some technical methods for unbalance dataset , we believe that model accuracy could be improved further.

7 Appendix: Code Explanation

7.1 Data Processing

- DuplicateLabel.m duplicates player records with label 1 for a certain number of times
- RandomPerm.m makes random permutation of the dataset with duplicated player records

7.2 Perceptron

- For perceptron algorithm, MainPerceptron.m contains the main code
- For kernel perceptron algorithm, MainTrainKernel.m is the main code for training and MainTestKernel.m is the main code for testing.

7.3 Support Vector Machine

Code notes for All League Team prediction. Same for All Star prediction.

- Main_All_Leag_Svm.m contains the main code

- svmTrainSMOwhx.m works for building the svm model by inputting the training data, training label, the kernel function type and related parameter, the criterion for SMO algorithm.
- smomodifiedwhxbeta.m contains SMO algorithm by inputting the training data, training label, kernel matrix, SVM parameter and the criterion for stopping. This is a modified code from Gavin Cawley's MATLAB Support Vector Machine Toolbox.
- SVMcgSearch.m contains the code for searching the best parameter by grid search.
- svmPredictwhx.m used to predict the label of the testing data by inputting the svm model, testing data, testing label.
- kernelwhx.m contains the kernel function including the linear kernel and RBF kernel.

7.4 Neural Network

- Main.m contains the main code
- sigmoidGradient.m and sigmoid.m contains the sigmoid function and the its derivative.
- randInitilazeWeights.m contains the initial random starts of Θ_1 and Θ_2.
- predict.m contains the code regarding 0/1 accuracy.
- prediction.m contains the code regarding All Star/All League accuracy.
- GradientCheck.m and GradientCheckInitializeWeights.m are about the gradient checking about our backward propagation with Finite Different Method to make sure it's correct.
- FinteDifferenceMethod.m contains the code of Finite Difference Method.
- fmincg.m contains the code of conjugate gradient descent.
- CostFunction.m contains the code for cost function and forward and backward process.

References

[1] Chris Bishop, *Pattern Recognition and Machine Learning*

[2] John C. Platt, *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*

[3] Christopher J.C. Burges, *A Tutorial on Support Vector Machines for Pattern Recognition*

[4] Chih-Wei Hsu and Chih-Chung Chang, *A Practical Guide to Support Vector Classification*

[5] Nitesh V. Chawla, *SMOTE: Synthetic Minority Over-sampling Technique*

[6] Abebe Geletu, *Solving Optimization Problems using the Matlab Optimization Toolbox*

[7] Fiona Nielsen, *Neural Networks ? algorithms and applications*

[8] Carlos Gershenson, *Artificial Neural Networks for Beginners*

[9] Andrew Ng, *Machine Learning Coursera*

[10] Christos Stergiou, Dimitrios Siganos, *Neural Networks*