ML Final Project : Kobe Bryant Shot Selection

Doogie Min
Department of Mechanical Engineering
Seoul National Univ.
2012-11598
Bellatoris@snu.ac.kr

Yongseok Lee
Department of Mechanical Engineering
Seoul National Univ.
2013-23082
yongseoklee@snu.ac.kr

1 Introduction

본 리포트에서는 'Topics in Comuter and VLSI' 강좌에서 진행한 기말 프로젝트 Kaggle Competition : Kobe Bryant Shot Selection 의 접근 방법과 결과에 대해서 다룬다. Kobe Bryant Shot Selection 는 코비 브라이언트가 20여년간의 선수생활을 마치고 은퇴하기 까지 기록한 모든 골에 대한 데이터를 기반으로 한다. 이 데이터 (골의 성공여부, 남은 시간, 슛 위치, 슛 타입 등등)을 머신러닝을 통해 결과가 비공개된 골의 성공여부를 얼마나 잘 예측하는지가 이 competition의 목표이다.

이에 우리 팀에서는 이를 위해 다양한 방법을 적용해 보았고, 최종적으로 **1) feature selection** 과 학습 알고리즘 중 **2) Xgboost** 를 통해 모델을 학습하였다. 그 결과 0.59958의 log-likelihood loss값을 얻어, 전체 32등 강좌 2등의 좋은 성적을 거두었다.

2 Methods

우리 팀에서 성능 개선을 위해 노력한 부분은 주어진 데이터를 어떻게 가공하여 어떤 feature를 사용할것인지와, 어떤 learning algorithm을 사용하고 파라미터를 정할것인지이다. 이에 본 절에서는 이 두가지를 나누어 우리 팀에서 접근한 방법을 설명하고자 한다.

2.1 Feature Selection

Kaggle에서 주어진 raw data의 경우 슛 당 총 24차원의 벡터로 표현 되는데, 이 중 쓸모 없는 피쳐들을 제거하여 러닝 속도와 정확도를 올리려 하였다. 우선 shot id는 인덱스일 뿐이고, matchup은 opponent와 같은 정보를 갖고 있으므로 제거하였다. 또한 모두 같은 값을 같는 team name과 team id도 제거하였고, 중복되는 의미를 갖는 season, game id와 game date, distance와 loc x, loc y도 통합하여 사용하였다.

이렇게 feature의 차원을 줄인 후에는 feature processing을 통해 더욱 의미있는 feature가 되도록 수정하였다. 우선 loc x, loc y의 경우는 슛을 쏘는 위치의 값인데, 이를 좀더 직관적이고 골 성공률과 관련이 깊은 각좌표계의 distance와 angle로 아래와 같이 바꾸어서 사용하였다.

$$r = \sqrt{x^2 + y^2} \qquad \theta = atan2(y, x) \tag{1}$$

또한 minutes과 seconds도 통합하여 남은시간을 초 단위로 환산하였다. 이와 관련해 중요하다고 생각하는 feature - last moment 를 추가하였다. 이는 남은시간이 linear하게 중요한것이 아니라, 시간이 촉박할수록 이것이 심리적인 요인 및 과감한 슛 시도에 영향을 줄것이라는 직관에서 나온 feature이고, 남은시간이 3분이하일 경우에는 1, 아니면 0으로 처리하였다.

game date의 경우 각각의 날짜를 별개의 값으로 지정하기 보다는, 이를 년과 월로 쪼개어서 feature로 만들었는데, 이는 그 해 (시즌)의 퍼포먼스가 다를 것이라는 직관과 더불어 계절 및 기후적인 요인(코비가 좋아하는 계절이라던가)도 작용할 것이라는 직관에서 비롯되었다.

또한 naive bayesian등을 시도 하였을 때 성공률에 가장 큰 영향을 주는 data가 action type으로 관측 되었다. 그러나 matlab에서 unique등의 함수를 쓰면 임의의 index가 action type마다

부여되는데, (e.g. dunk : 1, layup : 2, jump shot : 3) 이는 shot type이라는 중립적이고, 대소비교를 할 수 없는 feature에 임의의 크기의 숫자를 부여 함으로써 정확도에 좋지 않다. 이렇게 classification등 discrete한 feature나 label을 다룰때에는 따라서 각 label을 별개의 차원으로 보고, 그 class에 해당할때는 그 차원의 값만 1, 나머지는 0으로 지정하는 방법이 일반적으로 사용 된다. (e.g. dunk : (1,0,0), layup : (0,1,0), jump shot: (0,0,1)) 위와 같이 feature engineering을 통해 feature들을 가공하였고, 따로 떼어놓은 test set을 통해 성능이 향상되는지를 검증 한 후 사용하였다.

2.2 Learning Algorithm

Learning algorithm으로써 우리팀은 여러가지를 시도해 보았는데 우선 naive bayseian을 사용하여 보았으나 가장 최악의 결과를 보였다. 그러나 이는 각 feature의 슛성공률에 대한 확률을 보여줌으로써 feature selection에 많은 도움이 되었다. 이 밖에 neural network을 이용하여도 학습하였으나 역시 만족할만한 결과를 얻지 못하였다.

이에 우리팀이 찾은 가장 최상의 방법은 gradient boosting을 통해 Learning 하는 것이었다. gradient boosting은 ([1],[2]) weak classifier들을 decision trees로써 조합하여 최상의 classification 결과를 얻어내는 boosting의 일종으로써 이때에 differentiable loss function을 설정하고, 이를 iterative functional gradient descent algorithms으로 최적화 하는 것이다. 이를 통해 여러 hypothesis인 weak function들로 이루어진 function space에서 loss를 최대로 줄이는 방향의 gradient로 update함으로써 서로 다른 weak classifier들의 가중치와 ensemble 방법이 결정 된다.

이를 수식으로 표현하면 다음과 같다. target function을 F, target label과 예측값 간의 loss function 을 $L(y, F(x))$ 라 할때 Learning의 결과는 다음과 같은 함수를 찾는 것이다.

$$F^* = argmin_F E(L(y, F(x)) \tag{2}$$

이때 gradient boosting은 일반적인 boosting과 마찬가지로 weak classifier $h_i(x)$의 weighte sum으로 다음과 같이 나타나진다.

$$F(x) = \sum_{i=1}^{M} \gamma_i h_i(x) \tag{3}$$

이 function space에서 정의된 decision tree의 boosting 함수는 steepest descent 방식에 따라 loss를 다음과 같이 minimize해간다.

$$\gamma_{k+1} = argmin \sum_{i=1}^{n} L(y_i, F_k(x_i) - \gamma_k \frac{\partial L(y_i, F_k(x_i))}{\partial f(x_i)}) \tag{4}$$

$$F_{k+1}(x) = F_k(x) - \gamma_{k+1} \sum_{i=1}^{n} \nabla_f L(y_i, F_k(x_i)) \tag{5}$$

위와 같은 학습과정을 통해 최종 target에 대한 weak classifier의 앙상블 $F(x)$를 구하였고, 이를 사용하였을때 가장 최상의 결과를 얻을 수 있었다.

3 Results

앞 절에서 제안된 방법들을 실제 데이터를 토대로 Learning하여 보았다. 이때 outsample error또한 측정하기 위해 전체 약 2만 5천여개의 데이터중 5천개 가량을 따로 test set으로 지정하여 out sample error를 측정하였다. 이때 naive bayesian과 neural network는 matlab의 툴박스와 라이브러리를 이용하여 구현 하였으며, 이때 neural network의 hidden layer dimension은 10, nodes는 20으로 두고 학습하였다. Gradient boosting은 xgboost[3] 라이브러리를 사용해 파이썬 상에서 구현하였다. 이때의 결과를 비교한 값은 다음과 같다.

보이는 바와 같이 gradient boosting이 다른 결과들 보다 insample error뿐 아니라 test set에 대해 뛰어난 out sample error를 보였고, 실제 kaggle 제출 결과도 좋게 나옴이 확인되었다. 특히 computation의 측면에서도 neural network 같은경우 수렴까지 4분 30초정도의 시간이 걸리고 performance도 안좋은 반면, gradient boosting은 약 6초 정도만에 수렴이 완료되고, 이때

Table 1: Comparison of the learning algorithms

Algorithms	Insample error	Outsample error
Naive bayesian	0.6211	0.6321
Neural network	0.5972	0.6097
Gradient Boosting	0.5834	0.6031
Gradient Boosting (after feature processing)	0.5801	0.5982

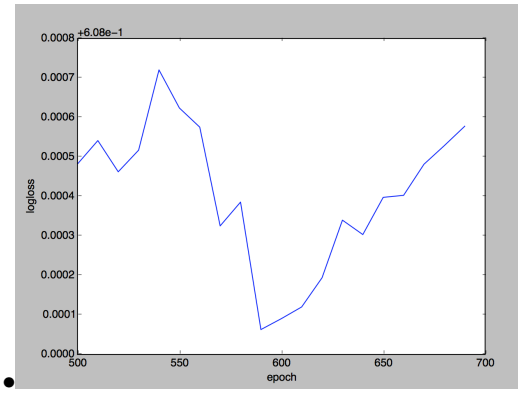

Figure 1: Graph of epoch versus logloss.

iteration epoch도 1000번 가량이 가능한것으로 볼때 주어진 문제에 적합하고 가벼운 알고리즘으로 보인다.

이때 위의 그래프에서 보이는것처럼 epoch가 590일때가 loss가 가장 작게 나타나는데, 이는 트레이닝 횟수가 일정이상 될 overfitting이 일어나 test set을 통한 outsample error가 커져서 임을 확인할수 있었다.

References

[1] MLA Breiman, Leo. Arcing the edge. Technical Report 486, Statistics Department, University of California at Berkeley, 1997.

[2] Mason, Llew, et al. ”Boosting algorithms as gradient descent in function space.” NIPS, 1999.

[2] XGBoost, https://xgboost.readthedocs.io/en/latest/.