Математички факултет
Универзиетет у Београду

Learning to Rank −RankNet

Јелена Поповић, индекс 1059/2019

Никола Жежељ, индекс 1058/2019

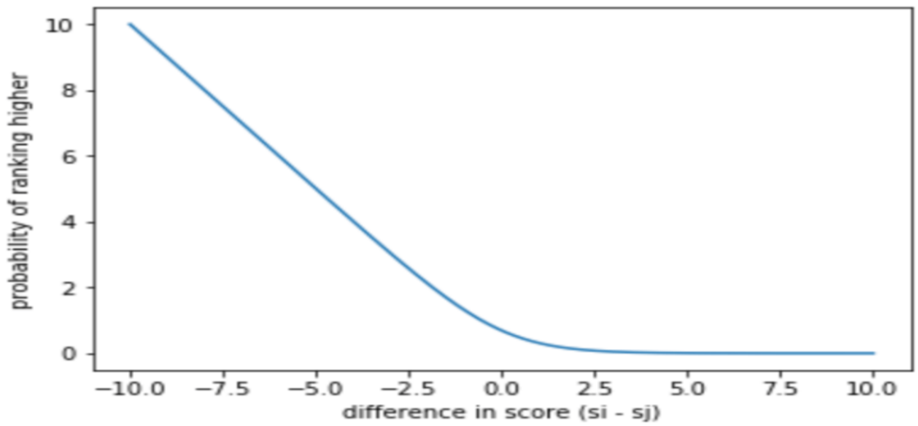
Мотивација

Иако су најчешћи задаци надгледаног учења класификација и регресија, некад наш проблем не можемо формулисати на овај начин. Нпр. уколико желимо да правимо системе најважнијих вести или препорука за куповину некаквих предмета. У овим ситуацијама желимо да знамо више од вероватноће да корисник клинке на понуђен чланак за читање или купи предмет, желимо да одредимо приоритете и наручимо чланке и предмете, да би максимизирали шансу да корисник кликне на чланак или купи предмрт. Ово је заправо чест случај, да користимо машинско учење да увећамо људске одлуке, а један ефикасан начин за то је рангирање ствари, где је циљ фокусирати људску пажњу на битне ствари. Иако се класификација и регресија могу користити за решавање проблема рангирања, ми ћемо показати директно учење рангирања као интуинтивнији приступ.

LearningtoRank је приступ проблему рангирања који учи директно да рангира тренирајући модел да предвиђа вероватноћу да је одређена ставка рангирана изнад друге. То се ради *scoring* функцијом где више рангиране ставке имају веће вредности ове функције. Модел се тренира градијентним спустом на функцију губитка дефинисану преко вредности *scoring* функције. За сваку ставку градијентни спуст повећава скор за све ставке које су рангиране испод посматране, и смањује скор за све ставке изнад посматране, пропорционално разлици између скорова посматраних ставки. Да би се осигурали да модел добро рангира ставке већег скора, можемо додати тежине које утичу на повећање или смањење скора код градијентног спуста.

Основна идеја

Основна идеја је да имамо модел који узима две ставке са улаза и предвиђа како ће их рангирати. Ово се једноставно постигне, тако што обучимо модел да додељује скорове ставкама и онда оне са већим вредностима бивају рангиране више у односу на оне са мањим вредностима функције скора. Једино својство које нам је важно је диференцијабилност модела да бисмо могли да приенимо градијентни спуст. Ваља приметити да овај приступ можемо користити у ситуацији кад имамо недоследна рангирања и кад све ставке нису нпр. упоредиве. функција губитка одређена је принципом максималне веродостојности.Природан начин да користимо принцип максималне веродостојности је да моделујемо вероватноћу да је нека ставка рангирана изнад друге преко скор функције. Користићемо вероватноћу да модел рангира сваки пар ставки добро за функцију губитка. Скор за ставку i означимо са s_i. Један начин да искористимо скорове за вероватноћу јесте да користмо сигмоидну функцију на разлици скорова, тј. $P(rank(i) > rank(j)) = \frac{1}{1+e^{-(s_i - s_j)}}$ Погледајмо сада како изгледа функција губитка. За један пар губитак ће бити дефинисан на следећи начин: $J_{ij} = -\log(\frac{1}{1+e^{-(s_i-s_j)}}) = \log(1 + e^{s_j - s_i})$. Погледајмо то на слици.



Укупна функција губитка је сума по свим паровима $(i, j) \in D$, где је ставка i више рангирана од j:

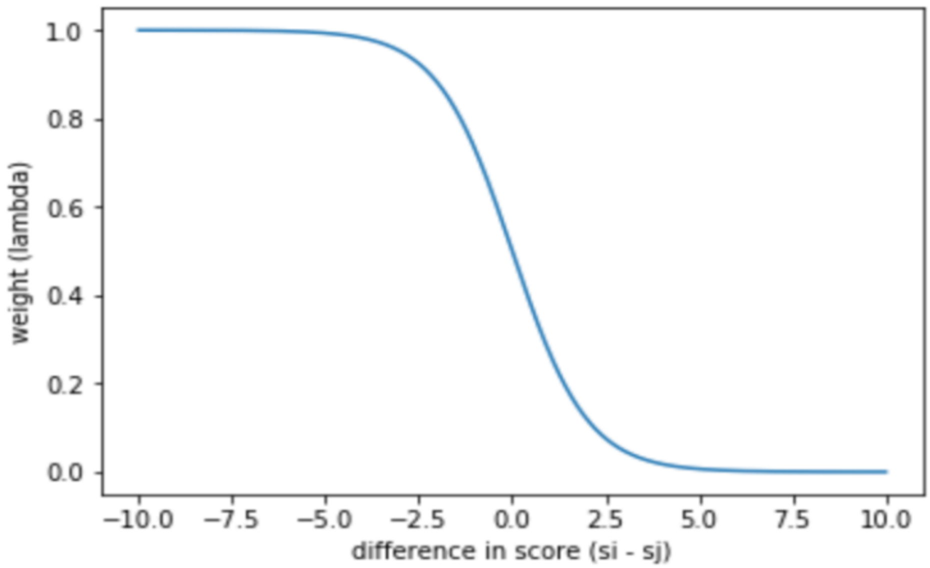
$$J = \sum_{(i,j) \in D} J_{ij}$$

На ову функцију ћемо примењивати градијентни спуст. Анализирајмо шта се заправо дешава кад применимо градијентни спуст. Посматрајмо за један пар ставки извод по неком параметру:

$$\frac{\partial J_{ij}}{\partial \Theta_k} = \frac{\partial J_{ij}}{\partial \ s_i}\frac{\partial \ s_i}{\partial \Theta_k} + \frac{\partial J_{ij}}{\partial \ s_j}\frac{\partial \ s_j}{\partial \Theta_k} = \frac{-e^{s_j - s_i}}{1 + e^{s_j - s_i}}\frac{\partial \ s_i}{\partial \Theta_k} + \frac{e^{s_j - s_i}}{1 + e^{s_j - s_i}}\frac{\partial \ s_j}{\partial \Theta_k} = \frac{-e^{s_j - s_i}}{1 + e^{s_j - s_i}}(\frac{\partial \ s_i}{\partial \Theta_k} - \frac{\partial \ s_j}{\partial \Theta_k})$$

$$= \frac{-1}{1 + e^{s_i - s_j}}(\frac{\partial \ s_i}{\partial \Theta_k} - \frac{\partial \ s_j}{\partial \Theta_k}) = \lambda_{ij}(\frac{\partial \ s_i}{\partial \Theta_k} - \frac{\partial \ s_j}{\partial \Theta_k})$$

Овде је $\lambda_{ij} = \frac{-1}{1+e^{s_i - s_j}}$. Како је λ_{ij} негативан ми заправо градијентним спустом при оптимизацији користимо растући градијент s_i и опадајући s_j, помноженени са одговарајућом тежином,што нам и даје заправо објашњење да кроз градијентни спуст функције грешке, ми померамо тежине тако да повећамо скор за ставке које су више рангиране и смањимо скор онима ниже рангираним, са одређеним фактором тежине. Погледајмо сада λ_{ij} на слици



Видимо да λ_{ij} је већа за парове где је нижи релативан резултат бишег ранга у односу на нижи ранг. Градијент за целу функцију губитка је

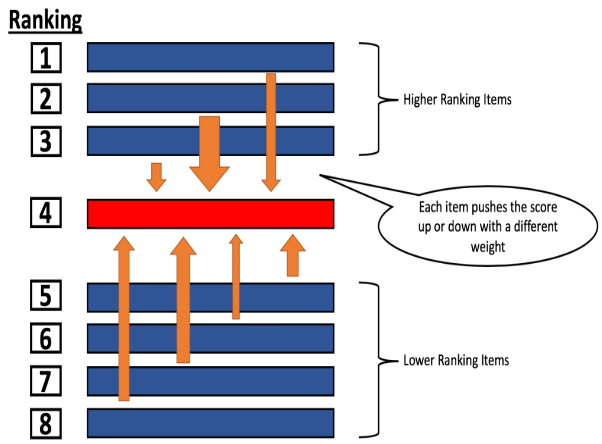
$$\frac{\partial J}{\partial \theta_k} = \sum_{(i,j)\in D} \frac{\partial J_{ij}}{\partial \theta_k} = \sum_{(i,j)\in D} \lambda_{ij}(\frac{\partial s_i}{\partial \theta_k} - \frac{\partial s_j}{\partial \theta_k})$$

Посматрајмо сада проблем из перспективе индивидуалних ставки а не парова што смо до сада разматрали.Градијент грешке можемо записати

$$\sum_{(i,j)\in D} \lambda_{ij}(\frac{\partial s_i}{\partial \theta_k} - \frac{\partial s_j}{\partial \theta_k}) = \sum_{(j|(i,j)\in D)} \lambda_{ij}\frac{\partial s_i}{\partial \theta_k} - \sum_{(i|(i,j)\in D)} \lambda_{ij}\frac{\partial s_j}{\partial \theta_k} = \sum_{l}(\sum_{(j|(l,j)\in D)} \lambda_{lj} - \sum_{(i|(i,l)\in D)} \lambda_{il})\frac{\partial s_l}{\partial \theta_k}$$

Ова једначина нам даје бољи увид у то шта заправо радимо. За сваку ставку која се рангира постоје две силе, једна која гура скор ка доле, друга ка горе, ка доле скор гура сила јачине сразмерне разлици у скоровима за ставке рангиране изнад посматране, а ка горе сила сразмерна разлици у скоровима у односу на ставке рангиране испод посматране.

Претходно разматрање је приказано на следећој слици.



Побољшања учења рангирања: LambdaRank метода

Претходни приступ коришћен у *RankNet* методи је добар у многим ситуацијама, међутим има велики недостатак:губитак ће бити исти за било који пар ставки i и j,без обзира на ком месту су рангирани.
Шта ово практично значи? Како је раније поменуто скор сваке ставке бива гуран навише од стране сваке ставке нижег ранга, односно гуран наниже од стране сваке ставке вишег ранга. Јачина овог гурања заснована је само на тренутној разлици у резултатима. Ово значи да ће оне ставке рангиране 1, 2 или 500 рангова ниже од тренутне ставке имати исти утицај на њен скор ,све док саме имају исти скор. Дакле *RankNet* брине само о укупном броју ранкова по паровима. Замена 1. и 10.
суштински је једнака замени 101. и 110. места, уколико су исте разлике у скоровима, једино је важно да се рангирање померило за 10 места. Ово је проблем у
ситуацији када нас интересују само највише рангиране ставке,што је чест случај. Решење за превазилажење овог проблема лежи у отежавању градијента функције губитка фактором који наглашава важност тачности рангираног пара. Дакле, уместо

$$\lambda_{i,j} = \frac{-1}{1 + e^{s_i - s_j}}$$

користићемо

$$\lambda_{i,j} = \frac{-|\Delta(i,j)|}{1 + e^{s_i - s_j}}$$

Вредност $\Delta(i,j)$ представља казну за погрешно рангирање ставки i и j .
Разне метрике се користе за Δ, у пракси најчешће
$NDCG$ (енг. *Normalized discounted cumulative gain*). $NDCG$ представља

нормализацију DCG метрике.

$$DCG = \sum_{r=1}^{N} (2^{relevance(r)} - 1) \times weight(r) = \sum_{r=1}^{N} \frac{2^{relevance(r)} - 1}{\log(r+1)}$$

Где N представља број ставки које су нам од значаја,$relevance(r)$ важност, а $weight(r)$ одговарајућу придружену тежину
Важност (r) је важност сваке ставке понаособ. Што је ставка боље рангирана и важност јој је већа.Ова сума је сумирање ранкова које је наш модел предвидео, не стварних ранкова.
Поједностављено, ДЦГ награђује позиционирање "добрих"ставки на више позиције (већи ранг) и додељује већу тежину за коректно рангиране више ставке.
$NDCG$ је нормализација овога, односно добија се дељењем DCG са максимумом који DCG може добити на подацима (то би био резултат $DCG - a$ са савршеним рангирањем).

$$NDCG = \frac{DCG}{maxDCG}$$

Разлог за чешће коришћење $NDCG$ метрике је чињеница да DCG мпже бити произвољно велика зависно од значајности придруженој свакој ставки.
Овако уведена метрика доводи до тога да већу тежину имају ставке којима модел додељује виши ранг.А управо нам је то било потребно да превазиђемо недостатке $RankNet$ приступа.
Емпиријски је показано да овакав приступ (првобитно представљен као $LambdaRank$) оптимизује $NDCG$. Али наравно овакав приступ је применљив на било коју другу меру квалитета рангирања , нпр. MAP (енг. $MeanAveragePrecision$).

Скуп података

Скуп података које смо користили је скуп из пакета $LETOR$ 4.0. Овај пакет садржи скупове података за рангирање. Ова верзија пакета доступна је од 2009. године и користи колекцију $Gov2$ веб страница и скупове упита из $Million$ $Query\ Track$,ткзв. $TREC$ 2007 и $TREC$ 2008. Они су означени са $MQ2007$ и $MQ2008$. $MQ2007$ садржи око 1700 упита,док $MQ2008$ садржи око 800 упита са документима. $LETOR4.0$ садржи 8 скупова података за различита рангирања, подељених у два скупа упита и $Gov2$ колекције страница. Сваки од скупова садржи 5 фолдера у сваком постоје фајлови за тренинг, валидацију и тест. Ми користимо $MQ2008$ скуп за надгледано учење.Скуп је тако генерисан да је дат као скуп парова документ-упит, по врстама.Прва колона је ознака релевантности пара, што нам је и циљна вредност, друга је id упита, затим постоји 46 колона које су атрибути који описују пар документ-упит, након чега следи колона која садржи id документа и коментар о посматраном пару. Листа атрибута садржи карактеристике односа упита и документа, први атрибут је фрекфенција појављивања речи из упита у телу документа.Сви су нумеричког типа. Следи пример пара документ-упит:

0 qid : 18219 1 : 0.052893 2 : 1.000000 3 : 0.750000 4 : 1.000000 5 : 0.066225 6 : 0.000000 7 : 0.000000 8 : 0.000000 9 : 0.000000 10 : 0.000000 11 : 0.047634 12 : 1.000000 13 : 0.740506 14 : 1.000000 15 : 0.058539 16 : 0.003995 17 : 0.500000 18 : 0.400000 19 : 0.400000 20 : 0.004121 21 : 1.000000 22 : 1.000000 23 : 0.974510 24 : 1.000000 25 : 0.929240 26 : 1.000000 27 : 1.000000 28 : 0.829951 29 : 1.000000 30 : 1.000000 31 : 0.768123 32 : 1.000000 33 : 1.000000 34 : 1.000000 35 : 1.000000 36 : 1.000000 37 : 1.000000 38 : 1.000000 39 : 0.998377 40 : 1.000000 41 : 0.333333 42 : 0.434783 43 : 0.000000 44 : 0.396910 45 : 0.447368 46 : 0.966667 $\#docid = GX004 - 93 - 7097963$ $inc = 0.0428115405134536$ $prob = 0.860366$

Модели за решавање

За решавање проблема користили смо два модела. Први модел који смо користили је $lightgbm.LGBMRanker$. $LightGBM$ је скраћеница од *Light Gradient Boosting Machine*, и алгоритми које користи су засновани на на градијентном појачању и користе стабла одлучивања. $lightgbm.LGBMRanker$ је модел који користи за функцију губитка ткзв. *LambdaRank* функцију коју смо разматрали. Други модел који смо разматрали је $xgboost.sklearn.XGBRanker$, библиотеке $xgboost$ која се заснива на градијентном појачању.И овај модел као претходни рангира ставке засноване на идеји *LambdaRank*.

Анализа резултата
За евалуацију модела користили смо поменуту $NGCD$ метрику. Резултат са коришћењем првог модела био је 0.8059293618324704, док у другом случају када смо све инстанце ставили у исту групу добили смо бољи резултат: 0.8123978788530549, али по цену дужег времена тренирања модела. Решење истог проблема функционалностима *XGBoost* библиотеке које смо приказали на крају даје вредност $NGCD$ метрике 0.7755382530421017, што је знатно слабије од претходна два резултата. Дакле, на основу реченог можемо закључити да ћемо у зависности од обима података и вреенских ресурса изабрати један од модела базираних на $LightGBM$ фрејмворку.

Литература

[1] https://mlexplained.com/2019/05/27/learning-to-rank-explained-with-code/