Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Рачунаркса техника и софтверско инжењерство

Ниво студија: Основне академске студије

Предмет: Софтверски инжењеринг 2

Број индекса: 587/2015

Јован Ж. Анђелковић

Развој система за подршку одлучивању приликом избора успешних фудбалских играча

Дипломски рад

Комисија за преглед и одбрану:

1. **Др. Велибор Исаиловић, доцент - ментор**
2. _____
3. _____

Датум одбране:_____

Оцена:_____

У оквиру дипломског рада „*Развој система за подршку одлучивању приликом избора успешних фудбалских играча*“ кандидат треба да:

Направи модел за подршку одлучивању приликом избора успешних фудбалских играча применом техника машинског учења. Као улазни подаци треда да буде коришћена реална статистика фудбалских играча. Осим тога, потребно је направити графичку корисничку апликацију за приказ статистике и предвиђања за наредну утакмицу.

Крагујевац, октобар 2019.

Ментор:
Др. Велибор Исаиловић, доцент

_____

САДРЖАЈ

Summary

The thesis is about the use of machine learning techniques and algorithms in order to accurately predict the performance of football players in succeeding football matches. Machine learning refers to creation and training of an adequate neural network. The implementation of these ventures is distinctly defined by rules of this scientific field.
Real statistics of football players is used for the training of the network. After training, based on its knowledge, the neural network can classify a player as “valuable for the first team lineup” or “not good enough”.

Key words: Performance prediction, machine learning, neural network, real player statistics

Резиме

Тема овог дипломског рада је примена техника и алгоритама машинског учења у циљу што прецизнијег предвиђања учинка фудбалских играча у наредним утакмицама. Под појмом машинског учења мисли се на креирање и обучавање адекватне неуронске мреже. Притом се користе јасно дефинисана правила ове научне области.
За обуку/тренирање мреже користе се подаци преузети из реалне статистике фудбалских играча, који су зарад квалитетнијих резултата прилагођени улазној логици неуронске мреже. Након обуке, неуронска мрежа на основу свог знања може да класификује играча као „корисног за прву поставу“ или „недовољно добар“.

Кључне речи: Предвиђање успешности, машинско учење, неуронска мрежа, реална статистика играча

Предговор

Главни разлог зашто сам одабрао баш ову тему је моје велико интересовање за научну област *машинско учење*. Сматрам да ће ова научна област направити велику промену читавог света и зато желим да наставим моје школовање и усавршавање у овом правцу. Писање рада на ову тему ми је омогућило да боље упознам и проучим методе и технике које се користе у машинском учењу. Још један разлог за одабир ове теме су личне симпатије које имам према интернет игрици „Fantasy Premier League“.

Највећу захвалност дугујем ментору Доц. др Велибору Исаиловићу на избору теме, труду и залагању при реализацији рада. Избором теме ми је омогућио да објединим своја интересовања према машинском учењу и према омиљеној игрици. Посебно бих се захвалио асистенту Бојани Анђелковић-Ћирковић која ми је значајно помогла при учењу алгоритама који су примењени у дипломском раду и чије су идеје учиниле овај рад озбиљнијим.

1. Увод

Систем за подршку одлучивању приликом избора успешних фудбалских играча чини неуронска мрежа. Да би се развио прецизан систем, потребно је прво „научити“ мрежу да направи разлику између успешног и лошег играча. За учење неуронске мреже се могу користити различите технике машинског учења.

У овом раду се примењује техника „Надгледано учење“ (Supervised learning) за обуку. Како би неуронска мрежа прорачунала математичку функцију одлучивања, користе се подаци који су груписани у парове „улаз-излаз“ (input-output). Улазне вредности су атрибути, односно реална статистика играча, док излазна вредност представља одговор на питање „Да ли је играч успешан?“.

Генерално посматрано, класификација представља предвиђање класе којој одређени објекат припада. Ради поједностављења проблема класификације фудбалских играча и у циљу развијања оптималних перформанси, играчи су подељени у две класе. Прва класа је „нулта класа“ (класа 0), којој припадају сви играчи који не испуњавају критеријуме успешног играча. Са друге стране, успешни играчи припадају класи 1. Због наведене поделе класа, излазна вредност може бити само нула или јединица, што значи да ова неуронска мрежа припада групи бинарних класификатора.

За тренинг податке се користе играчи за које се већ зна класа којој припадају. Мрежа на основу улазних атрибута који су груписани у вектор (матрицу колону), анализира и мапира излазне вредности на графику функцију. Након дефинисаног броја итерација, мрежа израчунава укупну грешку која је направљена при класификацији и проверава да ли је могуће икакво побољшање резултата. На тај начин покушава да добије оптималну функцију која прави најмању укупну грешку. Овај процес се понавља изнова све док се не догоди конвергенција функције ка глобалном минимуму.

Неуронска мрежа кроз овакав вид учења тежи да генерализује податке који су доступни у току тренинга. Циљ је добијање што веће прецизности при класификацији нових фудбалских играча, чије статистичке вредности до тада нису познате неуронској мрежи.

Да би се извршило обучавање система, потребни су подаци за тренинг и тестирање. Коришћена је реална статистика фудбалских играча из Енглеске, односно из „Премијер лиге“. За извор статистичких података користи се сајт интернет игрице „Fantasy Premier League“ https://fantasy.premierleague.com/. За имплементацију система се користи платформа „Spyder“ у оквиру развојног окружења „Anaconda“. Систем се развија применом програмског језика *Python* као и различите библиотеке овог језика, које ће бити објашњене у неком од наредних поглавља.

2. Машинско учење

Машинско учење је подобласт вештачке интелигенције, која чини део велике научне области „Рачунарске науке“ (Computer science). Овај део вештачке интелигенције је показао огромна побољшања у задњих пар деценија која су приказана кроз велики број креираних система машинског учења у различитим областима: индустрија, медицина, економија, финансије, природне науке итд... Ипак, треба нагласити да постоји разлика између машинског учења и традиционалног „AI“ (Artificial Intelligence). Машинско учење не развија аутоматска ограничења интелигентног понашања, већ покушава да унапреди предност и способност рачунарских система како би се употпунила људска интелигенција. Основни разлог побољшања ове области јесте принцип функционисања. То је аутоматско моделирање дефинисаних процеса на основу прикупљених података.

Учење података се може манифестовати кроз нова правила, функцију, систем једначина, кроз вероватноћу расподеле и томе слично. Креирани модели даље могу користити податке за доношење одлука, дијагностиковање, предвиђање, валидацију или симулацију. Овакав принцип функционисања се користи за анализу података, аутоматско генерисање база података за експертске системе, конструисање нумеричких модела, класификацију текста, аутоматско препознавање говора, рукописа, слика и за многе друге потребе.

Као интердисциплинарна област, машинско учење има додирних тачака са статистиком, теоријом информација, теоријом игара и оптимизацијом. Компонента искуства или обуке у машинском учењу се често односи на податке који су генерисани насумично. Задатак „супервизора“, који надгледа обучавање система, је обрада насумично генерисаних примера како би се извео закључак који важи за окружење којем припадају примери. Ова карактеристика је заједничка за статистику и машинско учење.

Међутим, постоји неколико значајних разлика између ове две дисциплине. Када доктор постави хипотезу да постоји повезаност између пушења цигарета и срчаних обољења, тада статистичари проверавају историју пацијената како би се доказала та хипотеза. Са друге стране, машинско учење користи прикупљене податке из историје како би се пронашла нека сличност или шаблон које нису запазили доктори. Резултат ове претраге би било откривање узрока срчаних обољења. У машинском учењу се све базира на „обучавању“ рачунара кроз алгоритме и на ефикасности тих обука. Још једна разлика је та што се статистика често фокусира на асимптотско понашање, као што је конвергенција статистичких процена заснованих на узорку, док величина узорка расте. Теорија машинског учења се више фокусира на крајње границе узорака. Наиме, у зависности од величине доступних узорака, тежи се ка утврђивању степена тачности предвиђања који се може очекивати на основу таквих узорака. Док се у статистици уобичајено ради према већ неком дефинисаном моделу података, у машинском учењу је нагласак на почетном раду са што мање дефиниција и што мање претпоставки о расподели података. То омогућава алгоритму учења да временом утврди који модел најбоље одговара процесу генерисања података.

Постоје два разлога због којих се тежи ка решавању проблема коришћењем машинског учења:

1. Проблем је исувише компликован за стандардно програмирање
2. Прилагодљивост

Људи за већину проблема и задатака које решавају рутински тврде да раде по осећају, који не могу другачије да објасне. Задаци попут препознавање говора, разумевање слика или вожње аутомобила се не могу егзактно програмирати, јер у овим примерима људи користе своја знања из претходног искуства у комбинацији са својим чулима. Предност машинског учења је баш у томе што програм може да учи из тренинг примера, који представљају туђа искуства.

Са друге стране, постоји и група проблема која су изнад људских могућности, а притом се могу решити машинским учењем. Примењивање медицинске архиве/статистике у дијагнози пацијента, временска прогноза, и проучавање великог броја астрономских података су само неки од примера проблема који су превише комплексни или преобимни за људске способности. Захваљујући великом броју података који су дигитално сачувани, могу се креирати програми који би се обучавали за ову проблематику. Предност „увежбаног“ програма у односу на човека је што програм кроз учење може да препозна шаблоне који се понављају у огромним и комплексним скуповима. Уколико се програму придодају велика меморија за чување свих података и рачунари са брзим процесорима, добија се значајно унапређење.

Што се тиче прилагодљивости, мисли се на апликације које применом машинског учења добијају на флексибилности. Углавном програми након инсталирања остају непроменљиви током читавог периода коришћења. С обзиром да их користе различити клијенти, постоји потреба за модификацијом функционалности. Као пример успешног „учења“ апликација могу се навести програми за декодирање текста који је ручно написан на папиру, где се апликација адаптира на рукописе различитих корисника. Механизми за детектовање нежељених мејлова (spam e-mails) се у току коришћења аутоматски адаптирају на нове трендове писања ове врсте мејлова. Исто важи и за програме који служе за препознавање говора.

Као што се може закључити, ова област се бави креирањем алгоритама који раде са великим бројем примера везаних за неки феномен, у циљу решавања практичних проблема. Примери који се користе су из реалног света, креирани од стране човека или их може генерисати неки други алгоритам. Без обзира на област којој припада проблем, процес његовог решавања кроз машинско учење се дели на два корака:

1. Прикупљање података
2. Развој статистичког модела који се заснива на прикупљеним подацима

Постоје четири врсте учења у процесу тренирања:

- Надгледано учење (Supervised learning)
- Ненадгледано учење (Unsupervised learning)
- Полу надгледано учење (Semi-Supervised learning)
- Учење са подршком (Reinforcement learning)

2.1 Надгледано учење

Код ове врсте машинског учења скуп података представља колекција означених примера. Сваки пример садржи такозвани „**вектор карактеристика**“, где свака његова ставка садржи вредност која описује пример на неки начин. Та вредност се назива атрибут објекта. На пример, ако вектор карактеристика X представља пример једне особе, онда први атрибут $x^{(1)}$ може да садржи висину особе изражену у центиметрима, други атрибут $x^{(2)}$ може да представља тежину особе изражену килограмима итд. За све примере у скупу података, атрибут на одређеној позицији у вектору увек садржи исту врсту информације о примеру. То значи да ако $x_к^{(1)}$ садржи вредност тежине изражене килограмима у примеру $x_к$, онда ће и $x_j^{(1)}$ такође представљати вредност у килограмима. То важи за све примере $x_к$, где је к = 1,…, N.

Ознака y_i може да представља број класе којој пример припада {1,2,…,C}, може да буде реалан број или чак и нека комплекснија структура попут вектора, матрице, стабла или графа. Класе су уствари категорије на које су подељени примери у скупу података. На пример, ако је реч о алгоритму који се користи за детектовање нежељених мејлова (spam), онда се примери деле на две класе: мејлови који јесу „spam“ и мејлови који нису.

Идеја овог вида учења је коришћење скупа података за развој модела који узима вектор карактеристика као улаз, а даје информације којима се може закључити класа којој пример припада. На пример, ако је модел креиран на основу скупа података различитих особа где вектор карактеристика даје физички опис особе, излазна вредност је вероватноћа да особа има рак.

2.2 Ненадгледано учење

При учењу без надгледања се користи скуп података чији примери нису означени. Као и код учења са надгледањем, користи се вектор карактеристика. Циљ овог алгоритма је креирање модела који користи векторе карактеристика X као улазну вредност и то претвара у неки други вектор или вредност која је адекватна за решење одређеног проблема.

Стандардан пример за ову врсту учења је проблем „кластеринг“ (clustering). Користи се велики број необележених података који се састоје само од вектора карактеристика. Процесом проналажења смислене структуре, различитим процесима и генералисањем карактеристика алгоритам покушава да групише податке у различите скупове. Подела се врши тако да подаци чије су карактеристике сличне припадају истој групи, а притом су другачије од података других групација. У таквом процесу, модел враћа идентификациони број „кластера“ (групе) за сваки вектор карактеристика у скупу података.

У проблему „смањења димензија“ (dimensionality reduction) излаз модела је вектор карактеристика који има мање атрибута од улазног вектора. Као пример се може узети и „детекција екстремних вредности“ (outlier detection), где је потребно пронаћи које значајно одступају од осталих запажања у подацима. Излаз код овог алгоритма при учењу без надгледања би била реална вредност која указује колико тај пример податка одступа од преосталог скупа података.

2.3 Полу надгледано учење

Као што сам назив каже, у питању је комбинација претходно наведених врста учења. За ову врсту се користе означени и неозначени примери. Углавном је много већи број неозначених примера у односу на означене у скупу. Циљ је исти као код учења са надгледањем. Разликује се само због неозначених примера у скупу који се користе у нади да ће помоћи при учењу алгоритма како би пронашао бољи модел.

Иако можда не делује логично да се користе обе врсте података како би се добило на бољим перформансама система, то није тачно. При учењу података, за алгоритам је битно да има што више података, односно што више примера. Додавањем необележених примера, алгоритам добија више информација о задатом проблему. Више узорака даје бољу вероватноћу расподеле података у односу на расподелу која се добија искључиво са обележеним примерима. Теоријски гледано сматра се да би алгоритам учења требало да може да искористи додатне информације које добија кроз необележене узорке.

2.4 Учење са подршком

Учење са подршком свој концепт базира на томе да програм/алгоритам „живи“ у окружењу чије стање константно прати кроз векторе карактеристика. Програм може у сваком стању да изврши одређени задатак. Сваки задатак са собом доноси адекватан резултат, који може да буде чак и промена стања окружења у коме се алгоритам налази. Циљ учења са подршком је да алгоритам научи „полису“ (policy). То је функција која као улазну вредност користи вектор карактеристика за стање окружења и на основу тога предлаже задатак, чије је извршење оптимално у датом стању. За задатак се сматра да је оптималан ако утиче на повећање „максималне просечне награде“ коју програм очекује.

Ова врста учења се користи при решавању специфичне врсте проблема, где је потребно секвенцијално одлучивање. Пример таквих проблема су области у којима се гледа добитак на „дуже стазе“ попут: игрица, роботике и логистике.

3. Примена машинског учења

За коришћење машинског учења при решавању проблема, треба детаљније анализирати основне принципе алгоритма, као што су моделирање података, мерење перформанси, унапређење алгоритма итд...

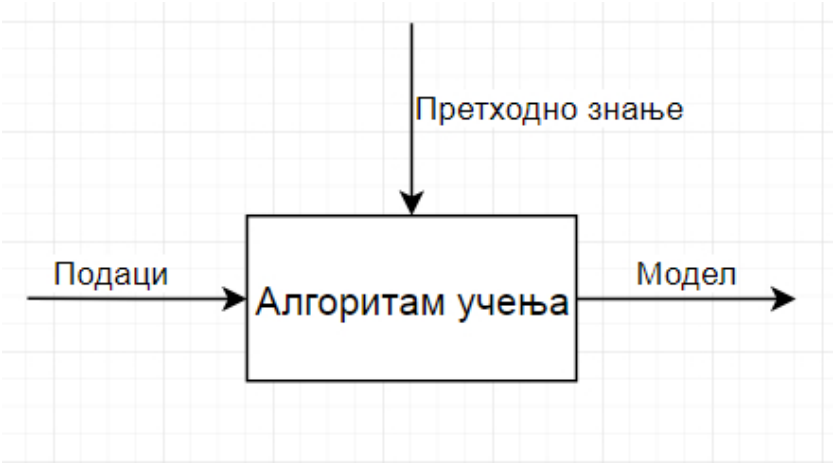
Прва и кључна ствар доброг алгоритма је сама обука алгоритма и моделирање података. Резултат учења је знање које систем користи у решавању нових задатака. Знање се може представити на неколико начина:

- Скуп већ виђених примера,
- Алгоритам за решавање одређених проблема,
- Скуп инструкција за ефикасно решавање проблема.

У машинском учењу постоји подела на „алгоритам учења“ и „извршни алгоритам“. Алгоритам учења служи за креирање нових или модификацију постојећих примера из скупа података. Генерисане податке даље користи извршни алгоритам за решавање нових задатака. Генерисано знање, односно подаци алгоритма учења, чине „модел“. По правилу би модел требао да се прилагођава претходно стеченом знању и учењу нових података.

У пракси се модел још и назива „хипотезом“ или „теоријом“. Сви улазни подаци алгоритма учења треба да буду веродостојни како би формула модела донела добре резултате. Хипотеза алгоритма учења уствари извлачи информације из улазних података које сматра кључним и представља њихову апстракцију.

Улазне вредности овог система су скуп података за обуку и претходно знање. Излазна вредност система је представљена кроз модел (хипотеза), који целокупно описује улазне вредности (Слика 1). Претходно знање садржи углавном потенцијалне моделе и критеријум оптималности алгоритма. Зато алгоритам учења тражи адекватан модел који задовољава те критеријуме. Често се дешава да претходно знање представља иницијалну хипотезу (неко приближно решење) и скуп хеуристике која се примењује при претрази адекватног модела.



Слика 1. Дијаграм алгоритма учења

Машинско учење се може посматрати и као проблем оптимизације. У односу на дата решења и критеријум оптималности, тражи се модел који задовољава услове. Вредност критеријума зависи од тренутног модела, претходног знања и учења података. Оптимално решење је често веома тешко пронаћи зато што постоји огроман број могућих решења (бесконачан). Због тога се траже приближно оптимална решења.

Постоје неколико врста хипотеза (модела), самим тим и неколико категорија проблема које решава машинско учење:

- Дискретне функције,
- Континуалне функције,
- Релације (Relations).

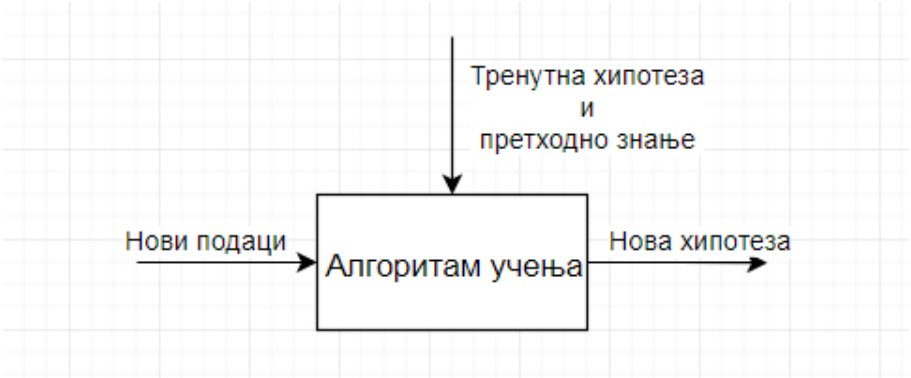
Домени дискретних функција су коначни, а скупови вредности су неуређени. Зависна променљива се зове „класа“ и њена вредност је ознака класе. Проблеми који захтевају моделе са дискретним функцијама се називају *проблеми класификације*. Када алгоритам пронађе функцију, користи је за класификацију нових примера. Често одабрана функција враћа скуп вредности уместо ознаке класе. У таквим случајевима вредности елемената у скупу представљају вероватноће. На пример, за проблем медицинске дијагностике, функција би на основу стања пацијента вратила скуп вероватноћа за потенцијална обољења. Као проблем класификације могу бити формулисани не само дијагностички проблеми, већ и проблеми предвиђања (у које спада и тема дипломског рада).

Континуалне функције имају потенцијално бесконачне домене са реалним вредностима у уређеном скупу. Циљна (зависна) променљива се назива регресиона променљива, а проблеми који се решавају коришћењем континуалних функција се називају *регресионим проблемима*. Користе се аутоматски генерисане функције за одређивање зависне променљиве на основу независних променљивих, као и код класификационих проблема. Регресијом се решавају различите врсте проблема одлучивања и предвиђања. Типични примери примене регресије су предвиђања, контролисање динамичних система као и процењивање утицаја различитих параметара на вредност зависне променљиве. У регресионим проблемима су потребни и *интервали поверења*, који квантитативно описују поузданост предложених решења.

Што се тиче „релација“, аутоматски повезују објекте који могу бити елементи или функције релације, где један или више параметара чине зависну променљиву. Број комбинација могућих релација је знатно већи од броја потенцијалинх функција, зато што су функције специфичније. Релациони проблеми учења због своје опште дефиниције могу имати више различитих исхода, али су и захтевнији у проналажењу приближно оптималних решења. Такође, количина података и претходно знање релационих проблема захтевају више претрага. Релације су прилагодљивије проблему јер могу бити континуалне (системи једначина) или дискретне (логичке везе). У ову групу проблема спадају учење структура хемијских једињења, учење геометријских облика и одређивање општих правилности у базама података.

Основни алгоритам учења на коме се базира машинско учење претпоставља да су сви подаци за учење доступни на почетку процеса. Алтернативна варијанта би била да алгоритам секвенцијално прихвата податке за учење. Тада алгоритам мења своју хипотезу (модел) након сваког прихватања нових података, тако да се прилагоде претходно добијени и нови подаци.

Процес учења у коме се врши модификација хипотезе након сваког прихватања нових података, назива се инкрементално учење (on-line learning). Најједноставнија врста инкременталног учења одбацује претходну хипотезу када добије нове улазне податке и развија нову хипотезу (Слика 2). Циљ ове врсте учења јесте да изврши што је мање могућих измена на тренутној хипотези како би одговарала свим до тог тренутка прихваћеним улазним подацима.



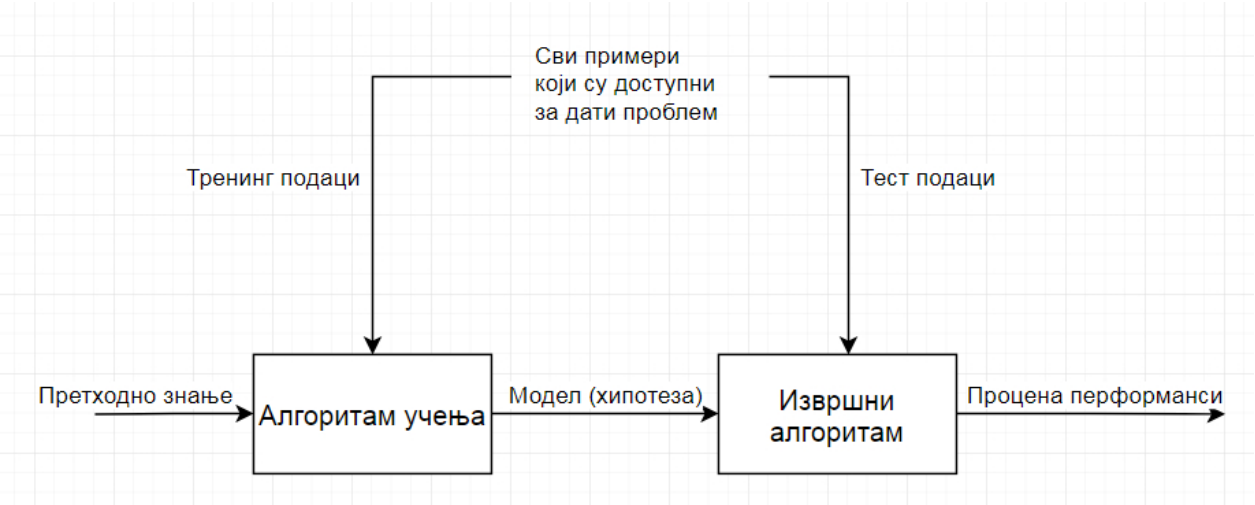
Слика 2. Дијаграм инкременталног учења

Инкрементални алгоритам у неким ситуацијама може да „заборави“ неке делове старијих података, па нове хипотезе могу бити контрадикторне тим улазним подацима. Овакви догађаји су добри у случајевима када се циљ алгоритма динамички мења, јер се на тај начин тежи ка оптималном моделу. Променом хипотезе, алгоритам конвергира ка оптималној хипотези за одређени проблем. Ипак, често се дешава да иако алгоритам добија нове улазне податке, хипотеза остане иста. Објашњење ове појаве је да нови подаци често могу да буду контрадикторни тренутној хипотези, што значи да алгоритам учења никад не зна да ли је пронашао оптималну хипотезу. Међутим, у пракси нови подаци након одређеног периода не утичу више на побољшање тренутне хипотезе.

3.1 Процена перформанси

Када се развије алгоритам за решавање проблема из одређене области, потребно је на неки начин проценити успешност алгоритма да реши нове проблеме, како би се видео квалитет. Код класификационих задатака, гледају се перформансе генерисаног модела при класификацији појединачних примера. За регресионе проблеме се прати прецизност предвиђених вредности и ниво поузданости тих предвиђања. Проценат примера који су тачно додељени одређеној релацији је кључна ствар за перформансе логичких релација.

За процену квалитета аутоматски генерисаних модела користе се доступни подаци за одређени проблем, подељени на два скупа података: тренинг и тест. Тренинг скуп се користи за алгоритам учења, док се тест подаци користе у процени квалитета (Слика 3).



Слика 3. Дијаграм процене перформанси аутоматски генерисаних модела

Потребно је гледати процену целокупног проблема, а не само процену алгоритма кроз бројање нетачних решења. То је поготово важно при доношењу одлука за проблеме попут играња игрица. Тада се успех или неуспех утврђује тек након низа извршених задатака. Слично томе, приликом контролисања динамичних система добијају се информације о успеху тек након дужег контролисања.

За процену перформанси класификационих алгоритама најчешће се користе следеће мере:

- Тачност класификације (accuracy) и матрица конфузије,
- Цена погрешне класификације (misclassification cost),
- Информативни резултати и маргине (Brier score),
- Сензитивност, специфичност, „ROC“ крива, прецизност и одзив (precision and recall).

Решење сваког класификационог примера је ознака једне класе од неколико потенцијалних класа. Тачност и матрица конфузије се углавном користе као мера квалитета класификатора. Успешност решења класификационог проблема се мери кроз тачност класификације. Поред тога, тачност се користи и за релационе проблеме под претпоставком да релација може да припада искључиво једној од две понуђене класе. Класификациона тачност је дефинисана као релативна фреквенција тачних класификација примера.

$$Acc = \frac{n_{\text{тачно}}}{n} \times 100\% \quad (1)$$

Симбол n представља укупан број примера за дати проблем, док је $n_{\text{тачно}}$ број тачно класификованих примера коришћењем тренутног модела. Тачност се може тумачити и као вероватноћа да ће насумично одабран пример бити исправно класификован.

У пракси је готово немогуће израчунати реалну тачност класификације, зато што исправан начин класификовања није познат за сваки пример. Такође, n је често превелик број који тежи ка бесконачности. Зато се тачност процењује на основу независног тест скупа који се састоји од решених примера, односно обележених примера. Ако је n_t број тест примера, онда је класификациона тачност:

$$Acc_t = \frac{n_{t_{\text{тачно}}}}{n_t} \times 100\% \quad (2)$$

Тест скуп мора да буде независтан од тренинг скупа. Слично томе, тачност класификације тренинг скупа је:

$$Acc_{\text{Т}} = \frac{n_{\text{Т}_{\text{тачно}}}}{n_{\text{Т}}} \times 100\% \quad (3)$$

Постоје алгоритми који на основу ових образаца лако могу да развију модел који постиже тачност од 100%. Ипак, то су модели који памте све тренинг примере и њихова решења, односно ознаке. Такви модели имају добре перформансе само на тренинг подацима, што доводи до лошег процента реалне тачности класификације. Хипотезе за које важи $Acc_{\text{Т}} \rangle\rangle Acc_t$, каже се да „оверфитују“ (overfit) тренинг податке. Тиме се мисли на прекомерно

поклапање решења које нуди класификатор са решењима примера. Многи алгоритми машинског учења захтевају посебне технике прекомерног поклапања.
Доња граница тачности класификације се мери на основу већинске класе. То је класа која се најчешће појављује међу свим примерима датог проблема. Како расподела класа у примерима углавном није позната, процена поделе се врши на основу тренинг примера. Ако се са $n(C)$ означи број тренинг примера који припадају класи C, а n укупан број примера у тренинг скупу, онда се најнижа прихватљива тачност класификације примера већинске класе рачуна на следећи начин:

$$Acc_m = max_C \left\{ \frac{n(C)}{n} \right\} \times 100\% \quad (4)$$

Многи класификатори постижу знатно мању тачност класификације за већинску класу, испод 80%. Разлог је коришћење великог броја параметара (атрибута) при тренирању алгоритма који су небтини за предвиђање. Ако се такви параметри не открију, њиховим даљим коришћењем се утиче на понашање алгоритма. Класификација тада постаје слична насумичном генератору који је пондерисан вероватноћама претходних класа.

Класификациона тачност је просечна вредност свих класа, тако да вредност овог параметра не говори ништа о расподели исправно класификованих примера. Конфузиона матрица је боља за приказивање тачности класификације сваке класе посебно.

Цена погрешне класификације у неким проблемима се може значајно разликовати међу класама. Једна класа може да има много већу вредност овог параметра у односу на друге. Прави пример овакве ситуације су медицинске дијагнозе, где погрешна дијагноза може озбиљно да угрози пацијента чије је здравствено стање алармантно. Са друге стране, мања је грешка у постављању дијагнозе за пацијента који је здравствено стабилан. Код класификационих проблема чије цене погрешних класификација нису подједнаке, као главни критеријум користи се просечна цена погрешне класификације. Ако се са $n_t(i,j)$ означи број тест примера који припадају класи C_i, а класификовани су као C_j, онда се укупан број исправно класификованих тест примера рачуна на следећи начин:

$$n_{t,\text{тачно}} = \sum_{i=1}^{m_0} n_t(i,i) \quad (5)$$

Ознака m_0 представља број класа. Често се при коришћењу цена погрешних класификација користи и матрица цена у којој су распоређене све вредности. За матрицу цена $\{Cost_{i,j}\}$ важи: $Cost_{i,i} = 0$, $i = 1, \ldots, m_0$. Матрица цена не мора да буде симетрична. Просечна цена погрешне класификације је дата формулом:

$$Cost_t = \frac{\sum_{i,j}(Cost_{i,j}\, n_t(i,j))}{n_t} \quad (6)$$

Нема ограничења за вредности $Cost_{i,j}$ и $Cost_{j,i}$, што значи да се те вредности могу понављати.

Када се рачуна тачност класификатора, најчешће се користи класа која има највећу вероватноћу класификације. Заправо се мисли на класу чији се примери најчешће појављују. Ипак, такав приступ занемарује преостале информације које се добијају из вероватноће расподеле. У идеалном случају, за мере процене квалитета би требало да се користе „претходне“ (prior) и „наредне“ (posterior) вероватноће расподеле класа. *Браеров резултат* (Brier score) [1] и *маргина* користе само наредне, односно последње вероватноће расподеле, док *информативни резултат* користи обе расподеле.

Браеров резултат процењује разлику између реалног и идеалног циља расподеле. Ако је m_0 број класа, c_j одговарајућа класа за j-ти тест пример и $P'_j(C_i)$, $i = 1 \ldots m_0$ наредна вероватноћа класификатора, онда се идеална циљна дистрибуција представља:

$$P_j(C_i) = \begin{cases} 0, & C_i \neq c_j \\ 1, & C_i = C_j \end{cases} \quad (7)$$

Просек за све n_t тест примере је:

$$Brier = \frac{1}{n_t} \sum_{j=1}^{n_t} \sum_{i=1}^{m_0} (P_j(C_i) - P'_j(C_i))^2 \quad (8)$$

Формула представља просечну квадратну грешку предвиђања вероватноће расподеле класа. У идеалном случају, ако класификатор увек предвиђа тачну класу са вероватноћом $P'_j(c_j) = 1$, Браеров резултат је једнак нули. Са друге стране, ако предвиђа увек погрешну класу са вероватноћом 1, Браеров резултат је тада 2. Јединицу додељује за погрешну класификацију исправних класа и јединицу за погрешну класификацију нетачних класа, што укупно чини резултат 2. Процена класификатора се због тога креће у опсегу од 1 до *Brier/2*.

Код идеалне процене квалитета алгоритма у обзир се узимају обе врсте вероватноће, добијене од стране класификатора. Зато се често користи **информативни резултат**. На пример, у медицини класификатор достигне тачност од 80% за предвиђање карцинома дојке, а 45% за примарну локализацију тумора. На први поглед делује да је класификатор успешнији за први случај. Ипак, треба узети у обзир да у класификатор првог проблема постиже боље резултате зато што има само две или три класе, док се у другом проблему примери деле у знатно већи број класа (на пример, десет класа). Тада је довољно да класификатор има тачност од 80% за предвиђање карцинома већинске класе. Алгоритам који на овакав начин постиже високу тачност је уствари непредвидив на тест подацима, зато што своју тачност базира на класификатору већинске класе. Са друге стране, ако класификатор другог проблема постиже 45% на основу већинске класе, такав алгоритам се сматра исправним.

Комплексност проблема класификације се може проценити као број потребних информација за класификовање једног примера. Ова мера се зове *ентропија класе* (class entropy). Што је већа вредност ентропије, класификациони проблем је тежи.

Информативни резултат је мера која узима у обзир и претходне вероватноће. Уколико је наредна вероватноћа исправне класе већа од претходне вероватноће исте класе, онда је вредност информативног резултата позитивна. Позитивна вредност значи да је добијена информација исправна. У случају када је наредна вероватноћа за тачну класу мања од претходне вероватноће, прихваћена информација је нетачна, што се манифестује негативним информативним резултатом.

Разлика између тачности класификације и информативног резултата се може објаснити на следећем примеру. Постоји скуп који чине две класе $C = \{C_1, C_2\}$ са претходним вероватноћама $P(C_1) = 0.8$ и $P(C_2) = 0.2$. За одређени тест пример, класификатор враћа наредне вероватноће: $P(C_1) = 0.6$ и $P(C_2) = 0.4$. Ако је исправна класа C_1, онда је по тачности класификације то исправан одговор, док је по критеријуму информативног резултата то негативна вредност, што значи да је у питању погрешан одговор. Са друге стране, ако је C_2 исправна класа, одговор је погрешан по параметру тачности, а информативни резултат има позитивну вредност.

Код идеалног класификатора чија је вредност вероватноће увек једнака јединици за исправну класу, просечан информативни резултат је једнак ентропији класе. Ентропија је горња граница информативног резултата. За једноставан класификатор који враћа претходну вероватноћу за сваки пример, вредност информативног резултата је једнака нули. То је доња граница за перформансе осетљивог класификатора. Некада је нормализација информативног резултата са ентропијом боља, због поређења перформанси између различитих проблема.

Маргина је једноставна мера перформанси која само делимично узима у обзир вероватноћу класификације. Дефинише се као разлика између наредне вероватноће класе чија ће вероватноћа бити највећа и наредне вероватноће друге класе по вероватноћи. Маргина се углавном користи за мерење поузданости класификатора. Што је већа маргина, то је поузданија одлука коју доноси класификатор. На пример, класификације прве класе је поузданија ако су вероватноће класа 0.9 и 0.1, него ако су вероватноће 0.6 и 0.4. Маргина не узима у обзир претходне вероватноће класа.

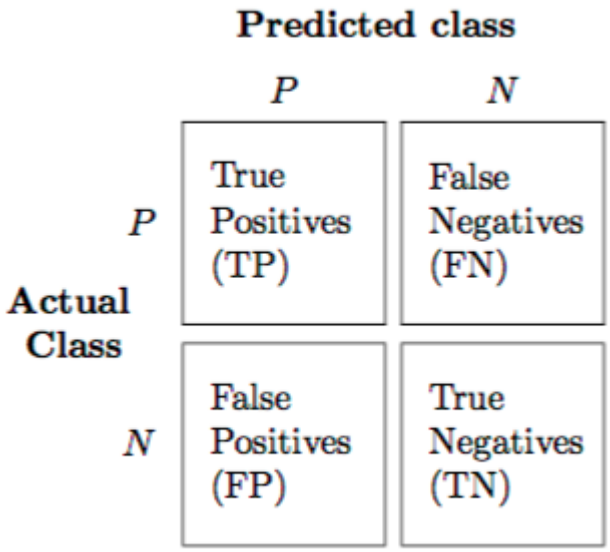
За дискриминаторне функције важи да је поузданост класификације дата као растојање између хиперравни. Што је већа удаљеност, већа је вероватноћа припадања примера класи на чијој се страни поделе налази. Такође, што је већа маргина поделе око хиперравни која не садржи примере учења, већа је поузданост дискриминаторне функције.

Како је проблем машинског учења у овом раду класификациони проблем, чији су примери подељени у две класе, потребно је анализирати начине мерења перформанси за ову врсту проблема.

3.2 Процена перформанси за проблеме са две класе

Под проблемима са две класе сматрају се ситуације у којима класификатор треба да се одлучи за или против одређене категорије. На пример, одлучивање да ли је пацијент болестан или не, да ли је играч довољно добар за прву поставу или не, да ли постоји постоји ризик за клијента при куповини или не итд... Примери који припадају „за“ класи се називају *позитивни примери*, док примери друге класе спадају у *негативне примере*. За анализу ових примера користи се матрица конфузије.

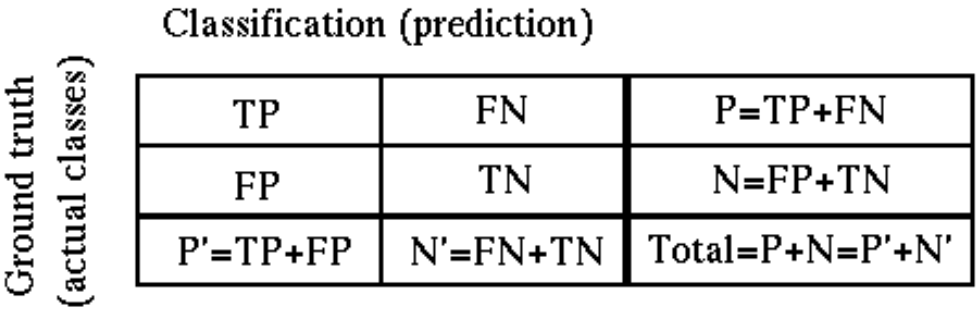
Сензитивност и специфичност су најчешће коришћене мере перформанси. Ове мере се рачунају помоћу четири основна елемента матрице конфузије (Слика 4).



Слика 4. Распоред елемената у матрици конфузије

Примери позитивне класе у матрици конфузије се деле на две групе: *True Positives* (TP) и *False Positives* (FP). Ознака TP представља број позитивних примера које је класификатор означио да припадају позитивној класи, односно исправно обележио. Са друге стране, FP је број примера за које се класификатор такође изјаснио да припадају позитивној класи, али се испоставило да су то негативни примери. Негативни примери се на сличан начин деле у матрици конфузије: *True Negatives* (TN) и *False Negatives* (FN). Аналогно подели позитивних примера, TN су примери који су исправно класификовани као негативни а FN су позитивни примери који су погрешно обележени као негативни.

Сабирањем означених примера по колонама и редовима у матрици конфузије, могу се добити статистички подаци на основу којих се рачунају сензитивност, специфичност и други параметри који ће бити наведени (Слика 5).



Слика 5. Статистички подаци у матрици конфузије

Сумирањем првог реда матрице, добија се вредност P која представља укупан број позитивних примера који су коришћени за тестирање. Вредност N је укупан број негативних примера, односно збир FP и TN. Суме по колонама представљају број класификованих позитивних, односно негативних примера. Коначно, збир све 4 ћелије матрице даје целокупан број тест примера.

Специфичност је релативна фреквенција исправно класификованих негативних примера:

$$Spec = \frac{TN}{TN + FP} = \frac{TN}{N} \quad (9)$$

Сензитивност је релативна фреквенција исправно класификованих позитивних примера:

$$Sens = \frac{TP}{TP + FN} = \frac{TP}{P} \quad (10)$$

Тачност класификације се може представити и преко елемената матрице конфузије:

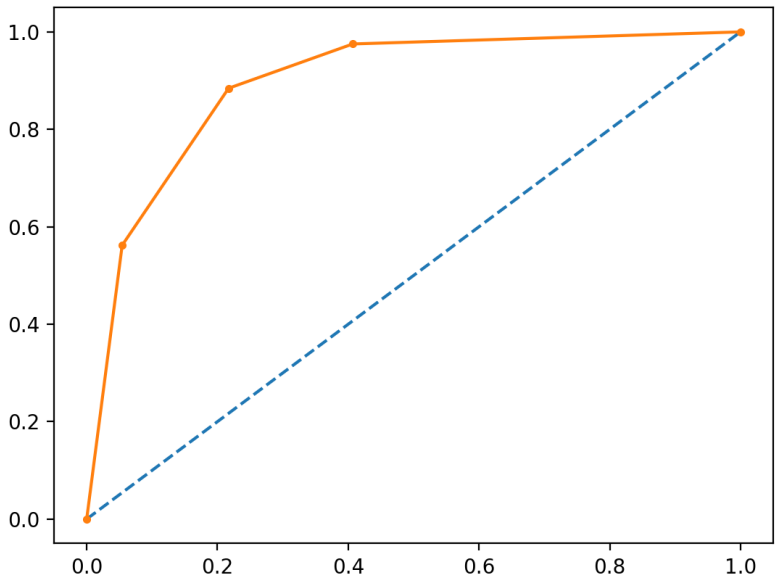
$$Acc = \frac{TP + TN}{TN + FP + FN + TN} = \frac{TP + TN}{Total} \quad (11)$$

Сензитивност и специфичност се често користе у медицинској терминологији за процену квалитета дијагностичких тестова. Ако се за неку дијагнозу у лабораторијским тестовима испостави да је сензитивност 0.85 и специфичност 0.95, онда постоје два могућа исхода. У случају да је пацијент болестан, тестом се потврђује болест са вероватноћом од 0.85. Друга варијанта важи ако пацијент није болестан. Тада тест погрешно класификује пацијента да је болестан са грешком од 0.05.

Када класификатор све примере означи позитивним, тада се постиже сензитивност 1.00, док је вредност специфичности једнака нули. Са друге стране, уколико се сви примери класификују као негативни, тада се добија максимална вредност специфичности (1.00) и сензитивност је 0.00.

Иако су ови параметри битни за оцењивање алгоритама класификације, могу се лако изманипулисати. Зато је потребна што боља балансираност односа између параметара.

За оптималан класификатор је веома битан однос између сензитивности и специфичности. Због тога се користи „ROC“ (Receiver Operating Characteristic) крива, за графичко приказивање ових параметара (Слика 6). Ова крива функционише по теорији детекције сигнала.



Слика 6. Пример „ROC“ криве

На хоризонталној оси је однос *false positives* примера, чија је вредност еквивалентна изразу „1 - специфичност“. На вертикалној оси се приказује однос *true positives* (сензитивност). Наранџастом бојом је на слици 6 представљена „ROC“ крива, док је граница „добрих“ класификатора приказана плавом испрекиданом линијом. Уколико класификатор увек предвиђа припадност за само један пример, онда се добија само једна тачка за „ROC“ криву. Ако класификатор за сваку класу враћа резултат, односно вероватноћу, онда је у питању класификатор оцењивања. За сваки пример *x*, класификатор оцењивања враћа резултат за позитивну и негативну класу у облику $f(x, +)$ и $f(x, -)$. Често се ова врста клафисикатора претвара у *класификаторе рангирања*, тако што се рачуна вероватноћа да је пример *x* позитиван:

$$f(x) = \frac{f(x,+)}{f(x,-)} \quad (12)$$

Вредност *f(x)* се користи за рангирање примера у опадајућем поретку по вероватноћи да је пример позитиван. „ROC“ крива за рангирајући класификатор се креира мењањем границе одлучивања. Стандардна граница је 0.5, што значи да ће припадност примера одређеној класи бити означена уколико је вероватноћа већа од 0.5. Уколико је граница 0.0, онда су сви примери класификовани као негативни. То значи да су сви негативни примери исправно класификовани, док су сви позитивни примери нетачно означени.

Када је граница 1.0, то значи да су сви примери класификовани као позитивни. Исправно класификовање свих позитивних примера и погрешно класификовање свих негативних примера се манифестује кроз вредности сензитивности 1.0 и специфичности 0.0. Тачка са овим координатама на графику „ROC“ криве налази у горњем десном углу (Слика 6). Друга екстремна тачка графика се налази у доњем левом углу, где су вредности сензитивности и специфичности, 0.0 и 1.0. Као што је већ наведено, ове вредности параметара се јављају у случају када класификатор све примере обележава као негативне.

Дијагонала која повезује две екстремне тачке је управо плава испрекидана линија, која карактерише насумичне класификаторе са различитим границама одлучивања. Класификатори се сматрају корисним ако се налазе у области изнад испрекидане линије. Идеалан случај класификатора чије су вредности 1.0 за оба параметра, требало би да се налази у левом горњем углу графика. Што је ближи класификатор идеалном случају на „ROC“ криви, сматра се квалитетнијим.

Када се пореде два класификатора, сматра се да је бољи онај чија је сензитивност и специфичност већа од параметара другог класификатора. Уколико је само један параметар класификатора бољи од другог, или нека друга варијанта, тада се класификатори не могу упоређивати. Ипак, ако се мењањем граница одлучивања у опсегу од 0 до 1, исцртају „ROC“ криве оба класификатора, квалитети класификатора се испољавају у области испод „ROC“ криве која се зове „AUC“ (Area Under the Curve). Бољи је онај класификатор који има већу област „AUC“.

Вредност „AUC“ има и статистичко значење, јер она заправо даје исту вредност као и „Wilcoxon-Mann-Whitney“[2] статистички тест. Тестом се рачуна вероватноћа да ће класификатор дати већу оцену насумично одабраном позитивном примеру него насумично одабраном негативном. Мењањем прага одлучивања у току цртања „ROC“ криве одговара промени цене погрешне класификације за позитивне и негативне примере. Зато се „ROC“ крива може користити и за анализу цене погрешне класификације.

За алгоритме проналажења информација и у бинарној класификацији користе се параметри **прецизност** (precision) и **одзив** (recall). Прецизност или „предвиђена позитивна вредност“ представља број релевантних примера у добијеном скупу примера. Одзив је број пронађених релевантних примера од укупног броја релевантних примера. Оба параметра се заснивају на мерењу релевантности система. На пример, класификатор добрих играча из понуђеног скупа од 12 играча (међу којима има и добрих и лоших), идентификује 8 играча као позитивне примере. Од класификованих 8 играча, 5 су заиста позитивни примери, док су преостали негативни. Тада је прецизност класификатора пет осмина (5/8), док је одзив једнак пет дванаестина (5/12). Из примера се може закључити да прецизност одговара на питање колико су користни резултати класификације, док одзив објашњава у којој мери су комплетни резултати. Једноставно речено, прецизност је мера квалитета, а одзив је мера комплетности алгоритма. Висока прецизност значи да је алгоритам класификовао више исправних примера, а висок одзив значи да је алгоритам вратио већину релевантних резултата.

Формула за израчунавање одзива:

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{P} \quad (13)$$

Образац за израчунавање прецизности:

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{P'} \quad (14)$$

У класификационим проблемима, прецизност класе је број примера који су исправно означени припадници позитивне класе подељен укупним бројем примера које је класификатор обележио као позитивне (међу којима се налазе примери позитивне и негативне класе). Одзив је дефинисан као број *true positives* подељен са целокупним бројем примера позитивне класе, у чијем скупу постоје позитивни и примери који нису обележени као позитивни иако би требало да буду.

При проналажењу докумената или информација, максимална прецизност 1.0 значи да је сваки резултат претраге релевантан (иако се не зна да ли су сви релевантни документи пронађени). За вредност одзива 1.0 сматра се да су све релевантне информације пронађене, али се не зна колики је број непотребних информација такође пронађен. За класификаторе оцењивања се може добити крива прецизност-одзив ако се мењају границе одлучивања. Добијена крива има слична својства као „ROC“ крива.

Још једна често коришћена мера перформанси код проблема проналажења информација је „F“ мера (F-measure). Ова мера је комбинација прецизности и одзива:

$$F = \frac{2 * Recall * Precision}{Recall + Precision} = \frac{2 * TP}{2 * TP + FP + FN} \quad (15)$$

Прецизност, одзив и „F“ мера се не фокусирају на потенцијално велики број примера који су исправно класификовани као негативни (true negatives), на шта се не обраћа пажња при претрази битних докумената.

4. Неуронска мрежа

Вештачка неуронска мрежа (Artificial Neural Networks - ANN) су рачунарски системи који су инспирисани биолошким неуронским мрежама, али нису идентични. Овакви системи „уче“ како да изврше задатак тако што разматрају примере, без икаквих иницијалних правила. Пошто не постоји универзално правило за класификовање успешних играча, неуронска мрежа је адекватан систем за проблем који је разматран у овом раду. Мрежа без претходних сазнања о играчима покушава да направи разлику тако што аутоматски обрађује добијене примере и идентификује карактеристике играча.

Основна предност неуронских мрежа је употреба дистрибуиране меморије при учењу нових података. Меморијски елементи су тежински фактори синапси. Сваки неурон приступа меморији тежинског фактора синапсе којом је повезан са другим неуронима. Углавном неурон није повезан са свим преосталим неуронима у систему, већ са мањом групом неурона. То може бити велико ограничење које утиче на понашање читаве мреже.

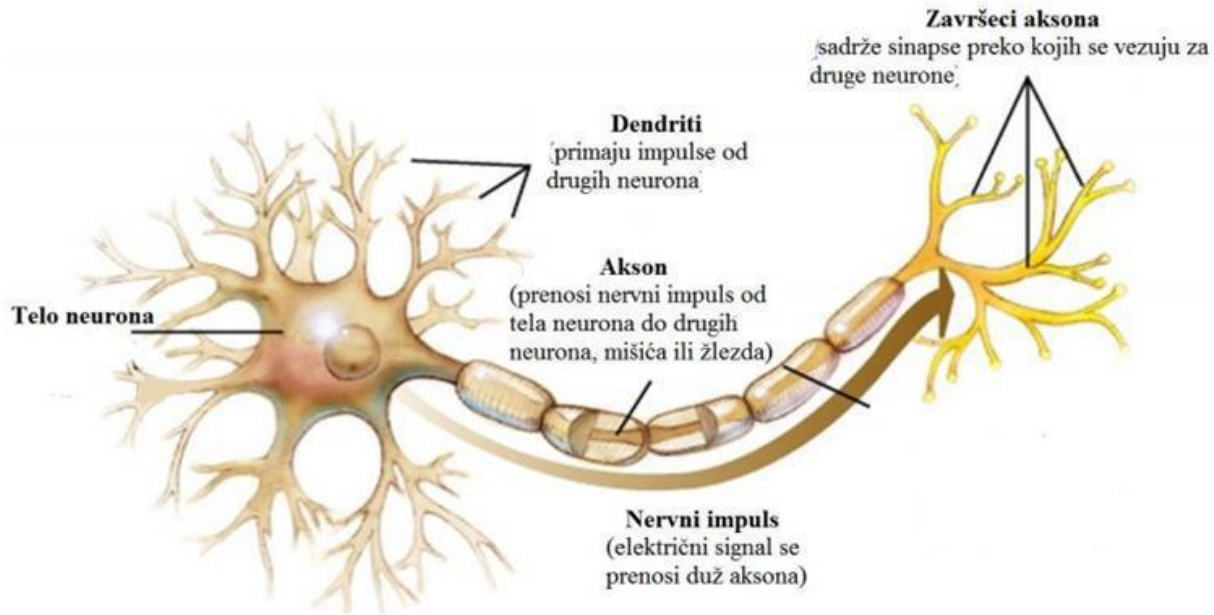
У дистрибуираној меморији сваки податак се парцијално чува у одређеном броју меморијских елемената. При захтеву за сачуваном информацијом, одзив меморијских елемената није увек исти. Зато је добијена информација приближна вредност оригиналне информације, јер сваки меморијски елемент чува истовремено више различитих информација. Сваки меморијски елемент представља неку врсту корелације између два повезана неурона. Зато неурони помажу једни другима у стварању приближне информације, када постоје неки недостатци или погрешне информације. Неуронска мрежа се може прилагодити променљивом окружењу, тако што постепено мења тежинске факторе својих неурона. При промени долази до заборављања старих информација, које мрежа контролише одређеним параметрима.

Повећањем нивоа дистрибутивности, може се повећати и прецизност при чувању информација. Свака компонента када се сачува, повезана је са још неколико других компоненти како би био прецизнији одзив при захтеву мреже. Уклањање једног неурона из мреже не води ка потпуном губитку информације коју је чувао, зато што је сваки неурон повезан са групом неурона и учествује у складиштењу информације. Али, зато утиче на прецизност поновног састављања информације када се то затражи. Ово својство је карактеристично и за биолошке неуронске мреже. Често се догађа да се мрежа присети заборављеног податка током учења нових.

Неуронске мреже се разликују од рачунарских система по чувању података. Сваки податак у рачунару има тачну адресу и чува се локално. Са друге стране, дистрибуирана меморија неуронске мреже адресира податке тако што их повезује са другим подацима. При захтеву мреже за неким податком, добија се оригиналан податак уколико се затражи тачна адреса. У супротном, добија се приближан одговор, који је сличан оригиналној информацији. Меморија мреже омогућава адресирање преко садржаја података. Зато нема разлике између сачуваног податка и његове адресе. Када је потребно скроз другачије приказивање података мреже, долази до промене модула у меморији.

Иако вештачка неуронска мрежа није идентична биолошкој, постоје својства која су слична биолошким системима: паралелизам, асихроно процесирање података, извршавање захтева у различитим смеровима, прилагодљивост у реалном времену, аутоматска генерализација и способност учења. Поред наведених својстава, вештачка неуронска мрежа има и добро развијену математичку основу. Највећа мана овог система јесте што мрежа нема способност објашњења својих одлука. Зато је људима понекад тотално нејасно одлучивање неуронске мреже. Ипак, ниједна мрежа не садржи сва наведена својства већ само одређена својства, у зависности од смера у коме се развија неуронска мрежа.

Циљ вештачке неуронске мреже јесте „имитирање“ функција које има биолошка неуронска мрежа и да побољша перформансе при решавању тешких проблема. Постоји више врста биолошких неурона у различитим деловима мозга, али су у основи сви исти (Слика 7).



Слика 7. Шема биолошког неурона

Дендрити неурона прихватају улазне сигнале, док аксони преносе излазне сигнале неруона преко синапси до дендрита других неурона. У вештачким неуронким мрежама користе се поједностављене верзије биолошких неурона, које без обзира на своју једноставност омогућавају квалитетно моделирање и анализе. Учење и функционисање вештачких неурона се базира на локалним информацијама. На пример, за промену тежинског фактора синапсе једног неурона у току учења, потребне су промене стања само неурона који су повезани том синапсом. Принцип локалности је сличан принципу биолошких неурона.

Висок ниво паралелизма у функционисању вештачке неуронске мреже се базира на независном раду сваког неурона. То омогућава веома брзе реакције целокупне мреже. Самим тим се и мрежа брже адаптира новим окружењима која су комплекснија од претходних.

Под извршавањем захтева у различитим смеровима мисли се на неке топологије мрежа које не праве разлику између улазних и излазних података. Сваки неурон се може користити за улаз, излаз или оба. При неком захтеву мреже, група неурона може да представља улаз система, док у другој ситуацији та иста група може да функционише као излаз. Та промена улога у систему не утиче на губитак или оштећење података. Као што је већ речено, уклањање неурона такође не води ка комплетном губитку информација, али опада тачност при „присећању“ одговарајућег податка. Као и код биолошких неурона, са постепеним уклањањем неурона, постепено опада и тачност. Губитак дела информација који је настао на овај налчин се компензује остатком те информације у меморији повезаних неурона. Од величине губитка зависи колико ће нова информација бити приближна оригиналној.

Учење неуронске мреже је процес „спонтане“ промене тежинских коефицијената у синапсама. Синапса је меморијски елемент који представља реакцију на активности које се дешавају између два повезана неурона. Гледано са софтверске стране, неуронске мреже не користе никакав класичан софтвер при учењу. Једино се неурони понашају по дефинисаном алгоритму, док се све остало сматра спонтаним реакцијама и променама. Неурони могу бити статички и динамички у погледу хардвера. Динамички неурони уклањају старе синапсе и стварају нове. Једини захтев при регулисању броја синапси јесте да тај број буде знатно већи од број неурона.

Када се помиње математичка основа неуронских мрежа, мисли се на теоријску основу линеарне алгебре. Примењују се појмови као што су: линеарност, линеарна и инверзна трансформација, ортогоналност, фиксна тачка и конвергенција. Свака одлука коју донесе неуронска мрежа је настала истовременом применом линеарне алгебре и комплексним процесима које извршавају сви неурони. Пошто је то тешко протумачити, неуронске мреже нису погодне за проблеме у којима се тражи транспарентност одлучивања.

Пошто ова област вештачке интелигенције напредује, долази до ширења области у којима се примењује. Постоји и више врста неуронских мрежа које су се развијањем и специјализацијом показале као добре у решавању разних проблема. Поред класификационих проблема, неуронске мреже се примењују у следећим проблемима: разне врсте детекција медицинских слика, анализа и предвиђање, препознавање и класификација слика, препознавање гласа и говора, контрола квалитета у процесима производње итд...

Код препознавања слика важно је добити обраду ротације и димензија, што постижу неке врсте неуронских мрежа. Основна идеја је испробавање неколико паралелних трансформација како би се упоредили резултати са сачуваним обрасцима. Итеративним понављањем овог процеса, мрежа покушава да потврди образац који даје најбоље резултате. За проблеме класификације великог броја класа постоји неуронска мрежа која сама организује своје неуроне, тако да је сваки неурон специјализован за једну класу слика. Још један пример примене је комбинаторна оптимизација, у коју спадају велики проблеми где треба да се обрати пажња истовремено на већи број ограничења. Због својства паралелизације проблема, неуронске мреже могу бити много ефикасније у односу на стандардне методе. Једини предуслов код примене мрежа на ову врсту проблема је да број улазних неурона треба да буде једнак броју параметара проблема.

4.1 Врсте неуронских мрежа

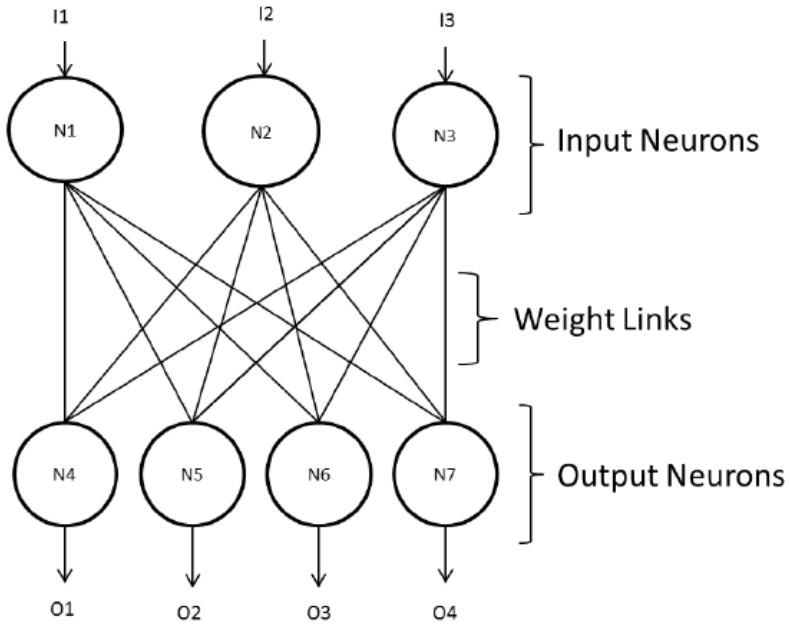
Вештачке неуронске мреже се у зависности од критеријума поделе деле по следећим параметрима:

- Топологија
- Захтеви апликације
- Правило учења
- Комбинациона функција

Постоје разне врсте топологија неуронских мрежа, међу којима су најчешће коришћене: топологија без слојева, топологија са два или више слоја неурона применом учења унапред (feedforward net) и бидирекциона двослојна неуронска мрежа.

Неуронска мрежа са топологијом која нема дефинисане слојеве подразумева се да је сваки неурон повезан са свим осталим неуронима и то у оба смера. Стање сваког неурона се иницијално подешава. Након стартних подешавања, почињу тренинг итерације. Сваки неурон сихроно или асихроно у односу на остале неуроне, истовремено рачуна и ажурира своје стање на основу сигнала који добије од других неурона. Процес итерирања се понавља све док се не испуне одређени услови, попут фиксне тачке. Како би се спречила циклична итерација и мрежа зауставила у жељеним условима, потребан је механизам који контролише преко повратних информација.

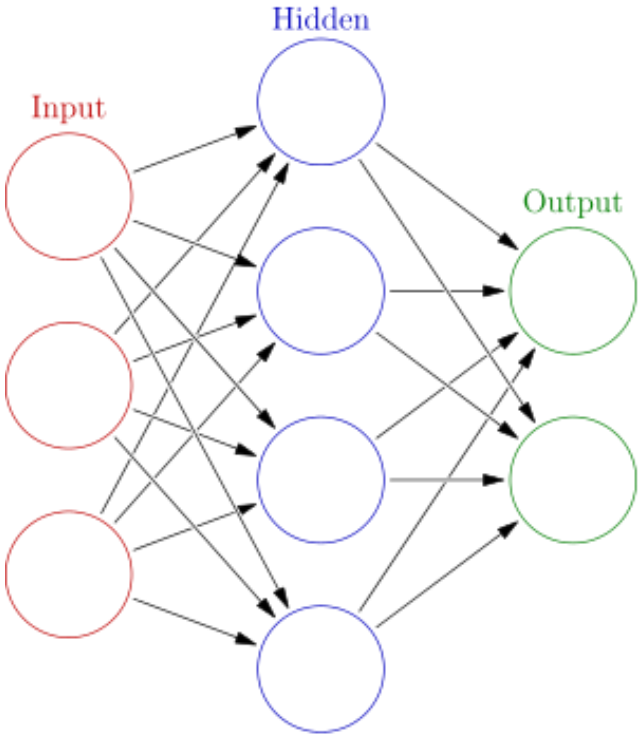
Топологија са два слоја неурона садржи улазни и излазни слој (Слика 8).



Слика 8. Пример топологије са два слоја неурона

Сваки улазни неурон је повезан са сваким излазним неуроном и то директном везом (једносмерном). Процесирање почиње као и код топологије без слојева, подешавањем стања сваког неурона, али само у улазном слоју. Неурони у излазном слоју само након прве итерације подешавају своје стање на основу добијених сигнала из улазног слоја. Често се ова врста топологије назива и неуронска мрежа са једним слојем, зато што улазни неурони не израчунавају своје стање, већ само преносе добијене вредности до излазног слоја. Ако сваки неурон користи линеарну комбинацију функција, тада мрежа са улазним и излазним слојем може да решава само линеарне проблеме.

Вишеслојна неуронска мрежа са учењем унапред је једноставно речено надоградња претходно наведене топологије. Између улазног и излазног слоја додаје се такозвани „скривени“ слој неурона (Слика 9). Функционишу слично мрежама са два слоја, само што је број корака у којима се израчунава стање свих неурона већи за број неурона у скривеном слоју плус један. Израчунавање почиње од скривеног слоја, а завршава се излазним слојем неурона. Иако користе функцију линеарне активације, ова врста топологије може да решава и нелинеарне проблеме.



Слика 9. Пример вишеслојне неуронске мреже са четири неурона у скривеном слоју

Бидирекциона (двосмерна) неуронска мрежа са два слоја садржи неуроне, где је сваки од њих повезан двосмерно са свим неуронима из другог слоја. Иницијално се подешавају неурони у једном слоју. У наредним итерацијама се врши израчунавање и ажурирање стања неурона само у супротном слоју. Итерирање престаје када се постигну жељени услови. Као и код неуронских мрежа без слојева, потребан је механизам за прекидање цикличног итерирања.

У заисности од захтева апликације постоје следеће неуронске мреже: ауто-асоцијативна меморија, хетеро-асоцијативна меморија, привремено асоцијативна меморија, класификатори и регресори и мреже за „кластеринг“ (clustering).

Ауто-асоцијативна меморија се користи код задатака у којима је потребно уредити непотпуне примере. Постоје и случајеви где се добије скуп „noisy“ примера, који може да садржи пуно грешака. Тада је задатак ауто-асоцијативне меморије да исправи примере, односно „филтрира“ сигнал. Углавном се ова врста меморије имплементира на топологији без слојева са повратним информацијама (feedback).

Хетеро-асоцијативна меморија је посебна варијанта ауто-асоцијативне меморије код које се пример дели на више мањих примера. Улазни сигнал неуронске мреже је један мањи пример или скуп сачињен од више мањих примера. Циљ мреже је реконструкција подпримера (мањег примера) који недостаје. Када се користе два подпримера као улазни сигнал, онда се хетеро-асоцијативна меморија имплементира на двосмерној топологији.

Привремено асоцијативна меморија се сматра специјалним случајем хетеро-асоцијативне меморије. Сваки подпример представља пример за посебан временски интервал. Углавном је циљ ове неуронске мреже да предвиди пример за наредни временски интервал, на основу примера из тренутног и неколико претходних временских интервала.

Код класификације постоје улазни неурони који прихватају карактеристике примера и излазни неурони који на основу добијених информација одређују класну припадност. Најчешће се за ову врсту неуронских мрежа користи вишеслојна топологија са учењем унапред. За регресију је довољан један излазни неурон са континуалним излазним вредностима.

Груписање примера (clustering) је проблем у коме неуронска мрежа треба да подели скуп примера у више подскупова који се називају „кластери“ (clusters). Примери су подељени у групе тако да свака група садржи примере који су међусобно слични. Неуронска мрежа се специјализује за ову врсту проблема тако што је сваки излазни неурон задужен за један кластер. Излазни неурон постане активан уколико улазни неурони добију карактеристике примера који би требао да припада одговарајућој групи. Структура неуронске мреже треба да буде организована тако да је број њених улазних неурона једнак броју карактеристика/атрибута, док је број излазних неурона једнак броју кластера. Сваки излазни неурон је повезан инхибиторном синапсом (синапса која спречава активацију) са свим преосталим излазним неуронима, да би истовремено могао само један излазни неурон да буде активан.

Неуронска мрежа може и самостално да организује неуроне уколико то захтева апликација. Због ефикасног представљања апликације некада је потребно на одговарајући начин организовати неуроне, тако да сваки одговара одређеном интервалу вредности улазних параметара. Уз одговарајућу топологију, неурони могу сами да се сортирају у таквом редоследу да неурони који су близу једни другима, реагују на сличне вредности улазних сигнала.

То се ради у циљу организације излазног слоја. Излазни неурони су повезани посебним синапсама са „комшијама“ у излазном слоју, док су са удаљенијим неуронима повезани инхибиторном синапсом. Локаклне интеракције између неурона чувају континуитет и граде основу неуронске самоорганизације.

Када су у питању правила учења, постоје: хебијско правило (Hebbian learning rule), делта правило и конкурентно правило учења. Хебијско и делта имају додатну поделу на основна и генерализована правила.

Хебијско основно правило учења се користи и у неуропсихологији. Сматра се да је вредност тежинског коефицијента везе два неурона пропорционална фреквенцији истовремене активности тих неурона. Правило дефинише постепено повећање (инкрементирање) тежинског коефицијента сваки пут када су оба неурона активна у току фазе учења. Са друге стране, генерализовано хебијско правило сматра да је вредност тежине везе два неурона пропорционално узајамним реакцијама повезаних неурона. Тежински коефицијент се такође инкрементује када су оба неурона активна, али и када ниједан од повезаних неурона није активан у фази учења. У случају када је активан само један неурон, тада генерализовано хебијско правило дефинише постепено смањивање тежинског коефицијента. Потребне информације се чувају локално, за обе врсте хебијског правила.

Делта правило учења се за основну врсту имплементира на двослојној неуронској мрежи са учењем унапред. Сви тежински коефицијенти се иницијално постављају на насумично одабране вредности. За сваки пример у току учења рачунају се излазне вредности, које се потом пореде са очекиваним излазним вредностима. Разлика између жељених и реалних излазних вредности представља грешку неуронске мреже. Грешка се користи за корекцију тежинских коефицијената, како би се грешка смањила. Правац који одређује корекцију је дефинисан изводом функције грешке. Амплитуда исправке се контролише посебним параметром који се назива коефицијент учења. Процес се итеративно понавља за сваки пример све док се не достигну жељени услови. Један од критеријума жељених услова је праг грешке који неуронска мрежа мора да задовољи да би се алгоритам учења зауставио.

Генерализовано делта правило се разликује од основног по томе што примењује учење са повратним информацијама грешака на двослојној „feedforward“ неуронској мрежи. Правило користи исте принципе као и основно делта правило. На почетку су сви тежински коефицијенти насумично одабрани и за сваки пример се рачуна грешка предвиђања. Грешка се користи за кориговање тежина у циљу побољшања. За разлику од основног делта правила, ажурирање тежинских фактора креће уназад, од излазног ка улазном слоју. Прво се ажурирају неурони излазног слоја. Процес аужирања се наставља све док се не стигне до улазног слоја. Тако се примењује учење са пропагацијом грешке уназад (backpropagation of errors). У односу на хебијско учење, учење са повратниим информацијама је мање веродостојно учењу биолошких неуронских мрежа. Још увек не постоје докази у неуропсихологији који би потврдили хипотезу „backpropagation“ учења у деловима мозга.

Конкуретно правило учења се од осталих правила разликује по активацији неурона. Само један неурон у слоју може у тренутку да буде активан. Преостали неурони тог слоја су неактивни услед повезаности инхибиторном синапсом. Како само један неурон може да буде активан, једино он има право да промени свој тежински коефицијент. Промена се врши тако што активни неурон постепено повећава своју тежину, док преостали неурони у слоју постепено смањују своје тежине. Неуропсихологија потрвђује доказима да мозак користи више комбинација конкурентног учења.

Још један веома битан фактор учења је брисање информација, односно заборављање застарелих података како би се неуронска мрежа фокусирала на нове податке. То је веома корисна функционалност код проблема у којима су новији подаци битнији од претходних. Када се врши ажурирање нових података, њима се даје већи фактор значаја, док се претходним подацима смањује фактор. Заборављање података се понекад спроводи постепеним смањивањем свих коефицијента тежине, где је смањивање пропорционално вредности тежине. Овакав процес уједно спречава превелике вредности тежинских коефицијената.

Комбинациона функција се састоји из два дела: активациона (*A*) и излазна функција (*f*). Активациона функција комбинује вредности улазних сигнала како би се добила нова вредност. Излазна вредност пресликава активацију која прелази одређени праг у стање неурона. Та стања даље утичу на излазну вредност неурона.

Најчешће коришћена активациона функција је линеарна функција, имплементирана као збир тежинских коефицијената:

$$A(X_j) = \sum_i W_{ij}X_i + C_j \quad (16)$$

Тежински коефицијент између *i*-тог и *j*-тог неурона је означен са W_{ij}. Стање *i*-тог неурона је X_i, док је са C_j означена активациона константа *j*-тог неурона која се назива *праг* (bias). Стања свих неурона у мрежи се представља вектором величине *N* (број неурона):

$$X = (X_1, X_2, \dots, X_N)^T \quad (17)$$

Тежине веза у неуронској мрежи се представљају матрицом:

$$M = \begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1N} \\ W_{21} & W_{22} & \cdots & W_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ W_{N1} & W_{N2} & \cdots & W_{NN} \end{bmatrix} \quad (18)$$

Излазне вредности *Y* неуронског слоја *X* након једне итерације се добијају формулом:

$$Y = MX \quad (19)$$

Матрица M представља линеарну трансформацију, па је зато активациона функција $Y = A(X)$ линеарна. Повремено се користи и општа фунцкија сигма-пи дата у форми:

$$sp(X_1, \dots, X_N) = \sum_{S_i \epsilon P} W_i \prod_{k \epsilon S_i} X_k \quad (20)$$

За скуп ознака свих неурона користи се P, док је S_i је подскуп неурона чије су излазне вредности помножене. Резултат множења је пондерисан тежином W_i. Коришћењем сигма-пи елемената може се имплементирати било која монотона Булова функција. Један неурон није довољан, јер се број подскупова S_i повећава при додавању улазних неурона. Сваки неурон користи онолико тежинских коефицијената колико има улазних неурона. Комбиновањем мање групе неурона, може се имплементирати било која монотона функција.

Излазна функција је увек детерминистичка, без обзира да ли је у питању бинарна, континуална или стохастичка функција. Под детерминистичком функцијом се подразумева да је излазна вредност неурона директно зависна од улазне вредности. Код детерминистичке стохастичке функције излазна вредност зависи искључиво од вероватноће појављивања примера одређене класе на улазу система. Са друге стране, бинарна детерминистичка функција је углавном заснована на некој граници (прагу вредности). Излазне вредности могу да буду 1 и 0, или 1 и -1. Праг функције може бити фиксиран или се може мењати у току учења.

Континуална детерминистичка функција је углавном сигмоидална функција, која произвољне реалне вредности пресликава у реалне вредности из интервала (0,1). За континуалну функцију излаза најчешће се користи:

$$f(X) = \frac{1}{1 + e^{-X}} \quad (21)$$

Највећа предност такве континуалне функције јесте могућност извода функције. То омогућава имплементацију градијентне претраге. Континуална функција даје предност и меморији, односно у повећању капацитета меморије неуронске мреже. На пример, ауто-асоцијативна неуронска мрежа са N неурона у комбинацији са континуалном функцијом може да има до 3^N фиксних тачака.

Неки модели неуронских мрежа користе стохастичку излазну функцију која омогућава претрагу вероватноћа, како би избегли локално оптимално решење. Наведена претрага даје вероватноће за потенцијалне излазне вредности, а потом бира одређену вредност на основу расподеле вероватноће. Овакав процес учења је знатно спорији, али омогућава мрежи да избегне велики број локалних оптималних решења. Такође, повећава вероватноћу проналажења глобалног оптимума.

Применом свих наведених карактеристика добијају се различите врсте неуронских мрежа. У дипломског раду је коришћена једна од најпознатијих врста неуронских мрежа, „вишеслојни перцептрон“.

4.2 Перцептрон

Перцептрон је неуронска мрежа са учењем унапред која користи улазни слој неурона само за преношење улазних сигнала до следећег слоја. Перцептрон који има само улазни и излазни слој неурона назива се „двослојни перцептрон“. Уколико има један или више додатних слојева (скривени слојеви), онда се назива „вишеслојни перцептрон“.

Вишеслојни перцепртон може да имплементира било коју нелинеарну функцију. Број неурона у улазном слоју зависи од захтева класификационог или регресионог проблема као што је већ наведено у претходном поглављу. Ипак, адекватан број скривених слојева и број неурона у тим слојевима се поставља емпиријски. Процес учења је сличан као код двослојне неуронске мреже, само се број корака израчунавања повећава за број скривених слојева. Израчунавање почиње од првог скривеног слоја и секвенцијално се наставља кроз скривене слојеве, закључно са излазним слојем. У сваком слоју неурони истовремено и независно рачунају своје излазне вредности и шаљу их свим неуронима у наредном слоју.

Ако је $m > 0$ број скривених слојева, где је број неурона по скривеном слоју N_k у границама $1 \leq k \leq m$. Са N_X и N_Y су означени улазни и излазни сигнали неурона. Тада се излазна вредност неурона у првом скривеном слоју за учење примера $t_l = X(l)$ рачуна на следећи начин:

$$H_{1i}(l) = f\left(\sum_{j=1}^{N_X} W_{ji}^{(1)} X_j(l)\right) \quad (22)$$

Ознака $W^{(k)}$ представља тежинску матрицу за синапсе које воде до k-тог скривеног слоја. Активациони ниво првог скривеног слоја је дефинисан образцем 23.

$$A_{1i}(l) = \sum_{j=1}^{N_X} W_{ji}^{(1)} X_j(l) \quad (23)$$

Неурони k-тог скривеног слоја рачунају своје излазне вредности на основу формуле 24, док се њихови активациони нивои рачунају на основу излаза $H_{k-1,j}$ из претходних слојева по образцу 25. Неурони у излазном слоју своје излазне вредности рачунају коришћењем образца 26.

$$H_{ki}(l) = f\big(A_{ki}(l)\big) \quad (24)$$

$$A_{ki}(l) = \sum_{j=1}^{N_k} W_{ji}^{(k)} H_{k-1,j}(l) \quad (25)$$

$$Y_i(l) = f\big(A_i(l)\big) \quad (26)$$

Како би се применило генерализовано делта правило учења, потребно је рачунање грешке излазних неурона. Грешка *i*-тог неурона за пример учења *l* је:

$$e_i(l) = d(l) - Y_i(l) \quad (27)$$

Целокупна грешка вишеслојног перцептрона се рачуна формулом 28.

$$E(l) = \frac{1}{2}\sum_{i=1}^{N_Y} e_i^2(l) \quad (28)$$

Вишеслојни перцептрон се може тренирати за решавање класификационих и регресионих проблема, применом генерализованог делта правила (познатије као „backpropagation of errors“). Идеја учења са повратним информацијама, односно „учења уназад“ је иста као код основног делта правила. Мрежа се иницијализује са насумично одабраним тежинским коефицијентима. Кроз учење примера адекватно се подешавају вредности улазних неурона, док мрежа рачуна излазну вредност. Разлика између жељених и добијених излазних вредности утиче на промену тежинских коефицијената, како би се смањила грешка. Процес модификације тежинских вредности се извршава уназад, од излазног слоја до првог скривеног слоја, без улазног слоја. Неурони улазног слоја немају тежинске коефицијенте, због чега се изостављају из процеса модификације.

Излазна функција мора бити континуална и са континуалним изводима, јер је потребно израчунати извод грешака за неуроне скривених слојева. Зато се као излазна функција користи сигмоидална функција:

$$f(X) = \frac{1}{1 + e^{-X}} \quad (29)$$

Проблем примене генерализованог правила учења код вишеслојног перцептрона лежи у чињеници да процес учења не води увек ка оптималној мрежи. Често се дешава да се неуронска мрежа заглави у локално оптималном решењу. Које ће решење бити одабрано у процесу учења највише зависи од иницијалног подешавања тежинских коефицијената. Ако одабрано решење не задовољава задате услове, читав процес учења се понавља. Такође, постоји проблем одабира броја скривених слојева и неурона у тим слојевима. За улазни и излазни слој се зна тачан број неурона, јер су они дефинисани самим проблемом. Ипак, за скривене слојеве не постоји никакво правило. Потребно је испробавање различитих комбинација скривених слојева, како би се утврдила адекватна структура скривених слојева која води ка оптималном решењу. Процес учења захтева велики број пролажења кроз исте примере како би се пронашло оптимално решење. Обично се број понављања мери стотинама или чак хиљадама итерација.

5. Имплементација система

За реализацију система за подршку одлучивању приликом избора успешних играча коришћен је програмски језик *Python*, у развојном окружењу *Anaconda3*. „Anaconda“ кроз своје виртуелно окружење олакшава употребу свих *Python* библиотека и пакета (којих има преко 1.500). Графичко окружење овог развојног алата се назива „Anaconda Navigator“ и омогућава покретање апликација, командне линије, стварање нових окружења итд... За развој система у овом раду је коришћена апликација *Spyder*, 3.3.3 верзија. Библиотеке и почетне промењљиве које су коришћене приказане су на слици 10.

```
import numpy as np
import glob
from imblearn.over_sampling import SMOTE
import os
path = 'Statistika 2018-2019'
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
import json
import requests
from PyQt5 import QtWidgets, uic, QtGui
from PyQt5.QtWidgets import QDialog,QVBoxLayout, QLabel, QLineEdit
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
import matplotlib.pyplot as plt


listaNepotrebnih = []
X = []
playerName = []
target = []
script_dir = os.path.dirname(os.path.abspath(__file__))+"/" + path
```

Слика 10. Библиотеке и промењљиве коришћене у скрипти

Свака од наведених библиотека биће објашњена на конкретном примеру употребе у коду. На почетку су инстанциране четири празне листе: *listaNepotrebnih, X, playerName* и *target*. Прва листа служи као евиденција играча чија статистика није коришћена у обучавању неуронске мреже. Како се користи статистика читаве Премијер лиге, на списку играча постоје играчи који су одиграли веома мали број утакмица. На основу статистике таквих играча не може се јасно проценити која је класа играча у питању, зато је потребно издвојити те примере. Листа *X* представља скуп примера са атрибутима играча на основу којих ће неуронска мрежа доносити одлуку, док листа *playerName* садржи имена играча чија се статистика користи. Сваки пример поред атрибута играча треба да садржи и ознаку 1 или 0, односно да ли тај играч припада класи успешних играча или не. Зато се користи листа *target*, која садржи само ознаке.

Промењљиву *script_dir* скрипта користи у проналажењу директоријума *Statistika 2018-2019*. У овом директоријуму се налазе текстуални фајлови који садрже целокупну статистику свих играча Премијер лиге у сезони 2018/2019, на којој се обучава неуронска мрежа. За проналажење путање до наведеног директоријума користи се библиотека „*os*“. Библиотека омогућава функционалности везане за оперативни систем. Углавном се користи за учитавање, писање датотека или манипулацију путања фајлова.

У овом случају комбиноване су методе *dirname* и *abspath* модула *os.path*, како би се пронашла апсолутна путања до директоријума у коме се налази покренута скрипта. На добијену путању се додаје назив жељеног директоријума како би путања била комплетна. Када се пронађе путања до директоријума, следи учитавање свих текстуалних фајлова и расподела података по листама (Слика 11).

```
for filename in glob.glob(os.path.join(script_dir, '*.txt')):
    fajl = np.genfromtxt(filename, delimiter='\t', skip_header=1, dtype=int)
    if (len(fajl) <= 11 or np.sum(fajl[:,5]) <= 360):
        listaNepotrebnih.append(os.path.splitext(os.path.basename(filename))[0])
    elif(np.mean(fajl[:,2]) >= 2):
        playerStats = np.array([np.mean(fajl[:,1]), np.mean(fajl[:,3]), np.mean(fajl[:,4]), np.mean(fajl[:,6]), np.mean(fajl[:,8])])
        X.append(playerStats)
        playerName.append(os.path.splitext(os.path.basename(filename))[0])
        target.append(1)

    else:
        playerStats = np.array([np.mean(fajl[:,1]), np.mean(fajl[:,3]), np.mean(fajl[:,4]), np.mean(fajl[:,6]), np.mean(fajl[:,8])])
        X.append(playerStats)
        playerName.append(os.path.splitext(os.path.basename(filename))[0])
        target.append(0)

targetArray = np.asarray(target, dtype=np.float64)
XArray = np.asarray(X, dtype=np.float64)
```

Слика 11. Учитавање статистике

Библиотека и њен модул *glob.glob* креирају списак свих фајлова који задовољавају наведени услов (екстензија „.txt“) и притом се налазе у наведеној путањи, односно у директоријуму *Statistika 2018-2019*. Пролажењем кроз *for* петљу врши се подела играча и креирање података који ће се користити за обуку неуронске мреже. За учитавање фајлова користи се метода *genfromtxt()* библиотеке *numpy*. Формат сачуване статистике играча у текстуалном фајлу је приказан на слици 12. Неуронска мрежа врши класификацију на основу просека следећих атрибута: број датих голова, број одбрана, број асистенција, бонус поена и броја жутих картона.

Aaron Wan-Bissaka.txt - Notepad

File Edit Format View Help

round	goals	points	cleanS	assists	min	bonus	home	yellowC	teamID	teamN
1	0	12	1	1	90	3	0	0	9	Fulham
2	0	-2	0	0	74	0	1	0	12	Liverpool
3	0	0	0	0	0	0	0	0	18	Watford
4	0	0	0	0	90	0	1	1	16	Southampton
5	0	9	1	0	90	3	0	0	10	Huddersfield
6	0	9	1	0	90	3	1	0	15	Newcastle
7	0	1	0	0	90	0	0	0	2	Bournemouth
8	0	2	0	0	90	0	1	0	20	Wolverhampton
9	0	1	0	0	90	0	0	0	8	Everton
10	0	1	0	0	90	0	1	0	1	Arsenal
11	0	1	0	0	90	0	0	0	6	Chelsea
12	0	2	0	0	90	0	1	0	17	Tottenham
13	0	6	1	0	90	0	0	0	14	Manchester United
14	0	6	1	0	90	0	1	0	4	Burnley
15	0	1	0	0	90	0	0	0	3	Brighton
16	0	1	0	0	90	0	0	0	19	West Ham
17	0	6	1	0	90	0	1	0	11	Leicester
18	0	1	0	0	90	0	0	0	13	Manchester City
19	0	8	1	0	90	2	1	0	5	Cardiff
20	0	2	0	0	90	0	1	0	6	Chelsea
21	0	6	1	0	90	0	0	0	20	Wolverhampton
22	0	4	0	0	90	3	1	0	18	Watford
23	0	0	0	0	90	0	0	0	12	Liverpool
24	0	1	0	0	90	0	0	1	16	Southampton
25	0	5	1	0	90	0	1	1	9	Fulham
26	0	1	0	0	90	0	1	1	19	West Ham
27	0	0	0	0	0	0	0	0	11	Leicester
28	0	0	0	0	0	0	1	0	14	Manchester United
29	0	5	0	1	90	0	0	0	4	Burnley
30	0	1	0	0	90	0	1	0	3	Brighton
32	0	6	1	0	90	0	1	0	10	Huddersfield
32	0	1	0	0	90	0	0	0	17	Tottenham
33	0	7	1	0	90	1	0	0	15	Newcastle
34	0	1	0	0	90	0	1	0	13	Manchester City
35	0	1	0	0	90	0	0	0	1	Arsenal
36	0	9	1	0	90	3	1	0	8	Everton
37	0	1	0	0	90	0	0	0	5	Cardiff
38	0	4	0	1	90	0	1	0	2	Bournemouth

Слика 12. Статистика играча

Да би фудбалер у игрици добио бар један поен, потребно је да на утакмици буде у првој постави или да одигра више од 10 минута. Сваки играч који је у току целе сезоне одиграо само 4 утакмице (360 минута) или је просечно проводио мање од 10 минута (по утакмици) на терену, додаје се у листу *listaNepotrebnih*. Критеријум по коме су играчи класификовани је број освојених поена у сваком колу. За целу одиграну утакмицу играч добија 2 поена, без обзира на његов учинак на терену. Дакле, ако играч на крају сезоне има просечан број поена преко 2, то значи да је највероватније у питању „првотимац“ (играч који редовно игра у првој постави). Како тренери за прву поставу бирају најбоље играче, може се закључити да је то успешан играч. Зато се за те играче у листи *target* уписује вредност 1. У супротном, ако играч има просечну вредност остварених поена испод 2, уписује се вредност 0. Када се заврши учитавање свих играча листе *target* и *X* се конвертују у низове помоћу методе *asarray()*, из математичке библиотеке *numpy*. Библиотека за креирање и обучавање вишеслојног перцептрона *sklearn* захтева податке у виду дводимензионих и једнодимензионих низова, због чега се и врши наведено претварање листа.

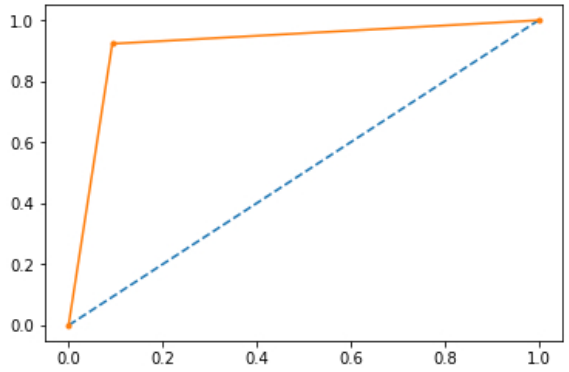
Као што је наведено у претходним поглављима, да би се развила прецизна неуронска мрежа потребно је испробавање различитих структура мрежа. Слика 13 приказује део кода који је коришћен за тренирање мреже и приказивање резултата.

```
X_train, X_test, y_train, y_test = train_test_split(XArray, targetArray, test_size = 0.2, random_state = 0)
# ================================================================================
# SMOTE balansiranje trening podataka
# ================================================================================
print("Trening podaci")
print(len(X_train))
print(np.count_nonzero(y_train == 1))
print(np.count_nonzero(y_train == 0))
smt = SMOTE()
X_train, y_train = smt.fit_sample(X_train, y_train)
print("Nakon SMOTE")
print(len(X_train))
print(np.count_nonzero(y_train == 1))
print(np.count_nonzero(y_train == 0))

#Neuronska mreza
mlp = MLPClassifier(hidden_layer_sizes=(15,10),max_iter=1000)
mlp.fit(X_train,y_train)

predictions = mlp.predict(X_test)
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
# ================================================================================
# ROC kriva
# ================================================================================
auc = roc_auc_score(y_test, predictions)
print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, predictions)
pyplot.plot([0, 1], [0, 1], linestyle='--')
pyplot.plot(fpr, tpr, marker='.')
pyplot.show()
```

Слика 13. Обучавање и приказивање резултата мреже

Методом *train_test_split()* врши се насумична подела података на тренинг и тест скупове. Најбоље се показала подела 80% према 20%. Метода прима као аргументе улазне податке (*XArray*), ознаке примера (*targetArray*), проценат података који ће припадати тренинг скупу у опсегу од 0 до 1 (*test_size*) и „seed“ за генератор насумичног одабира.

```
Trening podaci
327
144
183
Nakon SMOTE
366
183
183
[[39  4]
 [ 3 36]]
              precision    recall  f1-score   support

         0.0       0.93      0.91      0.92        43
         1.0       0.90      0.92      0.91        39

   micro avg       0.91      0.91      0.91        82
   macro avg       0.91      0.92      0.91        82
weighted avg       0.91      0.91      0.91        82

AUC: 0.915
```



Слика 14. Резултати предвиђања мреже

Након поделе, врши се пребројавање узорака обе класе у тренинг скупу. Да би се унапредиле перформансе, неуронска мрежа би требало да користи балансиране податке за обучавање. Под балансираним подацима се сматра скуп у коме је однос између примера обе класе 40% према 60%, најмање. За додатно балансирање користи се модул *SMOTE* из библиотеке *imblearn.over_sampling*. Овај модул врши балансирање података тако што креира синтетичке (вештачке) примере за класу која има мање примера. *SMOTE* бира међусобно сличне и суседне примере из класе и на основу њихових атрибута синтетише нове. Новогенерисани примери нису дупликати постојећих примера, али имају сличне вредности атрибута као суседни примери. На крају овог алгоритма, обе класе имају подједнак број примера (Слика 14).

Модул *MLPClassifier* из библиотеке *sklearn* се користи за креирање вишеслојног перцептрона и користи се у класификационим проблемима. Најбоље резултате је показала мрежа са два скривена слоја, где је у првом слоју 15, а у другом 10 неурона. Аргумент *max_iter* служи као ограничење броја итерација. Уколико мрежа не достигне оптималну вредност након 1000 итерација, обука се прекида. Након обуке мрежа врши класификацију тренинг података методом *predict()*. Разлика између предвиђених и реалних излазних вредности се приказује кроз матрицу конфузије и кроз параметре перформанси (Слика 14). Као једна од мера перформанси, у обучавању мреже је коришћен и график „ROC“ криве. Када се пронађе оптимално решење више нема потребе за коришћењем овог дела кода. Зато се тај део може „закоментарисати“ или уклонити.

За креирање и дизајнирање графичког интерфејса система коришћена је библиотека *PyQt5*. Почетни прозор је дизајниран помоћу апликације *QtDesigner*, која је саставни део *Anaconda3* окружења. На слици 15 приказан је део кода који служи за покретање и функционисање почетног прозора. Прозор садржи објекат *QComboBox* који представља падајући мени. Списак свих играча који тренутно играју у Премијер лиги се преузима са званичног сајта игрице „Fantasy Premier League“. Када се преузме списак, помоћу друге *for* петље се учитавају имена играча у *QComboBox*. Поред падајућег менија постоји и тастер *QPushButton* у почетном прозору. Кликом на тастер активира се метода *Promeni()*, која има задатак да отвори додатан прозор. Нови прозор је дефинисан класом *PlayerWindow()*.

```
dialogs = list()

def Promeni():
    dialog = PlayerWindow()
    dialogs.append(dialog)
    dialog.show()

app = QtWidgets.QApplication([])
dlg = uic.loadUi("MainGUI.ui")

url = "https://fantasy.premierleague.com/api/bootstrap-static/"
zahtev = requests.get(url)
if zahtev.status_code == requests.codes.ok:
    podaci = json.loads(zahtev.text)
    podaciOIgracima = podaci['elements']
    igraci = {}

    for i in podaciOIgracima:
        igraci[i['id']] = i['first_name']+' '+i['second_name']

for key, value in sorted(igraci.items(), key=lambda item: item[1]):
    dlg.comboBox.addItem(value)

dlg.pushButton.clicked.connect(Promeni)


dlg.show()
app.exec()
```

Слика 15. Креирање почетног прозора

Нови прозор садржи график остварених поена за одабраног играча и одлуку алгоритма. Подаци о поенима играча се преузимају са истог сајта и приказују се графиком помоћу библиотеке *matplotlib* (Слика 16). За преузимање података коришћена је библиотека *json* и библиотека за „URL“ адресе *requests*. На основу тренутне статистике одабраног играча неуронска мрежа врши предвиђање коришћењем истих атрибута који су употребљени за тренинг. У зависности од одлуке неуронске мреже корисник добија адекватан одговор.

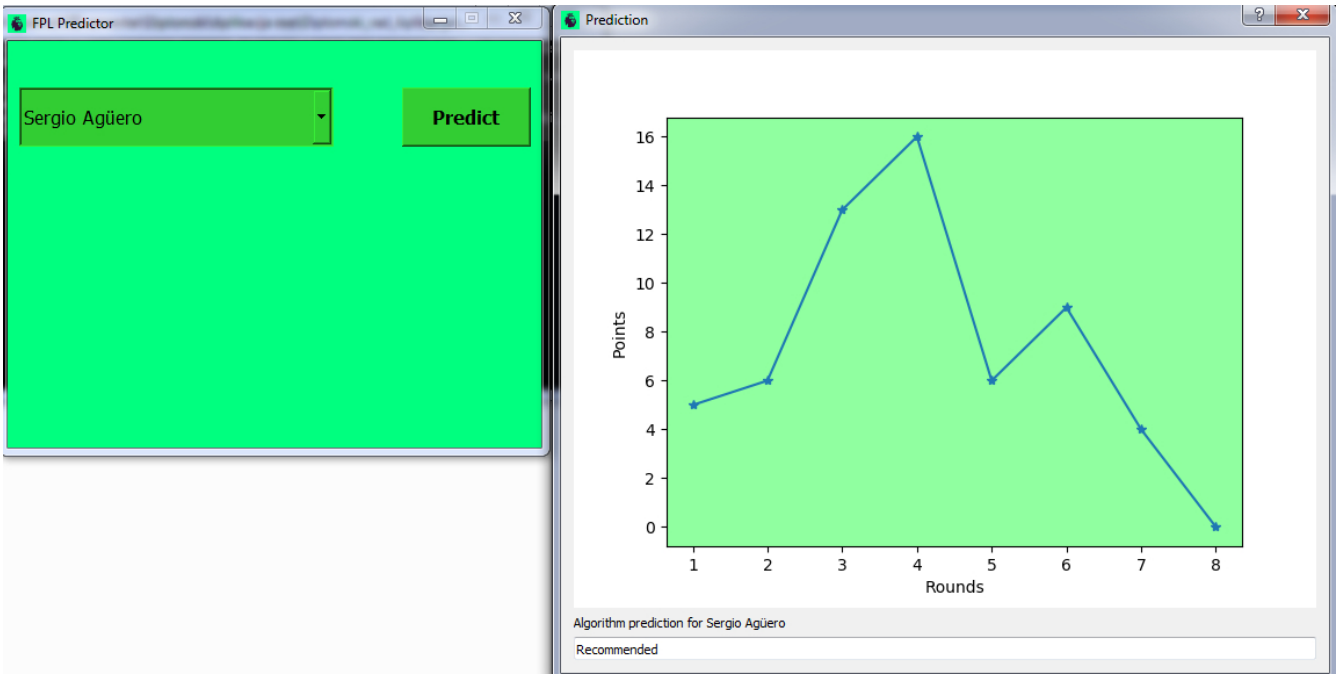
```
class PlayerWindow(QDialog):
    def __init__(self, parent=None):
        super(PlayerWindow, self).__init__(parent)
        self.setWindowTitle('Prediction')
        self.setWindowIcon(QtGui.QIcon('icon.jpg'))
        self.figure = plt.figure()
        self.canvas = FigureCanvas(self.figure)
        self.labelA = QLabel()
        self.textbox = QLineEdit()
        self.textbox.setReadOnly(True)

        playerName = str(dlg.comboBox.currentText())
        self.labelA.setText('Algorithm prediction for ' +playerName)
        keyNumber = [number for number, name in igraci.items() if name == playerName]
        url = "https://fantasy.premierleague.com/api/element-summary/" + str(keyNumber[0]) + "/"
        r  = requests.get(url)
        data = json.loads(r.text)
        value = data['history']
        playerPoints = []
        roundNumbers = []
        goalsScored = []
        cleanS = []
        asissts = []
        bonus = []
        yellowC = []
        for j in value:
            roundNumbers.append(int(j['round']))
            playerPoints.append(int(j['total_points']))
            goalsScored.append(int(j['goals_scored']))
            cleanS.append(int(j['clean_sheets']))
            asissts.append(int(j['assists']))
            bonus.append(int(j['bonus']))
            yellowC.append(int(j['yellow_cards']))

        predictionStats = np.array([np.mean(goalsScored), np.mean(cleanS), np.mean(asissts), np.mean(bonus), np.mean(yellowC)])
        prediction = mlp.predict([predictionStats])

        if(prediction == 0):
            self.textbox.setText("Not recommended")
        elif(prediction == 1):
            self.textbox.setText("Recommended")
        else:
            self.textbox.setText("None")
```

Слика 16. Класа „PlayerWindow“

Предвиђање мреже се приказује кориснику кроз објекат *QLineEdit*, док се график повезује са објектом *FigureCanvas*.

```
114         self.figure.clear()
115
116         ax = self.figure.add_subplot(111)
117
118         ax.set_xlabel("Rounds")
119         ax.set_ylabel("Points")
120         ax.patch.set_facecolor('xkcd:mint green')
121         ax.plot(roundNumbers, playerPoints, '*-')
122
123         # refresh canvas
124         self.canvas.draw()
125
126         # set the layout
127         layout = QVBoxLayout()
128         layout.addWidget(self.canvas)
129         layout.addWidget(self.labelA)
130         layout.addWidget(self.textbox)
131         self.setLayout(layout)
```

Слика 17. Повезивање графика и „FigureCanvas“



Слика 18. Покренута апликација

6. Закључак

Применом техника машинског учења и употребом неуронских мрежа направљен је систем који може значајно да помогне љубитељима игрице „Fantasy Premier League“ у одабиру играча за свој тим. Такође, предлагањем успешних играча алгоритма помаже корисницима да постигну боље резултате у игрици. Структура неуронске мреже са два скривена слоја (15 и 10 неурона) и просечна вредност наведених атрибута су допринели да прецизност класификације буде висока.

Параметар прецизности од чак 93% даје назнаке да је алгоритам постигао оптималне перформансе. Неуронска мрежа се обучава на примерима играча из сезоне 2018/2019, након чега се користи за класификацију актуелних играча. Систем је направљен тако да увек може бити актуелан, јер се користе списак и бодови играча са званичног сајта игрице.

7. Литература

[1] Shai Shalev-Shwartz, Shai Ben-David (2014). *Understanding Machine Learning: From Theory to Algorithms*. New York: Cambridge University Press

[2] Andriy Burkov (2019). *The Hundred-Page Machine Learning Book*. Quebec: Andriy Burkov

[3] Igor Kononenko, Matjaž Kukar (2007). *Machine Learning and Data Mining: Introduction to Principles and Algorithms.* Westergate: Horwood Publishing Limited

[4] David Kriesel (2007). *A Brief Introduction to Neural Networks*. url: **http://www.dkriesel.com**

[5] Nils J. Nilsson (2005). *Introduction to Machine Learning: An Early Draft of a Proposed Textbook.* California: Stanford University