BABEŞ–BOLYAI UNIVERSITY OF CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

Diploma Thesis

Tennis Bet Application

Abstract

The thesis presents a project for helping people to make easier tennis betting decisions. This application facilitates to choose the winner player between two players(men), considering both player's previous matches and their statistics on the related matches. To achieve this, the application uses neural networks.

During the training session the application process more then 100.000 match results and the related statistics (between 1991-2016). One input date contains more then 90 values(stat-rates), which are responsible for returning the wanted/expected results.

The test phase uses the match results and the related statistics from 2017. The application is working with matches just between 1991 and 2017, because I have not managed to find proper data after 2017. So this is an opportunity for improvement.

In the second chapter we are presented the web technologies used on the client side, in which we mainly examine the React programming language.

Chapter 3 deals with the presentation of server-side technologies used in the project, including more details about the Spring Framework.

Chapter 4 provides a more detailed description of the implementation of the project, which will cover the use of the technologies presented in the previous sections.

The next chapter is a user-made documentation, in which will be discussed the operation of the application. The last chapter contains a brief summary and some development plans.

This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

2019 LÁZÁR SZILÁRD

ADVISOR:
ASSIST PROF. DR. SZÖLLŐSI ISTVÁN

BABEŞ–BOLYAI UNIVERSITY OF CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

Diploma Thesis

Tennis Bet Application

ADVISOR:
ASSIST PROF. DR. SZÖLLŐSI ISTVÁN

STUDENT:
LÁZÁR SZILÁRD

2019

UNIVERSITATEA BABEŞ–BOLYAI, CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

Lucrare de licenţă

Aplicatie de pariuri de tenis

CONDUCĂTOR ŞTIINŢIFIC:
LECTOR DR. SZÖLLŐSI ISTVÁN

ABSOLVENT:
LÁZÁR SZILÁRD

2019

BABEŞ–BOLYAI TUDOMÁNYEGYETEM KOLOZSVÁR
MATEMATIKA ÉS INFORMATIKA KAR
INFORMATIKA SZAK

Szakdolgozat

Alkalmazás tenisz fogadáshoz

TÉMAVEZETŐ:

DR. SZÖLLŐSI ISTVÁN,
EGYETEMI ADJUNKTUS

SZERZŐ:

LÁZÁR SZILÁRD

2019

Tartalomjegyzék

1. fejezet

Bevezetés

1.1. Cél

Az alkamazás az egyéni sportágak közül a férfi tenniszmérkőzések eredményeivel foglalkozik, egy jövőbeli mérkőzésről próbál bizonyos adatok felhasználásával egy előrejelzést megfogalmazni, amelyet a mesterséges intelligencia felhasználása által próbál kivitelezni. Az alkalmazás a mesterséges intelligencián belül a neurális hálok felhasználásával egy valószinűségben kifejezett értéket szolgáltat a felhasználók számára, amely a két játékos párbajának kimenetelét próbálja megtippelni, megjósolni.

Az dolgozat és egyben az alkalmazás célja, hogy a sablonos és evidens statisztikák mellett olyan új/más szempontokat is figyelembe vegyen, amelyek felhasználása által a program egy jobb előrejelzést, egy jobb tippet legyen képes nyújtani, mint egy olyan ember, aki megfelelően jártas a témában.

Ennek elérése érdekében, a megszokott és "száraz" statisztikák mellett több olyan érdekesnek tűnhető statisztikai adatot is próbál figyelembe venni, amely hosszútávon egy jobb megközelítéshez vezethet.

Ezt egy webalkamazás formájában próbálja kivitelezni, amely nem csak az adott játékosok kiválasztására és a program lefutása utáni eredmény megjelenítésére alkalmas, hanem segítségével a felhasználó maga is megtekintheti az általa kiválaszott játékosok, tornák, évek mérkőzéseit illetve győzteseit, és azoknak statisztikáit, ezáltal lehetőséget teremt számára, hogy ő maga is értékelje a program által visszatérített eredményt.

1.2. A dolgozat szerkezete

A projekt, lévén hogy webalkalmazásról van szó, két nagy részből tevődik össze: szerver oldali részből és kliens oldali részből. Értelemszerűen mindkét rész részletesen tárgyalva lesz a dolgozatban, viszont a szerver oldali rész, további két részre tagolódik. Ennek a szerver oldali résznek az egyik részét az a szerver fogja alkotni, amely az adatbázissal kommunikálva a statisztikai adatokat szolgáltatja (hozza létre), a másik meg a gépi tanulásért (ezen statisztikai adatok feldolgozásáért) felelős. Mindkét "alrész" különkülön működő alegység, amelyek egymással kommunikálva szolgáltatnak megfelelő és megjeleníthető információkat a kliens oldal számára.

A dolgozat elején a gépi tanulás alapjairól lesz szó, bemutatva annak lényegét, működését és felhasználási körét. A második fejezetben bemutatásra kerülnek a kliensoldalon használt webtechnológiák, amely keretein belül főképp a React rejtelmeibe nyerhetünk betekintést.

A React egy viszonylag új JavaScript keretrendszer, segíti skálázható és könnyen karbantartható alkalmazások létrehozását. Ez a keretrendszer nem használ sablonrendszert az alkalmazás felépítéséhez, hanem egy deklarativ programozási stílust használva definiálja annak aktuális állapotát.

A React keretein belül szó lesz a viszonylag újonnal megjelent React Context API-ról és a React Hook-ról is. Mindezek mellett tárgyalva lesz majd, hogy hogyan létesítünk kapcsolatot a szerver oldali résszel, hogy ott hogyan kommunikálnak egymással a különböző szerverek, és hogy hogyan dolgozzuk fel a tőlük kapott információkat.

Ezt követően a szerver oldali rész részletesebb bemutatása következik. Szó lesz a működéséről, az ott felhasznált technológiákról is egyaránt. Részletesebben tárgyalva lesz a Spring keretrendszer, amely egy nyílt forráskódú, önálló modulokból felépülő keretrendszer. Itt lesznek részletesebben bemutatva a mérkőzések statisztikáinak az előállításai, azoknak kiszámitásai(Java nyelvben) és megfelelő alakba történő átalakításai, a neurális háló és annak megfelelő beállításai (amelyek Python nyelvben iródtak) is egyaránt.

Az ezt következő fejezetekben a projekt megvalósításának részletesebb bemutatására kerül sor, amelyben az előzőleg bemutatott technológiáknak a projektben való felhasználásáról lesz szó.

Az utolsó fejezetben a továbbfejlesztési lehetőségek lesznek tárgyalva, amelyek még jobban hozzásegíthetik a projektet a céljainak az eléréséhez, illetve tovább növelhetik a felhasználói élményt.

2. fejezet

Gépi tanulás - Machine learning

Összefoglaló: *A következõkben a gépi tanulás - a mesterséges intelligencia alapjait mutatjuk be nagyon röviden, amely az alkalmazásunkban Python nyelvben iródott.*

2.1. Rövid ismertetõ

A jelenkor egyik kulcsterülete a gépi tanulás, vagyis a „mesterséges intelligencia.” Egyre több és több program végez „tanulási folyamatot", azaz úgy módosítja működését futás közben, hogy jobban, és eredményesebben végre tudja hajtani a rábízott feladatokat, mintákat keresve az adatok között: például az arcfelismerés, szövegértés, önvezető autók stb.

Ez a gyakorlatban azt jelenti, hogy a rendszer az adatok és minták alapján képes arra, hogy önállóan fel tudjon ismerni vagy meghatározni bizonyos szabályokat. Tulajdonképpen a rendszer nem csupán betanulja „kívülről” a mintákat, hanem képes ezek alapján olyan általánosításokra, ami alapján a tanulási szakasz végzetével számára ismeretlen adatokkal dolgozva is „helyes” döntéseket képes meghozni.

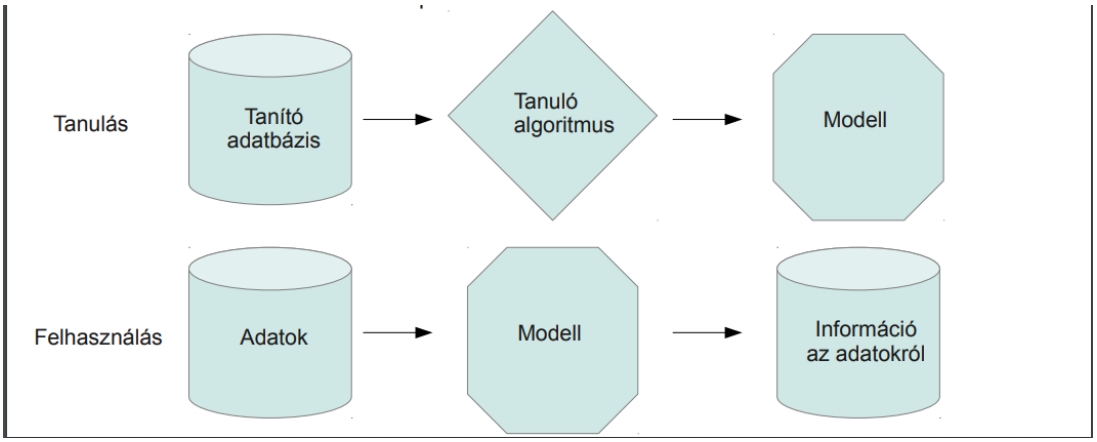
2.2. Neurális hálók

A Neurális hálózatok az emberi ideghálózat működését próbálják szimulálni. Modern használatban ezen kifejezés alatt a mesterséges neurális hálót értjük, amelyek mesterséges neuronokból állnak. A működési elve hogy egy több szintes hálózaton a „neuronok” a bemenő adatok alapján a megfelelő képletek végrehajtása során a megfelelő eredményhez vezetnek vagy sem.

Ezeknek a hálózatok alapelve, hogy a számolásokat neuronok (egymással összekapcsolt kis feldolgozóegységek) végzik. A számítások folyamán a neuronok közötti kapcsolatrendszer fontos szerepet játszik.

Érdemes a neurális hálózatok használata, ha:

1. Sok összefüggő bemenő adat-, összefüggő kimeneti paraméter áll rendelkezésre
2. A megoldandó problémával kapcsolatban gazdag adathalmaz áll rendelkezésre
3. A rendelkezésre álló adathalmaz nem teljes, hibás adatokat is tartalmazhat
4. A megoldáshoz szükséges szabályok ismeretlenek



2.1. ábra. Gépi tanulás általában

A neurális hálózat egyszerű egységekből áll olyan értelemben, hogy belső állapotai leírhatók számokkal (aktivációs értékek). Az egységek egyenként létrehoznak egy aktiválási értéktől függő kimeneti értéket és csatlakoznak egymáshoz. Mindegyik csatlakozás tartalmaz egy egyéni súlyt amelyek szintén számokkal vannak kifejezve. Ezen egységek mindegyike kiküldi a kimeneti értékét az összes többi egységnek, amelyekkel kimenő kapcsolatban vannak. A "rendszer" bemenetei lehetnek mesterséges szenzorok vagy akár érzékelők adatai, míg kimenetei lehet a viselkedés egy kimeneti neuronon. A kapcsolatok miatt az egység kimenete hatással van a másik egység aktivációjára. A bemeneti oldalán álló egység fogadja az értékéket, és azok súlyozásával kiszámolja az aktivációs értékét (összeszorozza a bemeneti jelet a hozzá tartozó bemenet súllyal,majd összegét számol) A kimenetet az aktivációs függvény határozza meg az aktivációtól függően.

2.2.1. A neuron

A neuronnak több bemeneti változója lehet. A neuron által adott predikció a bemenő értékek, és a súlyok szorzata, ami egy lineáris függvényt ad, majd ennek eredményére a szigmoid függvény van alkalmazva, mint aktivációs függvény. A bemeneti adatokat x-szel vannak jelölve(x=(x0, x1, x2)), míg a súlyokat a w-vektor jelöli. A h - hipotézisfüggvény: a neurális hálózat kimenete egy adott x - bemenetre, w - súlyok mellett.

A szigmoid függvény:

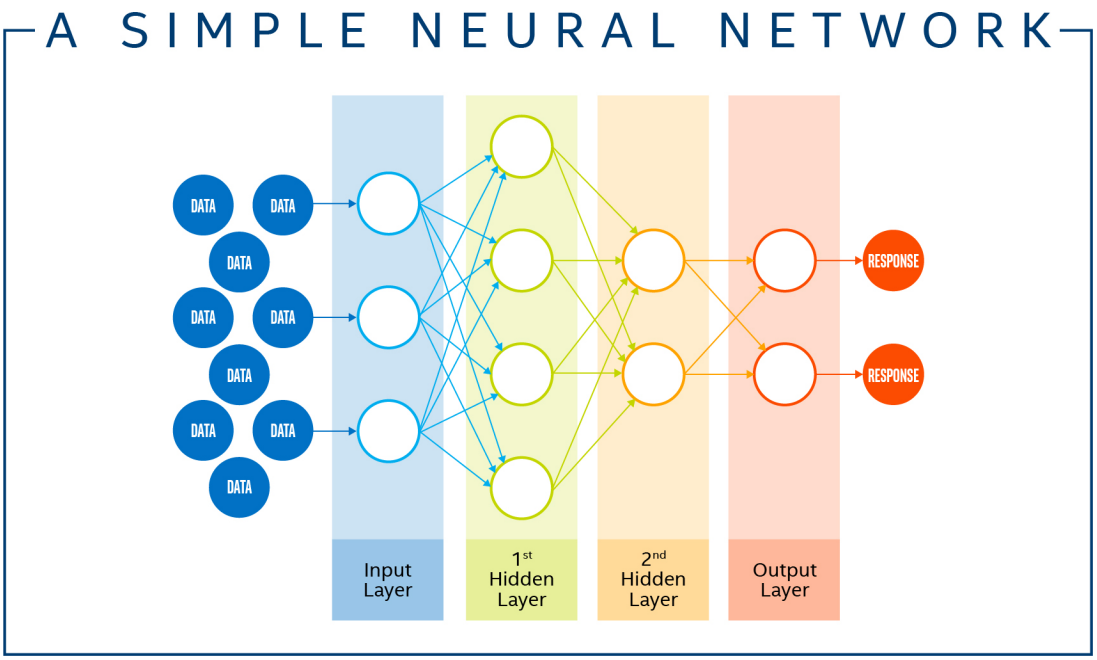
$$g(x) = \frac{1}{1 + e^{-x}}$$

2.2.2. Egy egyszerű neuron implementációja

1. Inicializálás: véletlenszerű kezdő súly értékek beállítása

2. Előreterjesztés (feed forward): kiszámolni a neuron kimeneteit minden egyes bemenetre.

$$g(x) = \frac{1}{1 + e^{-x}}$$



2.2. ábra. Neurális háló és különböző rétegei - Azonos a mi alkalmazásunk hálózatával

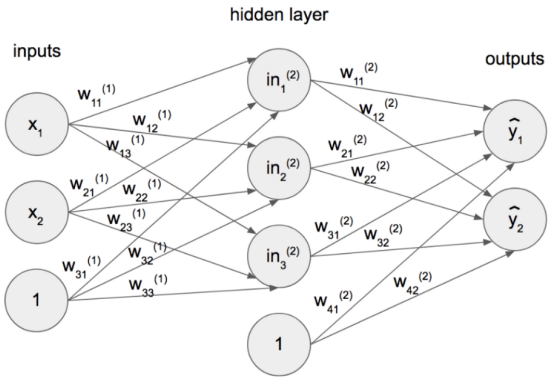
$$Hw(x) = g(x * w)$$

3. Hiba-visszaterjesztés (backpropagation): A hibafüggvény (J) aktuális értéke az aktuális súlyoktól függ. Ha az i. súlyt változtatjuk (valamit hozzáadunk/kivonunk), akkor ki tudjuk számolni, hogy mennyit változik a hibafüggvény értéke, ha a többi súlyt változatlanul hagyjuk. Ebből kiszámolható, hogy megközelítőleg mekkora a hibafüggvény meredeksége az. i. súly mentén, ha a többi súly változatlanul marad. Tehát tulajdonképpen a hibafüggvény parciális deriváltját számítjuk ki. Majd eztkövetően a parciális deriváltakból egy oszlopvektort állítunk ki. Az oszlopvektor sorainak száma megegyezik a súlyok számával, vagyis a bemenő változók számával. A kiszámolt vektort megszorozzuk a tanulási rátával, és kivonjuk a súlyvektor jelenlegi értékéből, amellyel megkapjuk az új súlyvektort.

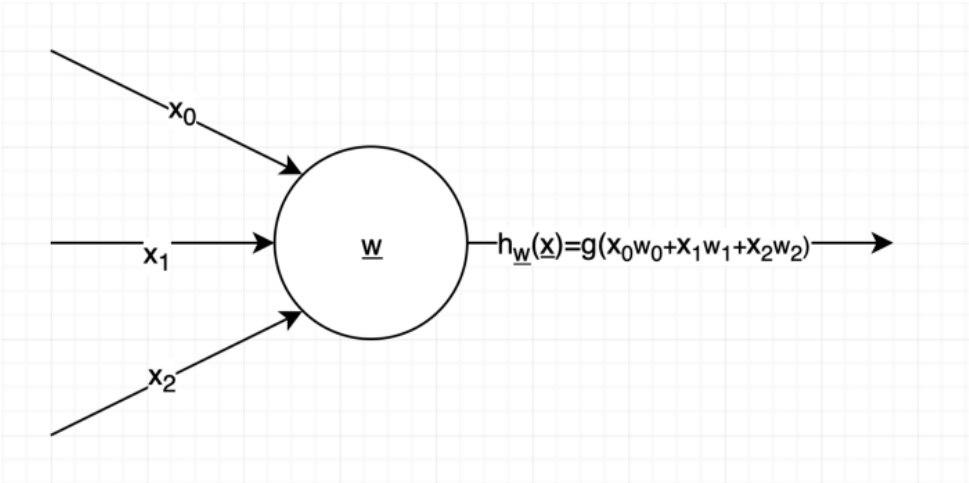
4. Az előző három lépést ismételve jó közelítéssel megtalálhatjuk a hibafüggvény minimumhelyét, tehát azokat a súlyokat, ahol a hibafüggvény a legkisebb. A hibafüggvény, J (y jelöli a tényleges eredményt, h pedig a hipotézist):

$$J(w) = \frac{1}{n}\sum_{i=0}^{+n}(y - hw(x))^2$$

5. A súlyokat minden iterációnál frissítem, így lépésről lépésre eljutok a hibafüggvény minimum értékének a közelébe



2.3. ábra. Neurális hálózat súlyokkal, input és output adatokkal, illetve egy középső réteggel



2.4. ábra. Az alábbi ábrán egyszeres aláhúzás jelöli a vektor típusú változókat (sorvektorokat és oszlopvektorokat)

3. fejezet

React keretrendszer

A React [1] nem más, mint egy JavaScript könyvtár, ezért feltételez egy alapvető JavaScript ismeretet,viszont használata megkönnyíti a felhasználói felületek létrehozását és karbantartását. Segítségével a kész felhasználói felület lebontható kisebb, újrahasznosítható komponensekre (react komponensekre). Megalkotója Jordan Walke, aki Facebook egyik szoftverfejlesztője(2011). Egy konferencián, amelyet 2013-ban nyílt forráskódúvá nyilvánították. Megalkotásának fő motivációja a Facebook gyors növekedése volt, amelynek hatására a régi rendszerek nem tudták tartani a lépést a fejlődéssel, mivel egyre körülményesebbé vált a fejlesztés. A keretrendszer gyors fejlődése következtében a React Native megjelenése által már Android és iOS készülékekre való fejlesztésekre is alkalmassá vált.

A keretrendszer legnagyobb erénye, hogy virtuális DOM (Document Object Model)-ot használ. A Virtual DOM nem mást csinál, mint új rétegként beékelődik a kódunk és a DOM közé (vagy egy natív vezérlő hierarchia mögé), csokrokba gyűjti a DOM-on végzett műveleteket, és optimálisabban hajtja végre azokat, figyelve a változásokra.

Tehát amikor módosítás történik az alkalmazás állapotában, akkor mindössze ennek a DOM-nak a legkisebb részfája lesz frissítve.

3.1. Használata

Ahhoz, hogy el tudjuk készíteni első alkalmazásunkat, először telepítenünk kell a NodeJs. Ha ezzel megvagyunk, a node package manager (npm) segítségével létrehozhatjuk és futtathatjuk az alkalmazásunkat a következő parancsok által:

```
npm install -g create-react-app
create-react-app my-app
cd my-app/
npm start
```

Ha végrehajtottuk a fenti utasítássorozatot, akkor összes olyan csomag letöltődik, amelyek szükségesek a kezdetleges alkalmazásunk futtatásához, illetve létrejönnek a szükséges konfigurációs állományok is. Természetesen ha úgy szeretnénk, akkor saját magunk is letölthetjük a szükséges csomagokat, illetve létrehozhatjuk a megfelelő állományokat, viszont ez az út sokkal hosszadalmasabb, bár esetenként akár

1. A React hivatalos oldala: https://reactjs.org/

hasznosabb is lehet, főleg ha már jártasak vagyunk a témában, és pontosan tudjuk, hogy mely csomagokra lesz szükségünk a továbbiakban.

A fejlesztés megkezdése előtt érdemes jól megválasztani a fejlesztői környezetünket. Természetesen nincs egy olyan környezet sem, amely mindenki számára teljesen megfelelő lenne, hiszen a fejlesztők más-más tulajdonságokat helyeznek előtérbe, ebből kifolyólag eltérő környezeteket is használnak, viszont nagyon jó alternatívák mindezek közül a JetBrains[2] által közzétett IntelliJ, vagy a Microsoft által fejlesztett Visual Studio Code[3]. Fontos megemlíteni, hogy az IntelliJ használata nem ingyenes, tehát ha használni szeretnénk, akkor fizetnünk kell, viszont létezik egy olyan próbaverziója, amely diákok számára ingyenesen elérhető.

3.2. React Context

Egy tipikus React alkalmazás esetén az adatok fentről-lefele (a szülőtől a gyerek fele) vannak átadva props-okon keresztül, viszont ez sok esetben nem optimális, például ha olyan információkat szeretnénk átadni, amelyeket nagyon sok különböző komponensben szeretnénk egyaránt használni, mivel akkor ezen paraméterátadások száma nagyon megnövekedne (például: UI theme, locale preference). A Context[4] épp ehhez nyújt egy egyszerűbb alternatívát, vagyis lehetőséget ad értékek/adatok megosztásához a különböző komponensek között anélkül, hogy át kellene adnunk őket a props-okon keresztül az összes szinten.

3.2.1. Használata

Mint ahogy a fentiekben is elhangzott, a Context arra volt kitalálva, hogy könnyebben tudjunk bizonyos adatokat globálisan is megosztani a különböző komponensek között, mint például a pillanatnyilag bejelentkezett felhasználó, témák, használt nyelv, stb.

Viszont gondoljuk át többször is a Context használatát, mielőtt mindent általa próbálnánk megvalósítani, mivel nem minden esetben célszerű a használata. Használata célravezető, amikor különböző mélységi szinteken (data needs to be accessible by many components at different nesting levels) is szükségünk van ugyanazokra az információkra, viszont használata nagyban veszélyezteti az adott komponens újrafelhasználhatóságát.

3.2.2. Context API

```
1 const PlayerContext = React.createContext(defaultValue);
```

Listing 3.1. React Create Context

2. A JetBrains hivatalos oldala: https://www.jetbrains.com/idea/
3. A Visual Studio Code hivatalos oldala: https://code.visualstudio.com/
4. A React Context oldala: https://reactjs.org/docs/context.html

A fenti kódrészletben létrehoztunk egy Context objektumot. Amikor a React kirendereli a komponenst, feliratkozik (subscribes) ehhez a Context objektumhoz, és az aktuális kontext értékét a hozzá legközelebb álló egyező Providerből fogja kiolvasni.

```
<PlayerContext.Provider value={{
  ...this.state,
  changeSelectedPlayer: this.changeSelectedPlayer,
  changeTournaments: this.changeTournaments,
  changeGrandSlams: this.changeGrandSlams,
  changeLastMatches: this.changeLastMatches}}>
      { this.props.children }
</PlayerContext.Provider>
```

Listing 3.2. React Context Provider

A Contex alapértelmezett értéke (defaultValue) csupán abban az esetben lesz használva, ha a komponensnek nincs egyező Providere a fenti fában. Ez hasznos lehet olyan esetekben, amikor tesztelni szeretnénk a komponenseinket önmagukban anélkül, hogy "becsomagolnánk" őket.

Figyelem: ha UNDEFINED értéket adunk át a Providernek, akkor az nem fogja használni az alapértelmezett értékként megadott adatokat.

Mindegyik Context objektum egy Provider react komponenssel együtt jár, amelyek megengedik az őt fogyasztó (használó) komponenseknek, hogy feliratkozzanak az adott context változásaira. Egy Provider több, őt használó komponenssel is kapcsolatban állhat, illetve a Providerek egymásba is ágyazhatóak.

```
class App extends Component {

  render() {
    return (
      <AppContextProvider>
          <PlayerContextProvider>
            <TournamentContextProvider>
              <PredicterContextProvider>
                <NavigationBar>
                  <Router />
                  <Footer />
                </NavigationBar>
              </PredicterContextProvider>
            </TournamentContextProvider>
          </PlayerContextProvider>
      </AppContextProvider>
    );
  }
}
```

Listing 3.3. A Providerek egymásba ágyazása

A Context használata egy belső komponensben a következő egyszerű módon történik:

```
const playerContext = useContext(PlayerContext);
...
const selectedPlayer = playerContext.selectedPlayer;
```

Listing 3.4. A Context értékének a felhasználása

3.3. React Hooks

A Hook[5]-ok a React 16.8-as verziójától számolva elérhetőek, amelyeknek egyik fő előnye, hogy megengedi a state használatát anélkül, hogy osztályokat kellene írnunk.

Az egyik legegyszerűbb, viszont leggyakrabban használt Hook a "useState". Egy függvényen belül kell hívnunk, amely visszatérít egy kételemú tömböt, amelynek elemei egy state változó, és a hozzá tartozó setter függvény.

```
const [player, setPlayer] = useState({});
const [age, setAge] = useState(42);
```

Listing 3.5. useState hook-nak a használata

Ahogyan azt a 3.5-ös kódrészletből is láthatjuk, megadhatunk alapértelmezett értékeket is az újonnal létrehozott változóknak a useState használtával.

De hogy micsodák is a Hook-ok? A Hook-ok föggvények, amelyek megengedik a react state-k és életciklus metódusok (lifecycle methods) használatát függvényekből anélkül, hogy osztályokon belül dolgoznánk.

Fontos megjegyezni, hogy a Hookok használata nem követeli meg, hogy az alkalmazásunkban minden egyes komponenst átírjunk úgy, hogy a Hook-okat használja, hanem tetszőlegesen használhatjuk őket, mivel visszafele is teljesen kompatibilisak.

Mindezek mellett használjuk őket a felső szinten, illetve kerüljük a ciklusokon, feltételeken, vagy beágyazott függvények belsejében történő használatukat. Csak React függvény komponensekből hívjuk őket és ne reguláris JavaScript függvényekből.

3.3.1. Effect Hook

Ezeknek a segítségével/felhasználásával küldhetünk és fogadhatunk kéréseket (fetching), vagy manipulálhatjuk a DOM szerkezetét. Tulajdonképpen ugyanazt a célt szolgálja, mint a componentDidMount, componentDidUpdate, componentWillUnmount a React osztályok esetén, viszont itt ezek egyesítve vannak egyetlen API-ba.

```
useEffect(() => {
    get_request(`${DEFALULT_SERVER_URL}/player/all`)
        .then(players => {
            setState({...state, players: players})
        })
}, []);
```

Listing 3.6. useEffect hook-nak a használata

Amikor meghívjuk a useEffect effektust, tulajdonképpen azt mondjuk a React-nek, hogy futtasd le a funkciót, miután feldolgoztad a DOM-ot. Ezeket az effekteket a komponens belsejében deklaráljuk, tehát hozzáférésük van a state-hez és a props-hoz egyaránt.

5. A React Hook hivatalos oldala: https://reactjs.org/docs/hooks-overview.html

Alapértelmezetten a React a render metódus minden lefutása után végrehajtja ezeket az effekteket, beleértve az elsőt is, viszont ez szabályozható is. Például a fenti, 3.6-os kódrészletben szereplő effect mindössze az első render lefutása után lesz lefuttatva annak köszönhetően, hogy a kódrészlet utolsó sorában egy üres tömböt adtunk meg paraméterként.

Lehetőségünk van úgy is személyreszabni ezeket az effekteket, hogy csak bizonyos adatok megváltozása következtében hivódjanak meg a különböző effektek, amelyet az már fent emlitett üres tömbnek, az adott változóval való kicserélésével érhetünk el.

4. fejezet

Spring keretrendszer

Az elmúlt években figyelemmel kísérhettünk egy sikertörténetet, a Spring[1] keretrendszer fejlődésének és elterjedésének történetét.

Ez a keretrendszer évekig nem terjedt el, mert helyette ott volt a J2EE majd Java EE technológia és biztonság szempontból kritikus alkalmazások fejlesztésére szinte kizárólag ezeket használták.

Kezdetben az volt a cél, hogy az egyéb keretrendszerek hiányosságait kiküszöbölje, napjainkban azonban szinte minden olyan funkcionalitást és szolgáltatást tartalmaz, amelyekre egy webalkalmazásnak vagy bonyolultabb üzleti alkalmazásnak szüksége lehet. Ez a fejezet ennek a keretrendszernek egy rövidebb bemutatását tartalmazza, kicsit nagyobb hangsúlyt fektetve annak webes részére.

4.1. Általános tudnivalók

A Spring egy inversion of controlt megvalósító, nyílt forráskódú Java alkalmazás keretrendszer. Ezen keretrendszer szolgáltatásait főként Java alkalmazás fejlesztésére használják. Nincs specifikált fejlesztési modellje, napjainkban az EJB(EnterpriseJavaBean) modell egyre népszerűbb alternatívájává(helyettesítőjévé) vált. A Spring moduláris jellegéből adódóan, csak azokat a részeit kell használnunk, amelyekre valóban szükségünk van, és nem kell belevonnunk a projektünkbe a nem használt részeket. Támogatja a deklaratív tranzakciókezelést és teljes értékű MVC keretrendszert biztosít.

4.2. Inversion of control

Mivel egy Java alkalmazás tipikusan objektumokból áll össze, és ezen objektumoknak az együttműködése alakítja ki magát az alkalmazást, ezért a benne található objektumok függenek egymástól. Annak érdekében, hogy alkalmazásunkat egy összefüggő egésszé szervezzük használhatunk különböző tervezési mintákat, mint például Factory (gyár), Abstract Factory (absztrakt gyár), Service Locator (szolgáltatás lokátor), és így tovább, amelyekkel különböző objektumpéldányokat hozhatunk létre az alkalmazás felépítése érdekében, viszont ezek csupán minták, legjobb megoldások, amelyeknek van egy elnevezésük, leírásuk, hogy melyik milyen problémát orvosolhat. Tehát elmondhatjuk, hogy a minták tulajdonképpen formalizált megoldások, amelyeket kénytelenek vagyunk magunk implementálni.

1. A Spring framework hivatalos oldala: https://spring.io/

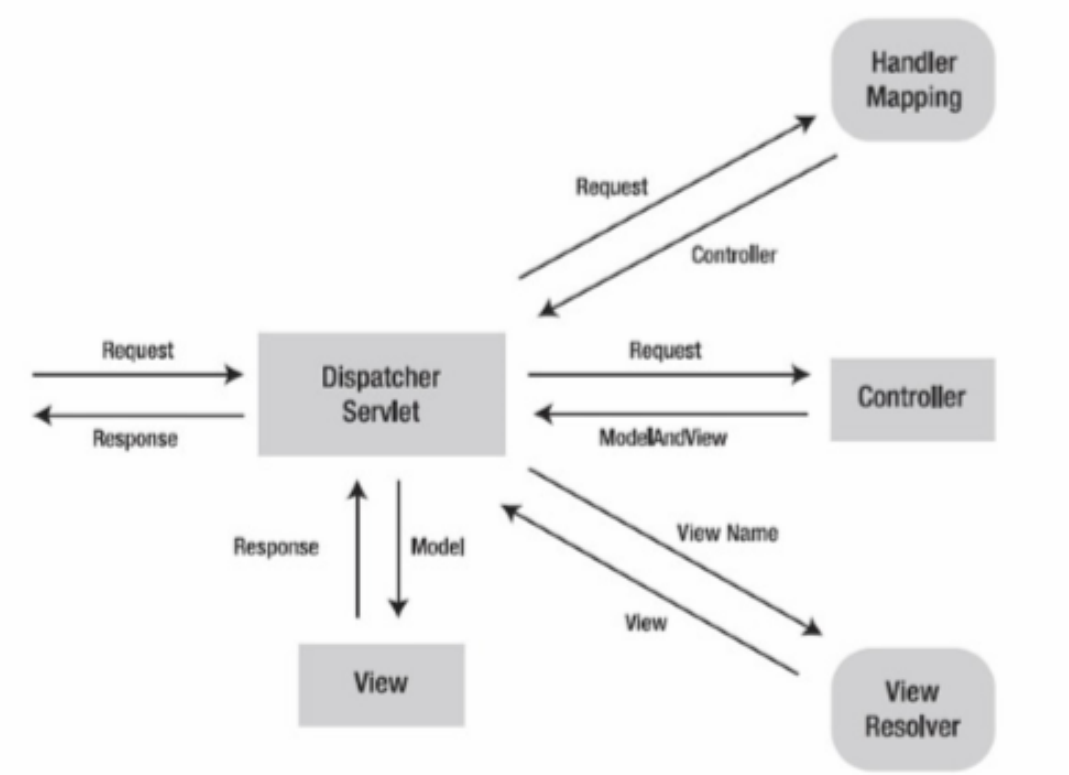
A Spring keretrendszer Inversion of Control(Vezérlés invertálása) komponense ezt a problémát oldja meg úgy, hogy ezeket az eltérő komponenseket formalizált módszerekkel egy használatra kész alkalmazássá állítja össze. A Spring első osztályú objektumokká kódolja át a formalizált tervezési mintákat, amelyeket így már integrálhatunk a saját alkalmazásainkba.

A Springben az alkalmazások gerincét alkotó objektumokat, amelyeket a Spring IoC konténer kezel, bean-nek nevezik. A bean egy olyan objektum, amelyet egy Spring IoC konténer példányosít, állít össze és kezel. Egyébként a bean egyszerűen az alkalmazás egyik objektuma. Az org.springframework.beans és az org.springframework.context csomagok képezik a Spring Framework IoC konténerének alapját. A Bean Factory interfész fejlett konfigurációs mechanizmust biztosít, amely képes bármilyen típusú objektum kezelésére. Az ApplicationContext a BeanFactory alinterfésze, amely könnyebben integrálható a Spring AOP funkcióival.

4.3. Spring Web MVC framework

A Spring Web MVC4 lehetővé teszi számunkra, hogy rugalmas alkalmazásokat készítsünk. A Spring Web MVC tulajdonképpen egy, a Dispatcher Servlet köré tervezett keretrendszer, amely kéréseket küld a kezelőknek kofigurációs kezelői leképzésekkel, nézetfelbontással, helyi időzónákkal és témafelbontással, valamit támogatja az MVC (model-view-controller) tervezési mintát, amelynek lényege a modell és a nézet szétválasztása, annak érdekében, hogy a felhasználói felület ne tudja befolyásolni az adatkezelést. Az alapértelmezett kezelő a @Controller és @RequestMapping annotációkonalapul, amelyek rugalmas kezelési módszerek széles skáláját kínálják.

A Spring nézetfelbontása nagyon flexibilis. Az esetek nagy részében a Controller felelős a Map modell adatainak elkészítéséért és a View nevének kiválasztásáért, de közvetlenül írhat a válaszfolyamra és kiegészítheti a kérést is. A modell (amely M az "MVC"-ben) egy Map interfész, amely lehetővé teszi a nézettechnológia teljes absztrakcióját. A Spring Web MVC keretrendszere, mint sok más webes MVC keretrendszer, kérésvezérelt. Egy központi Servlet köré van tervezve, amely kéréseket küld a Controllernek és ezek mellett olyan funkciókat tartalmaz, amelyek megkönnyítik a webes alkalmazások fejlesztését. A Spring DispatcherServlet azonban nem csak ennyiből áll, hanem teljesen integrálva van a Spring IoC konténer által, ezentúl minden más Spring jellemzőt használni tud.



4.1. ábra. A Spring Web MVC Dispatcher Servlet kérelem feldolgozási munkafolyamata

5. fejezet

Az alkalmazás felépítése

Összefoglaló: *Mint ahogyan az előző fejezetekből is megtudhattuk, az alkalmazás lényege, hogy egy, az általunk kiválasztott mérkőzésről, a kiválasztott játékosok formájától és addigi statisztikáiktól függően egy jóslatot tárjon elénk, amelyben a játékosok százalékos győzelmi esélyeit jeleníti meg.*

Ez az alkalmazás két fő részből áll, amelyek közül az egyik szintén tovább bontható:

- *Kliens oldali rész*
- *Szerver oldali rész*
  - *Spring: adatbázis műveletek*
  - *Python: neurális háló*

Az alkalmazás működését tekintve a következőképpen működik: a felületről érkező kéréseket a Java szerver kezeli, majd a neurális hálóval kapcsolatos műveletek elvégzése érdekében a Java szerver egy kérést küld a Python szerverünknek, amely a megfelelő utasítások végrehajtása során válaszol a Java szerverünknek, majd az a kliens oldalon futó szervernek, amely megjeleníti a kapott információkat.

A szerverek közti kommunikáció megkönnyítése érdekében a Java oldalon történő tréning adatok kiszámítását követően az adatokat egy JSON alakú fájlba írjuk, majd egy kérést küldünk a Python szerverünknek az állomány nevével, amely ezt követően beolvassa az adott állomány tartalmát, majd elvégzi a kért műveleteket.

5.1. A kliens oldal

A kliens oldal a React keretrendszer által lett megvalósítva. Az alkalmazás lehetőséget nyújt a felhasználók számára, hogy a tennis 4 nagy tornájának (Grand slams) a nyerteseit, mérkőzéseit megtekintjük táblázatos formában. Ugyanakkor lehetőségünk van tetszés szerint a tornák neve alapján is rákeresni bizonyos eseményekre, tornákra.

Ugyanúgy mint a tornák esetén, a játékosok esetében is lehetőségünk van a név szerinti keresésre, ahol a kiválasztott játékos karrierjének összes fontosabb mozzanatáról találhatunk néhány érdekesebb statisztikai adatot.

Az alkalmazás legérdekesebb és egyik legfontosabb része, a játékosok párba/szembeállítása, amikor az egymással szembeni, egymáshoz viszonyított statisztikai adatokat tárja elénk az alkamazás. Ennél a résznél van lehetőségünk a párharc kimenetelének a "megjósoltatására" is.

5.1.1. CSS keretrendszer - Bootstrap

Az alkalmazás reszponzív, azaz egy olyan alkalmazás, amely törekszik arra, hogy optimális megjelenítést biztosítson a könnyű olvashatóság és egyszerű navigációval egyidőben. Ezt különböző eszközökön egyaránt próbálja fenntartani az asztali számítógépek képernyőjétől a mobiltelefonok kijelzőjéig egyaránt. Azaz a webalkalmazás alkalmazkodik az őt használó eszköz méreteihez:

- A rugalmas felosztású koncepció alapján a honlap minden elemének mérete százalékosan, relatívan van meghatározva.
- A flexibilis képek úgyszintén a befoglaló elemhez képest, százalékosan határozódnak meg.
- A media query alkalmazásával megvalósíthatjuk, hogy a weboldalon mindig olyan CSS szabályok lépjenek érvénybe, amelyek a megjelenítő eszközön optimálisak.

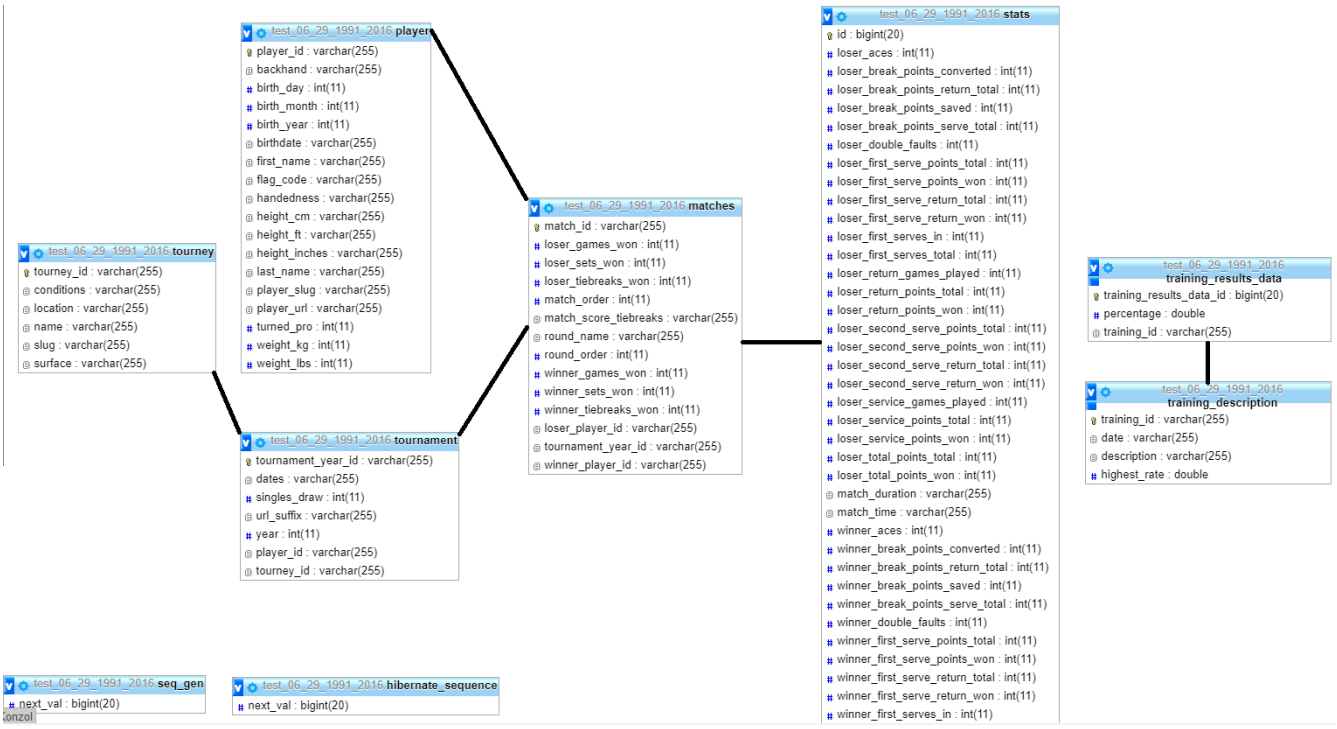
A Bootstrap[1] egy olyan eszközkészletet kínál, amely előre megírt, multifunkcionálisan alkalmazható, aminek a segítségével átláthatóbban, gyorsabban és hatékonyabban dolgozhatunk. A CSS tulajdonságok és a HTML struktúra mellett számos JavaScript bővítménnyel is rendelkezik, melyek igen rugalmasak!

```
<ul className="nav nav-pills mb-3" id="pills-tab" role="tablist">
  <li className="nav-item">
    <a className="nav-link active font-weight-bold" id="pills-latest-tab" data-
       toggle="pill" href="#pills-latest" role="tab" aria-controls="pills-latest"
       aria-selected="true">
      Latest
    </a>
  </li>
  <li className="nav-item">
    <a className="nav-link font-weight-bold" id="pills-grand-slam-tab" data-toggle="
       pill" href="#pills-grand-slam" role="tab" aria-controls="pills-grand-slam"
       aria-selected="false">
      Grand Slams
    </a>
  </li>
</ul>
```

Listing 5.1. Responsive webtervezés - Bootstrap keretrendszer

A kliensoldal felépítése során fontos szempontnak számított az újrafelhasználható komponensek használata, ezért az alkamazásunkban több helyen is ugyanazokat a komponenseket használtuk, ezáltal is törekedve a kód méretének minimalizálására.

1. A Bootstrap hivatalos oldala: https://getbootstrap.com/



5.1. ábra. Az adatbázis szerkezete

5.2. Szerver oldal

5.2.1. Java - Spring

Az alkalmazásunkban a szerver oldal Java-Spring része felelős az adatbázissal való kommunikációért. Itt hajtódnak végre az adatbázisból nyert adatok feldolgozásai annak érdekében, hogy olyan adatokat kapjunk, amelyeket a neurális háló számára tréning adatként bocsáthatunk.

A projekt megvalósítása során az adatbázis és az adatbázis séma közti függetlenség kialakítása érdekében ORM keretrendszert használtunk, pontosabban RedHat által fejlesztett Hibernate-t. A Hibernate egy objektum-relációs leképezést megvalósító programkönyvtár Java platformra, amelynek segítségével relációs adatbázisok tábláit és osztályokat tudunk egymásba leképezni, általa az adatbázisban lévő rekordokat úgy kezelhetjük mint objektumok. A leképzések az osztályok és az adattáblák között annotációk (mint ahogyan a projektben is használtuk) és XML állományok által is megvalósítható.

Az adatbázisból lekért adatokat olyan értékekké(input adatokká) kell alakítanunk, hogy a neurális háló számára felhasználhatóak legyenek. Ez a [0,1] intervallumot jelenti a mi esetünkben. Ezeket az átalakítások a következőképpen történnek: a kiválasztott játékosoknak lekérjük az utolsó, már előre meghatározott számú mérkőzéseiket, majd különböző valószínűségeket, statisztikákat számolunk mindkét játékos számára (5.2-es kódrészlet).

```
private double convertToPercentage(List<Match> matches, Player player){
  if (matches.size() == 0) return 0.00;
  long counter = matches.stream()
    .filter(match -> match.getWinnerPlayer().getPlayer_id() == player.getPlayer_id()
        )
    .count();

  return (matches.size() > 0) ?
    Double.parseDouble(df.format((double) counter / (double) matches.size())) :
    0.0;
}
```

Listing 5.2. Egy játékos utolsó mérkőzéseinek győzelmi rátája

Az input adatok listája		
Number	Name	Level
1	Winning Rate	ALL + Tournament + Surface
2	1 vs 1 Winning Rate	ALL + Tournament + Surface
3	Round experience	ALL + Tournament + Surface
4	Match Duration Avg	Tournament
5	Service Points Won Rate	ALL + Tournament + Surface
6	First Serve In Rate	ALL + Tournament + Surface
7	First Serve Won Rate	ALL + Tournament + Surface
8	Break Points Converted Rate	ALL + Tournament + Surface
9	Break Points Saved Rate	ALL + Tournament + Surface
10	Second Serve Points Won Rate	ALL + Tournament + Surface
11	Second Serve Return Won Rate	ALL + Tournament + Surface
12	Games Won Rate	ALL + Tournament + Surface
13	Sets Won Rate	ALL + Tournament + Surface

A fenti táblázatban azok a szempontok vannak felsorolva, amelyeket a mérkőzések kimenetelének megjósolása alkalmával figyelembe vettünk. A táblázat harmadik oszlopa azt mutatja, hogy az adott "statisztikai mutatót" a játékos összes meccsére (ALL), az adott tornán lejátszott meccseire (Tournament) vagy az adott talajon lejátszott meccseire (Surface) számítottuk ki és alkalmaztunk.

```
public double getSecondServeReturnWonRate(List<Stats> playerStats, Player player){
  playerStats = playerStats.stream()
    .filter(st -> !((double)st.getWinner_second_serve_return_total() < 0.0001 || (
        double) st.getLoser_second_serve_return_total() < 0.0001))
    .collect(Collectors.toList());

  double playerSecondServeReturnWonRate = (playerStats.stream()
    .map(s -> {
      if(s.getMatch().getWinnerPlayer().getPlayerSlug().equals(player.getPlayerSlug
          ())){
        return s.getWinner_second_serve_return_won() / (double)s.
            getWinner_second_serve_return_total();
      } else {
        return stat.getLoser_second_serve_return_won() / (double)s.
            getLoser_second_serve_return_total();
      }
    })
```

```
    .reduce(0.00, (a, b) -> a + b));

  if (playerStats.size() == 0) {
    throw new NumberFormatException();
  }

  playerSecondServeReturnWonRate = playerSecondServeReturnWonRate / (double)
      playerStats.size();
  return Double.parseDouble(df.format(playerSecondServeReturnWonRate));
}
```

Listing 5.3. A második adogatások utáni pontok győzelmi rátája fogadóként

```
    private List<TrainData> getTrainData(List<Match> matches){
        return matches.stream()
                .filter(match -> getInputs(match) != null)
                .map(match -> new TrainData(getInputs(match), getOutputs(match)))
                .collect(Collectors.toList());
    }
```

Listing 5.4. Az adatbázisban talált mérkőzések training adatokká való átalakítása

Minden mérkőzés esetén, amikor a neki megfelelő input adatok kiszámításán dolgozunk, csak az előtte lévő X mérkőzés adatait vesszük figyelembe. A mérkőzések közül csak azon meccsek maradnak a traininghez felhasznált mérkőzések között, amelyek teljesítenek bizonyos feltételeket, például egy minimum mérkőzésszám az adott játékos karrierjében, az adott talajon, tornán stb. Ez azért fontos, mivel a beválogatás során a legmegfelelőbb adatokat érdemes megtartanunk, hogy minél jobb eredményeket érhessünk el.

Abban az esetben, ha az adott mérkőzés nem felel meg az elvárásoknak, a "getInputs" metódus null értéket fog visszatéríteni, és mint ahogyan az 5.4-as kódrészletben is láthatjuk, az ilyen mérkőzések nem fognak bekerülni a training adataink közé.

Annak érdekében, hogy a training adatok létrehozásának folyamatát felgyorsítsuk, a játékosok mérkőzéseit változókba (5.5-ös kódrészlet) mentettük, amely azért lehet hasznos, mert a mérkőzések egyenként való feldolgozása alkalmával minden játékos mérkőzéseit így egyetlen alkalommal kell lekérnünk az adatbázisból, amely lényegesen hozzájárul a futásidő minimalizálásához (5.6-os kódrészlet).

```
    private HashMap<String, List<Match>> matchMap = new HashMap<>();
    private HashMap<String, List<Match>> matchMapOnSurface = new HashMap<>();
    private HashMap<String, List<Match>> matchMapHeadToHead = new HashMap<>();
    private HashMap<String, List<Match>> matchMapHeadToHeadOnSurface = new HashMap
        <>();
    private HashMap<String, List<Match>> matchMapOnTournament = new HashMap<>();
```

Listing 5.5. A futásidő minimalizálásához használt HashMap-ek listája

```
List<Match> winnerPlayerAllMatches;

if (matchMap.containsKey(winnerPlayer.getPlayerSlug())) {
  winnerPlayerAllMatches = matchMap.get(winnerPlayer.getPlayerSlug());
} else {
```

```
  winnerPlayerAllMatches = (List<Match>) matchService.findAllMatchesByPlayerName(
      winnerPlayer.getFirstName(), winnerPlayer.getLastName());
  matchMap.put(winnerPlayer.getPlayerSlug(), winnerPlayerAllMatches);
}
```

Listing 5.6. A győztes játékos meccseinek inicializálása

5.2.2. Python - Neurális hálók

A második fejezetben nagyon röviden már elkezdtük bemutatni a gépi tanulás és a neurális hálók világát, viszont lássuk kicsit részletesebben is, és hogy hogyan is használtuk fel mindezt az alkalmazásunkban.

Ahhoz, hogy jól megérthessük azt, hogy mi is az a neurális háló, tegyünk egy lépést hátra és beszéljünk egy kicsit a mesterséges neuronról, ismertebb nevén a perceptronról. A perceptront az 1950-es és 1960-as években Frank Rosenblatt tudós fejlesztette ki, inspirálódva Warren McCulloch és Walter Pitts munkájából. Manapság már gyakrabban használják a mesterséges neuron más modelljeit, például a sigmoid neuront, de hogy jobban megérthetssük annak működését, először lássuk hogyan is működik a perceptron.

A perceptron működése viszonylag egyszerű (2.3-as ábra), a több bemenő adatból (inputs: x1, x2, ...), egyetlen kimeneti értéket állít elő (output). A bemeneti értékek mellett súlyaink is vannak (w1, w2, ...), amelyek a bemeneti adatok fontosságát jelzik a kimenetre nézve. Minél nagyobb súllyal látunk el egy bemeneti értéket, annál fontosabb és nagyobb ráhatással lesz a kimenetünkre. A neuron kimenete 0 vagy 1, attól függően, hogy a bemeneti értékek és a nekik megfelelő súlyok szorzatösszege kisebb vagy nagyobb egy bizonyos küszöbértéknél (threshold value).

Történetesen amikor definiálunk egy perceptront, akkor annak a perceptronnak csak egyetlen kimenetele van. Egy háló esetén úgy tűnhet, hogy több is van, viszont még mindig igaz az az állítás, hogy csak egyetlen kimenet van. Viszont egy perceptron kimenete lehet több más perceptron bemenete egyaránt.

Annak érdekében, hogy megpróbáljuk leegyszerűsíteni a perceptron "szabályát", bevezethetjük a "bias" fogalmát (jelöljük b-vel), amelynek értéke feleljen meg a küszöbérték ellentettjével. Ennek következtében a képletünk a következőképpen alakul:

$$Output = \begin{cases} 0, w * x + b <= 0 \\ 1, w * x + b >= 0 \end{cases}$$

Erre a bias-re tulajdonképpen tekinthetünk úgy, mint egy olyan mértékre, amely megmondja, hogy milyen könnyen adhatja a perceptron számunkra az 1-es értéket. Mint ahogyan a képletből is láthatjuk, egy nagyon nagy bias értékkel nagyon egyszerűen tud a perceptronunk 1-es értéket visszatériteni. Természetesen a fordítottja igaz egy nagyon nagy negatív értékre.

Tételezzük fel, hogy az alkalmazásunkban perceptronokból felépített hálózatot használunk. Logikusan úgy gondoljuk, hogy egy teniszmérkőzés esetén, ha csak egy kis mértékben változtatjuk meg például az első adogatások sikerességének a súlyát és a bias értékét, akkor az egy kis eredménymódosulással járhat mindössze, például a két játékos győzelmi esélye változik valamelyik irányba 1-2 százalékkal. A probléma az, hogy egy kis változtatás ezeken az értékeken egy nagyon nagy bonyodalmat is okozhat olyan értelemben, hogy egy egyáltalán nem várt, irreális eredményt kapunk a végén. Ezeknek a fajta hibáknak a nagyobb eséllyel való elkerülése érdekében vezették be a sigmoid neuronokat.

A sigmoid neuronok hasonlítanak a perceptronokhoz, viszont különböznek egyetlen fontos tulajdonságban: általuk egy súlyon/bias-en végzett apróbb módosítás mindössze kisebb változással járhat a kimenetre nézve. Ezt úgy valósítja meg, hogy míg a perceptron esetében a kimeneti érték 0 vagy 1 lehet, addig a sigmoid neuron esetében a kimenet bármilyen érték lehet a [0,1] intervallumban, például 0.7496 is. A sigmoid függvény alakját már a második fejezetben is bemutattuk:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Szóval bizonyos műveletek elvégzése után a sigmoid neuron kimenete a következőképpen alakul (az x1, x2 ... bemeneti adatokat, a w1,w2, ... súlyokat és "b" a bias értéket jelöli):

$$\frac{1}{1 + exp(-\sum_{i=0}^{+n}(w_j * x_j - b))}$$

Alkalmazás

```
NR_OF_INPUTS = 90       // a bemenő adatok száma = INPUTS
NR_OF_LAY = 20          // az első layeren lévő neuronok száma
NR_OF_INNER_LAY = 8     // a második layeren lévő neuronok száma
NR_OF_OUTPUTS = 2       // a kimenő adatok száma = OUTPUTS
```

Listing 5.7. A neurális háló felépítése

```
def sigmoid(self, x):
    return 1.0 / (1.0 + numpy.exp(-x))

def sigmoid_prime(z):
    """A sigmoid függvény deriváltja"""
    return sigmoid(z)*(1-sigmoid(z))
```

Listing 5.8. A szigmoid függvény Pythonban

Mint azt ahogyan a fenti 5.7-es kódrészletben láthatjuk, az alkamazás 4 rétegből áll: egy 90 neuronból álló INPUT LAYER, egy 2 neuronból álló OUTPUT LAYER, és a köztük léve 20 és 8 neuronokból álló köztes layerek(HIDDEN LAYERS). Az alkalmazásban használt neurális háló felépítését a 2. fejezetben megjelenő 2.2-es ábrán tekinthetjük meg.

A már a fentiekben is tárgyalt, és a 5.8-as kódrészletben Python alakban is megtalálható szigmoid függvényt a "feedforward" metódus használja, amelyek a Network osztályunk metóduskészletéből származnak.

```
class Network(object):
    def __init__(self, sizes):
        self.layers = len(sizes)
        self.sizes = sizes
        self.biases = [numpy.random.randn(x, 1) for x in sizes[1:]]
        self.weights = [numpy.random.randn(y, x) for x, y in zip(sizes[:-1], sizes
            [1:])]
        self.data = []
        self.max_biases = []
        self.max_weights = []
        self.max_percentage = 0.0
        self.history = []

    def feedforward(self, a):
        """Return the output of the network if "a" is input."""
        for bias, weight in zip(self.biases, self.weights):
            a = self.sigmoid(numpy.dot(weight, a) + bias)
        return a

    ...
```

Listing 5.9. A Network osztály konstruktora illetve a feedforward függvény

Mint ahogyan a fenti 5.9-es kódrészletből is láthatjuk, a tömbben tárolandó adatok tárolásához a Python Numpy könyvtárát használjuk fel. Kezdetben a súlyokat és a bias értékeit random adatokkal töltsük fel, majd ahogyan a továbbiakban meglátjuk, ezeken az értékeken folyamatos apró módosításokat fogunk végrehajtani az eredményesség növelése érdekében.

A "feedforward" metódus egy, a paraméterként megadott (a) érték függvényében egy outputot fog visszatéríteni abban az esetben, ha a paraméterként megadott érték egy érvényes input adat.

És akkor most lássuk, hogyan is működik a tanulási/training folyamat:

```
def train(self, data, eta, weight_filename, biases_filename):
        j = 0
        self.history = []
        while j < NR_OF_EPOCH:
            j += 1
            self.update(data, eta)
            right = self.evaluate(data)
            whole = len(data)
            cur = right / whole
            print("Epoch", j, ":", cur*100, "%")
            self.history.append(cur*100)
            if cur > self.max_percentage:
                self.max_percentage = cur
                self.max_biases = self.biases
                self.max_weights = self.weights

        self.write_weights_to_file(self.max_weights, weight_filename)
        self.write_biases_to_file(self.max_biases, biases_filename)
        self.max_percentage = self.max_percentage*100
        print("Max percentage = ", self.max_percentage, "%")
```

Listing 5.10. A train függvény

A függvény (5.10-es kódrészlet) első fontos paramétere a "data" egy (inputs, outputs) párok listája, amelyek a tréning bemenő adatait és az eredményt (a győztest), azaz a kimeneti adatokat is tartalmazza. Ennek a szerkezete a következő:

```
{
  "outputs":[1,0],
  "inputs":[0.63,0.74,0.63,0.72,0.6,0.68,0.53,0.55,0.61,0.67,0.53,0.55,0.0,1.0 ...]
}
```

Listing 5.11. A data-paraméter egy eleme

Az "eta" a learning rate-nek felel meg, amelynek megválasztása nagyon is fontos, mert egy esetleges rossz érték választása nagyban eltorzíthatja, elronthatja a várt eredményt.

Az utolsó két paraméter (weight-filename, biases-filename) mint ahogyan a nevükből is kikövetkeztethető, az állományok nevét tartalmazzák, amelyekben a végső súlyokat és bias-eket fogjuk tárolni/kiírni. A tanulás folyamat egy előre meghatározott számú(NR OF EPOCH) alkalommal végrehajtott programblokkból áll, ugyanakkor annak érdekében, hogy a lehető legjobb beállításokat tartsuk meg, minden iteráció végén összehasonlítjuk az eredményeket, és a jobbat lementjük.

Minden iteréció alkalmával frissítjük (5.13) a súlyokat (weights) és a torzításokat (bias). Ha a training adatok rendelkezésre állnak, akkor a program minden iteráció alkalmával kiértékeli a hálózatot, amelyet az "evaluate" függvénnyel teszi (5.12-es kódrészlet). A kimeneti adatok alapján tudja majd ez a függvény eldönteni, hogy az álta megtippelt eredmény helyes volt vagy sem, amelynek függvényében más és más módosításokat eszközöl a beállításokon(weights, biases). Mindezek mellet a függvény minden körben kiírja a haladást (a háló pontosságát, hogy az összes mérkőzés/traning adat közül hányat tippelt meg helyesen, azaz hogy ezt milyen arányban tette: cur = right / whole). Ez természetesen bizonyos mértékben lassítja a program futását, növeli annak futási idejét, viszont nagyon hasznos és sokatmondó is egyben.

```
1 def evaluate(self, data):
      test_results = [(numpy.argmax(self.feedforward(x)), y) for (x, y) in data]
      return sum(int(x == numpy.argmax(y)) for (x, y) in test_results)
```

Listing 5.12. A neurális hálót kiértékelő függvény

```
  def update(self, data, eta):
2   dif_bias = [numpy.zeros(b.shape) for b in self.biases]
    dif_weight = [numpy.zeros(w.shape) for w in self.weights]
    for x, y in data:
      delta_dif_bias, delta_dif_weight = self.backpropagation(x, y)
      dif_bias = [newbias + difnewbias for newbias, difnewbias in zip(dif_bias,
          delta_dif_bias)]
7     dif_weight = [newweight + difnewweight for newweight, difnewweight in zip(
          dif_weight, delta_dif_weight)]
    self.weights = [weight - (eta / len(data)) * newweight for weight, newweight in
        zip(self.weights, dif_weight)]
    self.biases = [bias - (eta / len(data)) * newbias for bias, newbias in zip(self.
        biases, dif_bias)]
```

Listing 5.13. A neurális hálót kiértékelő függvény

Mint ahogyan már a fentiekben is említettük, az 5.13-as kódrészletben szereplő update függvény a felelős a súlyok és a torzítások frissítésében. Ennek a metódusnak a legfontosabb sora a következő:

```
delta_dif_bias, delta_dif_weight = self.backpropagation(x, y)
```

Itt hívja meg az úgynevezett backpropagation algoritmust (5.14), amelynek működésének a megértése egy elég komoly matematikai tudást feltételez és igényel, viszont ebben a dolgozatban nem megyünk ennek az algoritmusnak a működésébe. Tulajdonképpen ez az algoritmus egy gyors módszer a költségfüggvény gradiensének a kiszámításához, amely függvényében az update nevű függvényünk a megfelelő módon frissíti a súlyok és a torzítások értékeit.

```
    def backpropagation(self, x, y):
        dif_bias = [numpy.zeros(bias.shape) for bias in self.biases]
        dif_weight = [numpy.zeros(weight.shape) for weight in self.weights]
        activation = x
        activations = [x]
        results = []
        for bias, weight in zip(self.biases, self.weights):
            result = numpy.dot(weight, activation) + bias
            results.append(result)
            activation = self.sigmoid(result)
            activations.append(activation)
        delta = self.cost_derivative(activations[-1], y) * self.sigmoid_prime(
            results[-1])
        dif_bias[-1] = delta
        dif_weight[-1] = numpy.dot(delta, activations[-2].transpose())
        for l in range(2, self.layers):
            result = results[-l]
            sp = self.sigmoid_prime(result)
            delta = numpy.dot(self.weights[-l + 1].transpose(), delta) * sp
            dif_bias[-l] = delta
            dif_weight[-l] = numpy.dot(delta, activations[-l - 1].transpose())
        return (dif_bias, dif_weight)
```

Listing 5.14. A backpropagation algoritmus

A training végeztével a kiszámolt és a folyamatosan javított súlyokat és torzításokat egy-egy állományba mentjük, amelyeket majd újra beolvasunk és Numpy tömbök formájában újra felépítünk. Ezeket az tömböket (self.weights, self.biases) fogjuk használni egy egy párharc megjósolása érdekében:

```
@app.route('/prediction', methods=['POST'])
def predict():
    network = Network([NR_OF_INPUTS, NR_OF_LAY, NR_OF_OUTPUTS])
    network.set_weights_and_biases(add_relative_path(request.json['weights_filename'
        ]), add_relative_path(request.json['biases_filename']), False)

    numpy_inputs = convert_data_to_input_data(request.json['inputs'])
    resp_data = network.feedforward(numpy_inputs)

    numpy_inputs_revert = convert_data_to_input_data(revert_input_data(request.json[
        'inputs']))
    resp_data_invert = network.feedforward(numpy_inputs_revert)

    percentage1 = ((resp_data[0][0] + resp_data_invert[1][0]) / 2) * 100
    percentage2 = ((resp_data[1][0] + resp_data_invert[0][0]) / 2) * 100
```

```
    to_json = {}
    to_json['first_percentage']  = percentage1
    to_json['second_percentage'] = percentage2

    return app.response_class(
        response=json.dumps(to_json),
        status=200,
        mimetype='application/json'
    )
```

Listing 5.15. A Predict Controller a Python szerver esetén:

A fenti Python Controller által kiszámított valószínűségeket egy JSON objektummá alakítjuk, majd visszaküldjük a Java alapú szerverünknek. Az majd megpróbálja értelmezni a kapott választ, majd továbbítja azt a kliens oldal irányába (5.16-os kódrészlet).

```
PredictionRequestDTO requestDTO =
  new PredictionRequestDTO(playerOne.getPlayerSlug(),
      playerTwo.getPlayerSlug(),
      weight_filename,
      biases_filename,
      getInputsToPredict(playerOne, playerTwo, surface, tourneyName));

HttpEntity<PredictionRequestDTO> request = new HttpEntity<>(requestDTO);
ResponseEntity<PredictionResponseDTO> response;

try{
    response = restTemplate.postForEntity(URL, request, PredictionResponseDTO.class)
        ;
    PredictionResponseDTO responseDTO = response.getBody();
        return predicterService.convertSumToOneHundredPercent(
          responseDTO.getFirst_percentage(),responseDTO.getSecond_percentage());
} catch (Exception e){
    System.out.println(e);
    return Arrays.asList(50, 50);
}
```

Listing 5.16. Szerverek közti kommunikáció

6. fejezet

Összegzés

A projekt megvalósításának az érdekében szükség volt elsajátítani a React programozás és a JavaScript alapjait, amelyek által az alkalmazás kliens oldali részét valósítottuk meg. Ez mellett szükségünk volt a Spring keretrendszer bizonyos szolgáltatásainak a megértésre, illetve bizonyos Hibernate elemek elsajátítására is, amely segítségével le tudtuk képezni egymásba a relációs adatbázisok tábláit és a különböző osztályokat.

Mindezek mellett szükség volt a Python programozási nyelv alapjainak az elsajátítására, valamint a neurális hálók alapjainak és működésének a megértésére egyaránt.

Véleményem szerint ez a projekt több szempontból is hasznos volt. Elsősorban egy viszonylag mélyebb tudásra tettünk szert a neurális hálók működését illetően, másrészt meg sikerült egy olyan alkamazást létrehoznunk, amely nagyon jó megközelítéssel jósolja meg a férfi teniszmérkőzések végkimeneteleit. Ez a pontosság már fogadás szempontjából és iránymutató lehet, mert rengeteg olyan statisztikai adatot vizsgál meg és vesz figyelembe, amelyeket egy, a témában jártas fogadó sem gondol át, jegyez meg, és amelyek döntőnek bizonyúlhatnak egy-egy mérkőzés esetében.

6.1. Továbbfejlesztési lehetőségek

Az alkalmazás további fejlesztésre alkalmas, és több olyan potenciális fejlesztési lehetőség létezik, amely még használhatóbbá, pontosabbá illetve átláthatóbbá teheti alkalmazásunkat.

Ezen fejlesztési lehetőségek közül az egyik legfontosabb és legnagyobb hatással bíró újítás a szorzók bevezetése lenne. Ez úgy segíthetne fejleszteni az alkalmazásunkat, hogy általa profitot számolhatnánk az alkalmazás teszelése alkalmával, amely nagyban hozzájárulna abban, hogy egy reális képet kaphassunk az alkalmazás megbizhatóságát illetően.

Egy másik fejlesztési lehetőség a 2018 és 2019-es adatok hozzáadása az adatbázisunkhoz, amely jelen esetben mindössze az 1991 és 2017-es évek mérkőzéseit tartalmazza.

Az alkalmazás pontosságának a növekedését segítheti elő az új statisztikák bevezetése, mint például a kikényszerített és a ki nem kényszerített hibák figyelembe vétele, amelyek rávilágíthatnak a játékosok formájára, jelenlegi mentális állapotára és koncentrációjára egyaránt.

Irodalomjegyzék

1. A React hivatalos oldala: https://reactjs.org/.

2. A React Context hivatalos oldala: https://reactjs.org/docs/context.html.

3. A React Hooks hivatalos oldala: https://reactjs.org/docs/hooks-intro.html.

4. A Spring hivatalos oldala: https://spring.io/.

5. A neurál hálókkal kapcsolatos elmélet és algoritmusok:: http://neuralnetworksanddeeplearning.com/.

6. A Jet Brains hivatalos oldala: https://www.jetbrains.com/

7. A Visual Studio Code hivatalos oldala: https://code.visualstudio.com/

8. A Bootstrap hivatalos oldala: https://getbootstrap.com/