Szegedi Tudományegyetem
Informatikai Intézet

SQL injection detektálás természetes nyelvi feldolgozó rendszerek és gépi tanulás által

SQL injection detection using natural language processing systems and machine learning

Szakdolgozat

Készítette:
Stiller Marianna
programtervező
informatika szakos
hallgató

Témavezető:
Dr. Vidács László
tudományos
főmunkatárs

Szeged
2021

Feladatkiírás

Az SQL Injection napjaink leggyakoribb és legveszélyesebb sérülékenysége, amely a felhasználói input elégtelen mértékű validációjából ered, következménye pedig SQL lekérdezések futtatása érzékeny adatokon.

A rendszer természetes nyelvi feldolgozó rendszerek és gépi tanulás által képes lesz forrásódban SQL Injection sérülékenységeket keresni. Működésének alapja, hogy valós PHP alapú forráskódokból reprezentál absztrakt szintaxis fákat, melyekből mintákat nyer ki, amit később megtanul.

Tartalmi összefoglaló

A téma megnevezése:

SQL injection detektálás természetes nyelvi feldolgozó rendszerek és gépi tanulás által.

A megadott feladat megfogalmazása:

Gépi tanuló algoritmusok és természetes nyelvi feldolgozó rendszerek alkalmazásával tanított program képes detektálni SQL injection típusú sérülékenységet.

A megoldási mód:

Tanuló példákból generált absztrakt szintaxis fákat a program tokenizálja, konkatenálja és felcímkézi, mint érzékeny vagy biztonságos kód. Ezek átesnek egy vektorizáláson, majd egy kifejezett osztályozó segítségével – amely a legkevesebb hibás döntéseket hozza – a program megtanulja felismerni a sérülékenység jeleit. Ennek hatására egy nem tanuló kódról képes kiértékelést készíteni.

Alkalmazott eszközök, módszerek:

Visual Studio Code és IntelliJ fejlesztői környezetet, emellett a gépi tanulás során Python, az asztali alkalmazás megvalósításához back-end-en Java programozási nyelvet, front-end-en FXML leíró nyelvet alkalmaztam. Verziókövetésre a GitHub rendszert használtam.

Elért eredmények:

Az SQL Injection detektálás problémát gépi tanulási algoritmusok és természetes nyelvi feldolgozó rendszerek alkalmazásával közelítettem meg. Osztályozási módszerrel osztályoztam a bejövő kódot sebezhető vagy biztonságos kódként, előzetes AST generálás, tokenizálás és vektorizálás segítségével. Öt gépi osztályozási algoritmust teszteltem 102 tanuló adattal, amelyek a Logistic Regression, K-Nearest Neighbors, Linear Suppoet Vector, Gaussian Naive Bayes és Random Forest osztályozó. A program felhasználói felületén bemenetként megadott kódot a legpontosabbal osztályoztam, azaz a PHP kódra kimenetként meghatározza azt, hogy a kód milyen valószínűséggel sérülékeny.

Kulcsszavak:

Felügyelt tanulás, train/test készlet, K-Fold, confusion matrix, recall.

Tartalomjegyzék

BEVEZETÉS

A mai világban, az informatika világában, a digitalizálás visszafordíthatatlan, illetve nem okszerű. Az informatika meglehetősen gyors fejlődése miatt, egy generáció már születésétől kezdve természetesnek vélheti az internethasználatot. A társadalom kiélvezi az összes kényelmi funkcióját, lebegnek a felszínen, a boldog tudatlanságban, az adataik pedig másodpercről másodpercre szivárognak le, a felszín alá. Az információk megtartása és biztonsága ezáltal nyert a tudományágában külön területet. Mivel értékesebbek lettek, mint a pénz maga.

Információbiztonságra szakosodó programtervező informatikusként állítom, hogy a szoftverfejlesztés nem minden. A bemeneti mezőket nem elég implementálni, meg is kell védeni őket. Ehhez pedig már több módszer van, annál is több példával. A programom mesterséges intelligencia segítségével ezeknek egy elenyésző részét használja fel a tanuláshoz és annak hasznosításához lehetőséget adva ezzel a fejlesztőknek arra, hogy biztonságos programkódot készítsenek.

A témaválasztásom egyszerű célja, hogy egy olyan kezdetleges rendszert készítsek, amely lehetőséget ad megvédi azt, ami a pénznél is drágább. Bár az injektálás csak egy része a biztonsági kockázatoknak, ám a legfenyegetőbb [1]. Az SQL injection és általában a webalapú támadások a kiberbiztonsági kutatók egyik legfőbb gondját definiálják. Fontos kérdéseket jelentenek a pénzügyi, egészségügyi és egyéb kritikus adatok biztonságában. Ennek a problémának a jelentősége pedig csak növekszik, mivel a társadalmi folyamatok egyre inkább függenek az internettől, ahogy korábban említettem.

A dolgozatban ki fogom fejteni többek között ezt a fajta sérülékenységet az első fejezetben, a gépi tanulást a másodikban és a kettő ötvözetét a harmadikban, amely a program tulajdonképpeni felépítése. Ehhez a fejezethez kötődik a címben megnevezett másik detektálási eszköz, a természetes nyelvi feldolgozó rendszerek és azok technikái. Legvégül pedig kifejtem a legutolsó fejezet keretei között, a program által elért eredményeket. A program, kitűzött célként ki fogja tudni mutatni a biztonságtechnikai fenyegetettséget az SQL injection irányából.

1. SQL INJECTION

A témamegnevezés során meglehetősen kevés időt vett igénybe a konkrét biztonsági kockázat kiválasztása. Szerettem volna egy olyan sérülékenység köré építeni a programom megvalósítását, amelyhez sok példa és ellenpélda tartozik, valamint a dokumentáltsága jobb, mint kiváló. A választásom ez okból esett az első számú kockázat egy fajtájára, az SQL injection-re [2].

AZ OWASP (The Open Web Application Security Project) listáján, amely a legkritikusabb biztonsági kockázatokat taglalja, az injection (magyarul: befecskendezés, injekció) áll az első helyen. Ez a támadási típus a felhasználói input elégtelen mértékű validációjából ered, amely lehetővé tesz rosszindulatú SQL utasítások végrehajtását, amelyek közvetlenül az adatbázisból kérhetnek le adatokat, emellett adminisztrációs műveleteket hajthatnak végre az adatbázisban, a DBMS fájlban található adott fájl tartalmát helyreállíthatja a rendszerben, és egyes esetekben parancsokat adhat ki az operációs rendszernek. Például az alábbi sor *' UNION SELECT username, password FROM users*-- [3] megadása bemeneti paraméterként az alábbi SQL lekérdezést futtathatja amennyiben a bemenet sérülékeny, létezik *users* tábla, valamint *username* és *password* oszlop: *SELECT name, description FROM products WHERE category = 'Gifts' UNION SELECT username, password FROM users*-- [3]. A lekérdezés eredménye az adatbázis összes felhasználójának neve és jelszava.

Ebből adódóan kijelenthetjük, hogy az injection napjaink egyik leggyakoribb és legveszélyesebb sérülékenysége.

1.1. Típusai

1.1.1. Sávon belüli (in-band, classic)

Ahogy a korábbi példa szemléltette, a sikeres SQL injection támadás képes érzékeny adatok kiolvasására az adatbázisból. Egyik legfőbb eszköze az *UNION* operátor, amely lehetővé tesz egy vagy több további *SELECT* lekérdezés végrehajtását és az eredmények hozzáfűzését az eredeti lekérdezéshez. Sikerességének feltétele, hogy a két vagy több *SELECT* oszlopszáma és oszloptípusai megegyezzenek vagy konvertálhatóak legyenek páronként.

Adatok kiolvasásán, lekérdezésén kívül képes továbbá adatbázis béli adatok törlésére, módosítására, beszúrására és frissítésére, ha az alábbi sémát vesszük alapul:

105; DROP TABLE Suppliers [4]. Így a tényleges lekérdezés *SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suspplier*s [4]*;* lesz. Ez a támadási forma a pontosvessző SQL béli funkcióját használja ki halmozott lekérdezések indításához. Szükséges megjegyezni, hogy a DBMS (Database Management System) rendszerek csak privilegizált felhasználók számára szokták engedélyezni a *DROP* parancsot és azt is korlátozott forrásból. Ebből adódóan a DBMS rossz konfigurálása is feltétele ennek a támadási típusnak.

További lehetőség, amikor a támadó SQL lekérdezéseken keresztül az adatbázis-kiszolgáló által dobott hibaüzenetekre támaszkodva gyűjt információkat (error-based), ezzel feltérképezve az adatbázis szerkezetét. Például, ha a *https://www.example.beaglesecurity.com/gallery.php?id=6'* [5] URL megadása esetén a szerver a hibát SQL szintaxisban adja vissza, akkor többlépcsős lekérdezésen keresztül az adatok szintén kinyerhetőek.

Ezek a típusok tehát azért „sávon belüliek”, mert a támadó ugyanazt a kommunikációs csatornát képes használni a támadás elindítására és az eredmények gyűjtésére. Továbbá „klasszikus”, mert a legkedveltebb támadási lehetőséget foglalja magába, eredményeket pedig gyorsan lehet kinyerni általa, ellentétben a következő fajtával.

1.1.2. Vak (blind, inferential)

Az következtetéses befecskendezés esetén – ellentétben a sávon belüli injektálással – hosszabb ideig tarthat, amíg a támadók kihasználják a hibát, ugyanakkor ugyanolyan veszélyes, mint az SQL Injection bármely más formája. Ezen fajta esetén a webalkalmazáson keresztül nem kerülnek át tényleges adatok, és a támadó sem láthatja a sávon belüli támadás eredményét. Ehelyett a támadó hasznos terhelések küldésével képes rekonstruálni az adatbázis-struktúrát, figyelemmel kísérve a webalkalmazás válaszát és az adatbázis-kiszolgáló ebből fakadó viselkedését.

Mint altípusa, a logikai alapú (boolean-based) injekció egy következtetéses SQL Injection technika, amely SQL lekérdezéssel arra kényszeríti az alkalmazást, hogy a lekérdezéstől függően más eredményt adjon vissza. Az alábbi URL *https://www.example.beaglesecurity.com/gallery.php?id=1' AND 1=0 --+* [6] egy lekérdezést generál: *SELECT title, description, doby FROM items WHERE id=1’ AND 1=0* [6]. Ha az alkalmazás sebezhető SQL Injekcióval szemben, akkor nem ad vissza semmit, mivel ez egy hamis állítás, 1 nem egyenlő 0-val. Ezek után, ha egy igaz állítást

adunk meg: *https://www.example.beaglesecurity.com/gallery.php?id=1' AND 1=1 --+* [6] és az oldal tartalma megváltozik az előző hamis állapothoz képest, akkor a támadó arra következtethet, hogy az injekció működik. A logikai eredménytől (IGAZ vagy HAMIS) függően tehát a HTTP válasz tartalma megváltozik, vagy ugyanaz marad. Ezt használja ki a vak injection.

Az időalapú (time-based) injekció esetén a támadó egy SQL-parancsot küld a kiszolgálónak, hogy késleltesse a lekérdezések végrehajtását. Erre például MySQL-ben a legkézenfekvőbb megoldást a *SLEEP* függvény adhatja: *SELECT * FROM card WHERE id=1-SLEEP(15)* [7]. A másodpercben megadott szám késlelteti a lekérdezés eredményét ideális esetben. Így tehát a válaszidő jelzi a támadó számára, hogy az injektálás lehetséges és a szerver a MySQL-t használja adatbázisként. Következő lépésként például kinyerhető az adatbázis verziója: *SELECT * FROM card WHERE id=1-IF(MID(VERSION(),1,1)='5', SLEEP(15),0)* [7], amely esetén ha a kiszolgáló válasza legalább 15 másodpercet vesz igénybe, megállapíthatjuk, hogy ez az adatbázis-kiszolgáló a MySQL 5.x verzióját futtatja.

1.1.3. Sávon kívüli (out-of-band)

Sávon kívüli injekció akkor fordul elő, amikor a támadó nem tudja ugyan azt a csatornát használni a támadás elindításához és az eredmények összegyűjtéséhez. A legritkább használt típus a három közül, mivel a webalkalmazás által használt adatbázis-kiszolgálón engedélyezett funkcióktól függ.

Ez a technika az adatbázis-szerver azon képességére támaszkodik, hogy adatok továbbításához DNS vagy HTTP kéréseket használ. A típus sikeres használatakor a felhasználó DNS kérelmét a támadó egy olyan szerverre küldi át például a Microsoft SQL Server *xp_dirtree* parancsával, amelyet ő irányít. Ilyen támadásra alkalmas URL például:*https://example.com/products.aspx?id=1;EXEC%20master..xp_dirtree%20'%5c%5ctest.attacker.com%5c'+--+* [7]. Dekódoláskor a *%20* és a *%5c* a szóköznek és fordított per jelnek felelnek meg, így az URL ténylegesen így néz ki: *https://example.com/products.aspx?id=1;EXEC master..xp_dirtree '\\test.attacker.com\'* – [8], amely az alábbi lekérdezést eredményezi: *SELECT * FROM product WHERE id=1;EXEC master..xp_dirtree '\\test.attacker.com\'* -- [8]. Ezzel a támadó a *test.master.com* webhelyre kényszeríti a DNS kérést, a támadás sikeres.

1.2. Megelőzési módjai

Az SQL injekciós sebezhetőség elkerülése meglepően egyszerű, szembekerülve azzal a ténnyel, milyen sok hasonló fajta támadás fordul elő. A biztonsági rések megelőzésére is több lehetőség van.

Elkészített utasítások (prepared statements) használata

Az elkészített utasítás egy paraméterezett és újrafelhasználható SQL lekérdezés, amely arra kényszeríti a fejlesztőt, hogy külön írja be az SQL parancsot és a felhasználó által megadott adatokat. Ez a kódolási stílus lehetővé teszi az adatbázis számára, hogy különbséget tegyen a kód és az adatok között.

Gyakorlatias példával élve, ha vesszük a boolean-based altípus példáját, akkor a paraméterezett lekérdezést használva az, egy felhasználói azonosítót keresne, amely megegyezik a támadó által megadott karakterlánccal, tehát nem lenne sérülékeny.

Language	Example
C#	`String stmt = &quot;SELECT * FROM table WHERE data = ? &quot;; OleDbCommand command = new OleDbCommand(stmt, connection); command.Parameters.Add(new OleDbParameter(&quot;data&quot;, Data d.Text)); OleDbDataReader reader = command.ExecuteReader();`
Java java.sql	`PreparedStatement stmt = con.prepareStatement (&quot;SELECT * FROM table WHERE data = ?&quot;); stmt.setString(1, data);`
PHP PDO class using named parameters	`$stmt = $db-&gt;prepare(&quot;SELECT * FROM table WHERE data = :data&quot;); $stmt-&gt;bindParam(':data', $data); $stmt-&gt;execute();`
PHP PDO class using ordinal parameters	`$stmt = $db-&gt;prepare(&quot;SELECT * FROM table WHERE data = ?&quot;); $stmt-&gt;bindParam(1, $data); $stmt-&lt;execute();`
PHP PDO class using array	`$stmt = $db-&gt;prepare(&quot;SELECT * FROM table WHERE data = :data&quot;); $stmt-&gt;execute(array(':data' =&gt; $data)); $stmt = $db-&gt;prepare(&quot;SELECT * FROM table WHERE data = ?&quot;); $stmt-&gt;execute(array($data));`
PHP mysqli	`$stmt = $mysqli-&gt;prepare(&quot;SELECT * FROM table WHERE data = ?&quot;); $stmt-&gt;bindParam('s', $data);`
Python django.db	`from django.db import connection, transaction cursor = connection.cursor() cursor.execute(&quot;SELECT * FROM table WHERE data = %s&quot;, [data])`

1.2.1-es ábra: példák elkészített utasításokra [9]

Tárolt eljárások (stored procedures) használata

A tárolt eljárások biztonságosan megvalósítva (azaz a tárolt eljárás nem tartalmaz semmilyen nem biztonságos dinamikus SQL-t) ugyanolyan hatást gyakorolnak, mint a

paraméterezett lekérdezések. Különbségük csupán csak annyi, hogy a tárolt eljárás SQL-kódját meghatározzák és magában az adatbázisban tárolják, majd az alkalmazásból meghívják (alábbiakban az *sp_getAccountBalance*).

```
// This should REALLY be validated
String custname = request.getParameter("customerName");
try {
  CallableStatement cs = connection.prepareCall("{call sp_getAccountBalance(?)}");
  cs.setString(1, custname);
  ResultSet results = cs.executeQuery();
  // … result set handling
} catch (SQLException se) {
  // … logging and error handling
}
```

1.2.2-es ábra: Java példa tárolt eljárásra [10]

Engedélyezőlista (allow-list) bemenetének ellenőrzése

Az SQL lekérdezések különböző részei néha nem alkalmasak olyan változók használatára, amelyeket csak érvényes utasítással vagy értékcímmel lehet helyettesíteni, hogy az utasítás sikeresen végrehajtható legyen (bind variables). Ilyen helyzetekben a bemenet ellenőrzése vagy a lekérdezés újratervezése a legmegfelelőbb védekezés. Az adatbázis táblázatainak vagy oszlopainak neve esetén például ideális esetben az értékek a kódból származnak, és nem a felhasználói paraméterekből.

Az alábbi példa egy cikluson belül különböző ágakban vizsgálja a *tableName* értékének lehetőségeit, ezzel lekorlátozva az elérhető és lekérhető táblák számát.

```
String tableName;
switch(PARAM):
  case "Value1":
    tableName = "fooTable";
    break;
  case "Value2":
    tableName = "barTable";
    break;
  ...
  default:
    throw new InputValidationException("unexpected value provided for table name");
```

1.2.3-as ábra: Java példa adatbázis tábla neveinek ellenőrzésére [10]

Az összes felhasználó által adott bemenet elkerülése

Ennek a technikának az egyszerű célja, hogy elkerülje a felhasználói bevitelt, mielőtt lekérdezéshez lehetne használni. Alkalmazása a fenti módszerek megvalósíthatatlansága esetén kézenfekvő lehet, ám nem garancia az SQL injection megakadályozására, ún. végső lehetőséget biztosít.

2. GÉPI TANULÁS

A gépi tanulás a mesterséges intelligencia olyan alkalmazása, amely lehetőséget nyújt a rendszereknek arra, hogy kifejezetten programozás nélkül automatikusan tanuljanak és fejlődjenek tapasztalatok segítségével. Forráskód biztonsági kontextusban a gépi tanulást használják többek között arra is, hogy megvalósítsák a forráskód SQL injection-re sérülékeny kód vagy nem sérülékeny kódként történő besorolásának legpontosabb és leghatékonyabb módját a digitális eszközök legjobb védelme érdekében [11].

A gépi tanulási módszereknek három fő kategóriája van: felügyelt (supervised), felügyelet nélküli (unsupervised) és félig felügyelt (reinforcement) tanulás. Az implementált programom a legelső módszert alkalmazza, így a fejezetben ehhez adok szélesebb körű ismeretet.

2.1. Felügyelt tanulás

Felügyelt tanulás során címkézett adatkészletek felhasználásával algoritmusokat képeznek, amelyek pontosan osztályozzák az adatokat vagy előre jelzik az eredményeket. Ez a típusú tanulás segít a szervezeteknek számos valós problémát megoldani, például a fent említett forráskódok osztályozása által.

Ahhoz, hogy egy algoritmus meg tudjon határozni egy eredményosztályt gépi tanulás által, ahhoz végig kell mennie annak felépítésének és fejlesztésének lépcsőfokain. A felügyelt gépi tanulás felépítésének hét alapvető lépése van:

1. Adatok gyűjtése és 2. előkészítése

   A releváns példák gyűjtése után, az előkészítéskor a példák két részhalmazba kerülnek felügyelt tanulás során: a train és a test halmazba. A címkézett adatok (train), amely ideálisan az összes adat ~80%-a, az algoritmus tanítására szolgálnak. Forráskód biztonsági kontextusban a két címke – a fenti példából adódóan – a sérülékeny és nem sérülékeny kód. A nem címkézett adatokat (test), amely az összes adat maradék ~20%-a, az algoritmus önállóan címkézi fel, ezzel tesztelve, hogy a megállapítása jó-e vagy rossz. A halmazokra bontás egyszóval azért fontos, mert ugyanazok az adatsorok mind a tanításhoz, mind a teszteléshez nem adnának igazságos értékelést a modell teljesítményéről, ezáltal rontanák a modell hatékonyságát.

1. Algoritmus kiválasztása

   Az algoritmus típusa többek között függ a képzési adatkészlet típusától, az adatok mennyiségétől, a megoldandó probléma típusától és így tovább. A 2.1.1-es ábra segítségével szeretném szemlélteti, hogy különböző eloszlású adatok esetén milyen fontos a megfelelő osztályozóválasztás, mivel az osztályozás pontosságát határozza meg. A 2.3 alfejezeten kerül kifejtésre az osztályozási algoritmusok teljesítményének mérése, amely lehetőséget ad azok összehasonlítására és teljesítményorientált szelektálására.

2. Algoritmus betanítása

   Ez a folyamat legfontosabb része. A konkrét „tanulás” nagy része ebben a szakaszban történik. Itt felhasználjuk az adatkészlet tanításához elkülönített részét, hogy megtanítsuk algoritmusunk a két állapot (normál adatforgalom, veszélyes adatforgalom) megkülönböztetésére. Az így kapott betanított, pontos algoritmus a gépi tanulási modell.

3. Modell értékelése

   Ebben a lépésben az adatkészlet teszteléséhez elkülönített részét használjuk fel. Ahogy korábban említettem, ekkor az algoritmus önállóan címkézi fel az adatokat, ezzel tesztelve, hogy a megállapítása jó-e vagy rossz, ezzel értékelést adva magáról a modellről.

4. Modell fejlesztése

   A hatodik lépés megpróbálja javítani az értékelési lépés során elért eredményeket. Ha az eredmények szinten aluli, akkor az eddigi lépések ellenőrzésére, javítására vagy módosítására kerül sor addig, amíg a tesztelés kimenete nem lesz elégséges vagy megfelelő.

5. Modell használata

   Az utolsó lépés a modell új adatokkal történő felhasználása. A modell ekkor függetlenséget nyer az emberi beavatkozásaitól és saját adatállománya, valamint képzése alapján vonja le saját következtetéseit. Szoftverfejlesztési megközelítésből ez az a lépés az, amit a végfelhasználó lát. Ez a lépés rávilágít arra, miért tartják sokan a gépi tanulást a különböző iparágak jövőjének.

Az emberek csak bizonyos mennyiségű adatot és releváns tényezőt tarthatnak szem előtt a döntés meghozatalakor. Ezzel szemben a gépi tanulási modellek nagy

mennyiségű adatot képesek feldolgozni és összekapcsolni, ezzel betekintést nyerve abba, amit normális megközelítéssel látni nem lehet.

Több kategória alakult ki gépi tanuláson belül a felügyelt gépi tanulás körében: az osztályozás (classification) és a regresszió (regression). Míg az osztályozás célja, hogy megjósolja azt a kategóriát, amelyhez az adatok tartoznak, addig a regresszió számértéket jósol a korábban megfigyelt adatok alapján. Ez egy példán keresztül könnyedén megérthető: az osztályozás a „Holnap meleg lesz vagy hideg?” kérdésre válaszol, a regresszió pedig a „Milyen idő lesz holnap?”. A program kérdésköre az osztályozáshoz tartozik, így annak algoritmusait veszem sorra.

2.2. Osztályozás

Az osztályozás tanuló adatok alapján – továbbiakban független változók (independent variables) – annak meghatározása, hogy melyik osztályba tartozik a bemenetként megadott nem tanuló adat – továbbiakban függő változó (dependent variable).

2.2.1. Logistic Regression

Az osztályozási algoritmusok célja, hogy a függő változót egy osztályba sorolják független változók halmazát felhasználva. A Logistic Regression ennek egy speciális esete, mivel csak bináris kimenet előrejelzésére alkalmas, azaz az osztály adott értelmezéssel ellátva (sérülékeny, nem sérülékeny) 0 vagy 1 lehet. Alapja a Linear Regression, mivel az első lépésében ezt regressziós képletet hajtja végre, ahol *y* jelöli a függő változót, *x* a független változót, a b_0 és b_1 pedig a konstansokat.

$$y = b_0 + b_1 x$$

2.2.1.1-es-képlet: a lineáris regresszió képlete

A második lépés, a 2.2.1.1-es ábra *y* értékét használja fel a logisztikai függvény felépítéséhez.

$$p = \frac{1}{1 + e^{-y}}$$

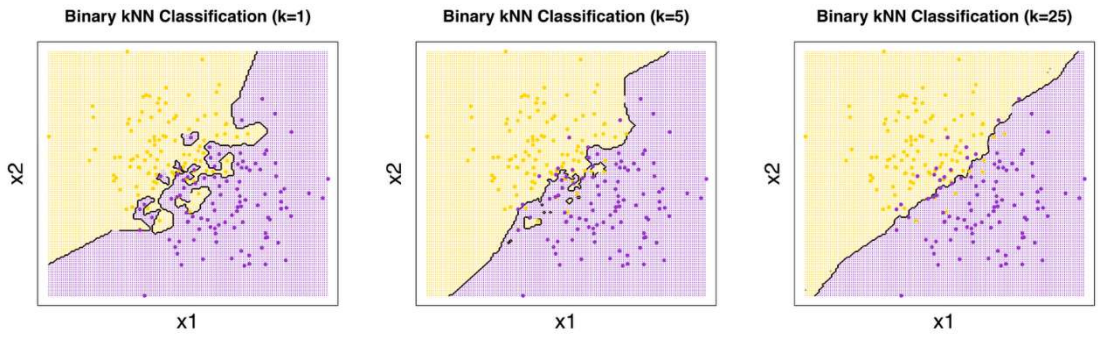
2.2.1.2-es képlet: a logisztikai szigmoid függvény képlete

Ekkor a függvény már egy valószínűséget köt a függő változóhoz, amelyet a 2.2.1.3-as ábra egy-egy fekete ponttal jelöl a logisztikai függvényen. Ez megadja, hogy

mekkora valószínűséggel tartozik az egyes osztályokhoz. Amennyiben a 0,5-ös küszöbérték alatt esik, a 0 osztályba sorolható, ellenkező esetben az 1-es osztályba.

2.2.2. K-Nearest Neighbors (K-NN)

A K-NN algoritmus az egyik legegyszerűbb osztályozási algoritmus, mivel az új eseteket azok K legközelebbi szomszédaik osztályai alapján osztályozza. Példának okáért, ha a 3 legközelebbi szomszéd közül 2 db (67%) tartozik az 1-es osztályba és csak 1 db (33%) a 0-ás osztályba, akkor az algoritmus az 1-eshez rendeli az új esetet. Példán belül, ha a 9 legközelebbi szomszéd közül 3 db (33%) tartozik az 1-es osztályba és 6 db (67%) a 0-ás osztályba, akkor az algoritmus a 0-áshoz rendeli az új esetet. Így a K értékének megválasztása már jóval fontosabbá válik, ezt szemlélteti három példán keresztül a 2.2.2.1-es ábra.



2.2.2.1-es ábra: a K-NN algoritmus pontossága a K érték függvényében [12]

A K szám meghatározásakor tehát két dolgot fontos figyelembe venni: ha az értéke túl kicsi, az bár pontosítja az algoritmust, de a kiugró értékei miatt zajossá teheti azt. Ellenben, ha túl nagy, az az algoritmus pontatlanságát éri el, ez a túltanulás jelensége. Ekkor az algoritmus a tanuló a példákra pontos értéket ér el, de új bemenetekre pontatlan.

A K-NN megvalósításakor az adatpontok jellemző vektorokká alakítása után az algoritmus megtalálja a vektorok közötti távolságot. A távolság megtalálásának leggyakoribb módja az euklidészi távolság kiszámítása.

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

2.2.2.2-es képlet: az n-dimenziós euklidészi távolság képlete

2.2.3. Linear Support Vector (LinearSVC)

Az Support Vector Machine (SVM) a döntési határokat meghatározó döntési síkok koncepcióján alapul. A LinearSVC gépi algoritmus célja egy lineáris hipersík (hyperplane) megtalálása, amely – ahogy a 2.2.3.1-es ábrán látható – egyértelműen elválasztja a különböző osztályok tagjainak halmazát. Ehhez sokféle hipersík választható. A cél egy olyan sík megtalálása, amely rendelkezik a maximális margóval, vagyis a maximális távolsággal mindkét osztály adatpontja között.

Az optimális sík megtalálása esetén ($w^T x + b = 0$) egy új pont (x_i) osztályozása (y_i) már roppant egyszerű, az algoritmus a lineáris regresszió kimenetét vizsgálja. Mivel az SVM a küszöbértékeket 1-re ($w^T x + b = 1$) és -1-re ($w^T x + b = -1$) állítja, megkapjuk azt az értéktartományt ([-1,1]), amely margóként működik. Ha a kimenet nagyobb, mint 1 ($w^T x_i + b \geq 1$), akkor azonosítjuk egy osztállyal ($y_i = 1$), és ha a kimenet kisebb, mint -1 ($w^T x_i + b \leq -1$), akkor egy másik osztállyal ($y_i = -1$).

2.2.4. Gaussian Naive Bayes

A Naive Bayes a felügyelt gépi tanulási osztályozási algoritmusok egy csoportja, amely a Bayes-tételen alapul és feltételezi, hogy az általa használt funkciók mindegyike feltételesen független egymástól, adott osztályban. Természetesen ezek a függetlenségi feltételezések ritkán igazak, de a gyakorlatban a Naive Bayes-modellek meglepően jól teljesítettek, még olyan összetett feladatoknál is, ahol egyértelmű, hogy az erős a függetlenségi feltételezések hamisak.

$$P(A|B_1, B_2 \dots B_n) = \propto P(A)P(B_1, B_2 \dots B_n|A) \approx \propto P(A) \prod_{i=1}^{n} P(B_i|A)$$

2.2.4.1-es képlet: Naive Bayes tétele

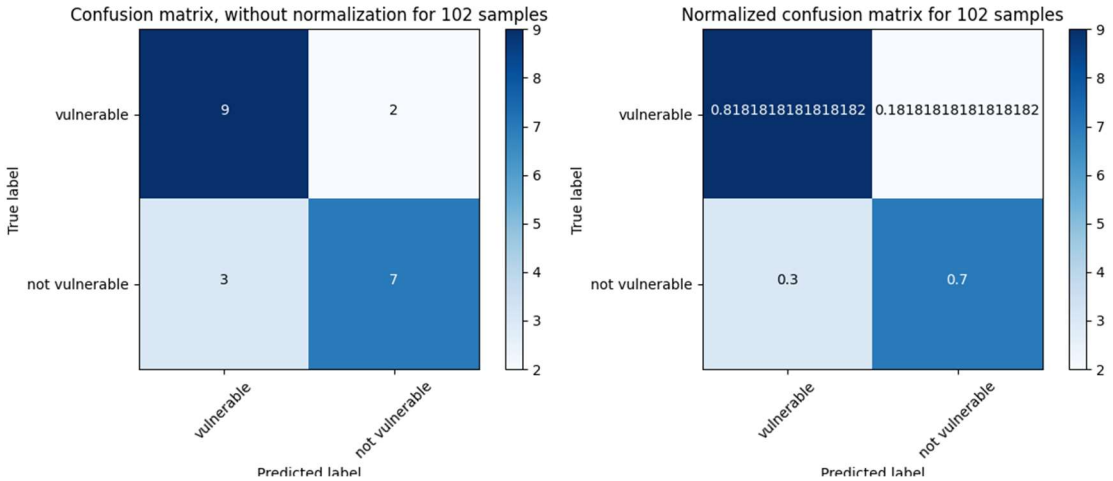
Háromféle Naive Bayes osztályozó létezik, a normál (Gauss-) eloszlás alapján történő osztályozáshoz a Gaussian Naive Bayes-t használják, ekkor Naive Bayes tétele az új adattömb minden szavához ($B_1, B_2 \dots B_n$) ad egy valószínűségi értéket az adott osztályra (A) vonatkozóan, amelyek a számítás végén összegzésre kerülnek. Értelemszerűen az az osztály kerül kiválasztásra, amelynek nagyobb a valószínűsége.

2.2.5. Random Forest

Az együttes tanulási módszerek (Ensemble Methods) technikailag több felügyelt tanulási modellt tartalmaznak, amelyeket egyénileg képeznek ki. Az eredmények különböző módon egyesülnek a végső „jóslat” elérése érdekében, ez által az azoknak nagyobb a prediktív ereje, mint bármelyik önálló tanulási algoritmusénak.

A Random Forest osztályozás egy, az együttes tanulási módszerek közül, amely lényegében több döntési fát összesít, néhány hasznos módosítással. Egyrészt az egyes csomópontokban felosztható funkciók száma a teljes érték bizonyos százalékára korlátozódik (ami hiperparaméter néven ismert). Ez biztosítja, hogy az együttes modell ne támaszkodjon túlzottan egyetlen egyedi tulajdonságra sem és felhasználja az összes potenciálisan prediktív tulajdonságot. Másrészt minden fa egy véletlenszerű mintát vesz az eredeti adatsorból, amikor a felosztásokat létrehozza, és további véletlenszerűség-elemeket ad hozzá, amelyek megakadályozzák a túlillesztést.

2.3. Confusion Matrix



2.3.1-es ábra: normalizálás nélküli és normalizált Confusion Matrix a diagram.py-ból, 21 tesztadat esetén

A Confusion Matrix olyan táblázat, amelyet gyakran használnak egy osztályozási modell teljesítményének leírására olyan vizsgálati adatok halmazán, amelyek ismerik a valós értékeket. Ez egy táblázat, amely négy előrejelzett és tényleges érték kombinációját tartalmazza bináris osztályozó esetén, amelyekből következtethetünk adott algoritmus pontosságára:

- igaz pozitív (true positive, TP): eredmény, ahol a modell helyesen jósolja meg a pozitív osztályt

- igaz negatív (true negative, TN): eredmény, ahol a modell helyesen jósolja meg a negatív osztályt
- hamis pozitív (false positive, FP): eredmény, ahol a modell helytelenül pozitívnak jósol egy negatív osztályt
- hamis negatív (false negative, FN): eredmény, ahol a modell helytelenül negatívnak jósol egy pozitív osztályt

Ezekkel az eredményekkel olyan teljesítménymutatók hívhatók elő, amelyek hasznosak az osztályozó működésének gyors értékeléséhez.

Precision

A Precision a helyesen megjósolt pozitív megfigyelések (true positive) és az összes jósolt pozitív megfigyelések (total predicted positive), tehát a helyesen és a helytelenül megjósolt pozitív megfigyelések összegének aránya. A Precision akkor hasznos, ha az adatpontok helytelen pozitívként történő osztályozásának költsége nagyon magas, például amikor egy számunkra fontos levelet spamnek minősítenek.

Kiszámítása a következő képlet segítségével történik: $\frac{TP}{TP+FP}$.

Recall

A Recall – avagy a valódi pozitív ráta – a helyesen megjósolt pozitív megfigyelések (true positive) és az összes tényleges pozitív megfigyelések (total actual positive), tehát a helyesen megjósolt pozitív és negatív megfigyelésének összegének aránya. A Recall akkor hasznos, ha a rossz osztályozás magas költségekkel jár például, ha a program nem sérülékenynek címkézi azt a kódot, ami egyébként az.

Kiszámítása a $\frac{TP}{TP+FN}$. képlettel történik.

F-1 score

Az F1 pontszám a precision és a recall harmonikus átlaga, ezért ez a pontszám a hamis pozitívokat és a hamis negatívokat egyaránt figyelembe veszi, valamint sokkal nagyobb súlyt ad az alacsony értékeknek. Ennek eredményeként az osztályozó csak akkor kap magas F-1 pontszámot, ha mind a Recall, mind a Precision magas.

Kiszámítása a $2 * \frac{precision*recall}{precision+rec}$ képlettel történik.

3. TERMÉSZETES NYELVFELDOLGOZÁS

A természetes nyelv feldolgozása (NLP) – ahogy a gépi tanulás is – a mesterséges intelligencia egyik területe [13]. A mesterséges intelligencia magában foglalja azokat a rendszereket, amelyek utánozzák a kognitív képességeket, például tanulnak a példákból és megoldják a problémákat. Ez az alkalmazások széles skáláját öleli fel, az önvezető autóktól a prediktív rendszerekig. A természetes nyelvfeldolgozás célja, hogy az emberi nyelv a maga összetett, nem egyértelmű, kontextus által befolyásolt és rendkívül sokszínű módján érthető legyen a gépek számára. Ezeknek a folyamatoknak az automatizálásához, az ismétlődő és időigényes feladatok elvégzéséhez gépi tanulást alkalmaznak a hatékonynöveléshez. A program lényegében a gépi tanulás és a természetes nyelvfeldolgozás metszete.

3.1. Előfeldolgozás

Az előfeldolgozás során a program egy szövegből AST (Abstract Syntax Tree) fát generál. Az absztrakt szintaxisfák olyan adatszerkezetek, amelyeket a fordítók széles körben használnak a programkód szerkezetének ábrázolására. A program szempontjából fontos, hogy a generálás eltávolítja a kódból a lényegtelen írásjeleket és elválasztókat, valamint extra információkat dokumentál a kódról, például a „kind” kulcs segítségével („variable”, „expressionstatement”, „string” stb.). Ám természetesen annak fő szavait is megőrzi például a „name” és „raw” kulcsokkal („mysql_query”, „SET NAMES utf8”).

3.2. Tokenizálás

A tokenizálás a szintaktikai elemzés* egy részfeladata. Célja egy szövegdarab szétválasztása kisebb egységekre, amelyeket tokeneknek neveznek. Itt a tokenek lehetnek szavak, karakterek vagy alszavak. Céljuk, hogy a szöveget könnyebben kezelhetővé tegyék. A szövegkivonat lehetővé teszi az előre meghatározott információk, releváns kulcsszavak kinyerését a szövegből.

A program tokenizálás segítségével lényeges és releváns információkat nyer ki az absztrakt szintaxisfa „kind”, „name” és „raw” kulcsai által, melyek értékeiből whitespace szeparálóval tokensorozatot készít.

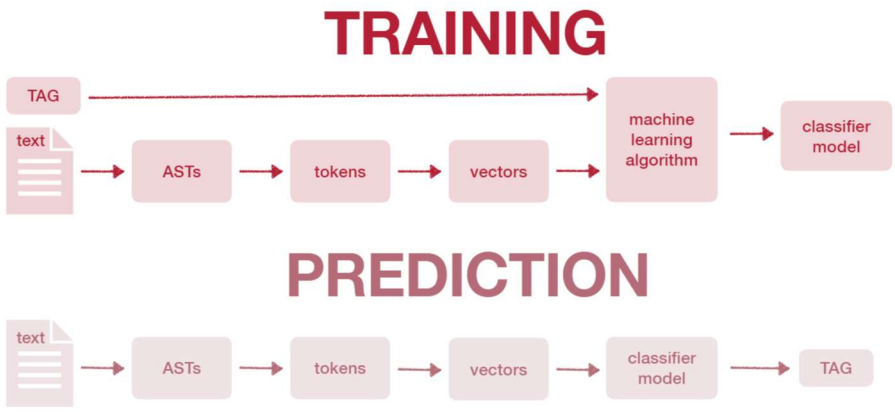
*A szintaktikus elemzés a természetes nyelv feldolgozásának egyik fő technikája, amely a szöveget alapvető nyelvtani szabályok segítségével elemzi a mondatszerkezet, a szavak szerveződésének és a szavak egymáshoz való viszonyának azonosításához.

Mivel a tokenek a természetes nyelv építőkövei, a nyers szöveg feldolgozásának leggyakoribb módja tokenek szintjén történik. Ezért a tokenizálás a legfontosabb lépés többek között szövegek osztályozásában.

3.3. Osztályozás

A szöveg osztályozása (más néven szöveg kategorizálása vagy szöveg címkézése) az egyik legalapvetőbb használati esete a természetes nyelv feldolgozásának. Feladata, hogy előre meghatározott kategóriákat rendeljen a nyílt végű szöveghez automatikusan és költséghatékonyan. Számos megközelítés létezik az automatikus szöveges osztályozásra, de ezek mind három típusú rendszerbe tartoznak: szabályalapú, gépi tanulás alapú és hibrid rendszerek. A dolgozat és a program szempontjából a második a releváns. A gépi tanulási szövegosztályozás a manuálisan kidolgozott szabályok helyett a korábbi megfigyelések alapján tanul meg osztályozni. Előre felcímkézett példák felhasználásával a gépi tanulási algoritmusok megtanulhatják a szövegdarabok közötti különböző asszociációkat, és azt, hogy egy adott bemenetre (azaz szövegre) egy adott kimenet (azaz címke) várható. A címke az előre meghatározott osztályozás vagy kategória, amely bármely adott szövegre felkerülhet.

A gépi tanulási NLP osztályozó képzésének első lépése, hogy egy módszerrel minden tokenizált szöveget vektor formában a program numerikus ábrázolássá alakít. Ezután a gépi tanulási algoritmust olyan képzési adatokkal látja el, amelyek jellemzőkészletek (vektorok az egyes szövegpéldákhoz) és címkék (pl. biztonságos, sérülékeny) párjából állnak. A képzés végén a program egy osztályozási modellt állít elő, melynek segítségével pontos előrejelzések készíthetők.



3.3.1-es ábra: a képzési és előrejelzési folyamat a kód NLP osztályozásában

A szöveges osztályozás a gépi tanulással általában sokkal pontosabb, mint az ember által kidolgozott szabályrendszerek, különösen az összetett NLP osztályozási feladatoknál. A gépi tanulással rendelkező osztályozókat is könnyebb karbantartani, és mindig új példákat jelölhet meg az új feladatok megtanulásához.

4. PROGRAM FELÉPÍTÉSE

Ebben a fejezetben ismertetem a program felépítésének lépéseit a korábbi fejezetet alapul véve. Az implementálás legelső lépésével kezdnem, az adatgyűjtéssel. Az adatok gyűjtése során több forrást használtam sérülékeny és nem sérülékeny PHP kódok keresésére. Példának okáért több adat származik a DVWA-ról (Damn Vulnerable Web Application), GitHub projektekről és a CVE (Common Vulnerabilities and Exposures) adatbázisból. A tényleges tanuló kódok és azok forrásaik a tanulo_adatok.md fájlban találhatóak. A 102 darab tanuló kód, amely a tanulo_adatok mappában helyezkedik el, a gépi tanulási modellem alapja.

```
<?php
mysql_query('SET NAMES utf8');
$var = mysql_real_escape_string("\xbf\x27 OR 1=1 /*");
mysql_query("SELECT * FROM test WHERE name = '$var' LIMIT 1");
```

4.1-es ábra: pdo1.php, egy nem sérülékeny kód a tanulo_adatok mappában

Az 57 darab sérülékeny és 45 darab nem sérülékeny kód legelső lépésben AST generáláson esik át, amihez az irodalomjegyzék [14] elemét használtam. Ez egy manuális folyamat volt, de a parser.js a front-end által bekért kódra ezt már automatizálja az irodalomjegyzék [15] eleme segítségével. A PHP fájlokhoz generált AST fák az AST mappábában közel azonos névvel helyezkednek el .txt kiterjesztéssel.


```
            "offset": 151
          }
        },
        "value": "' LIMIT 1",
        "raw": "' LIMIT 1",
        "unicode": false,
        "isDoubleQuote": false
      },
      "syntax": null,
      "curly": false
    }
  ],
  "raw": "\"SELECT * FROM test WHERE name = '$var' LIMIT 1\"",
  "type": "string"
}
]
```


4.2-es ábra: pdo1_ast.txt, a 3.1-es ábra kódjának AST megfelelője

A tokenizer.py ezekkel dolgozik tovább. Ez a fájl a NLP (Natural Language Processing) egy igen gyakori technikáját, a tokenizálást – utalva ezzel a fájl nevére – alkalmazza, amely a szövegfájlokat kisebb részekre, ún. tokenekre bontja a könnyebb kezelhetőség érdekében [16]. Elsőként listázza és egy ciklussal sorra beolvassa az AST

mappa .txt kiterjesztésű fájljait. Majd ezeknek a fájloknak a sorait JSON felépítésből adódó kulcsok alapján szelektálja („kind”, „name”, „raw”). Ezeknek az értékei adják a fájlok fontos szavait és kifejezéseit, így azonos nevű .csv kiterjesztésű fájlok első oszlopaiba írja azokat listákba fűzve. Az oszlopokat egy olyan karakter választja el, ami az értékben – továbbiakban tokenekben – nem fordul elő, ezáltal a program egy automatizált lépésben nem hoz létre több oszlopot, mint szükséges. Ez a karakter a ^.

```
program expressionstatement call name mysql_query string 'SET NAMES utf8' expressionstatement assign
variable var call name mysql_real_escape_string string \\\xbf\\x27 OR 1=1 /*\ expressionstatement
call name mysql_query encapsed encapsedpart string SELECT * FROM test WHERE name = ' encapsedpart
variable var encapsedpart string ' LIMIT 1 \SELECT * FROM test WHERE name = '$var' LIMIT 1\^
```

4.3-as ábra: pdo1_ast.csv, a nodes.py kimenete, a 3.2-es ábra bemenetre

A következő lépésben a .csv fájlokat fűzi össze a concatenate.py egy csv fájlba concatenated néven. Ezen a ponton manuálisan felcímkéztem őket annak duplikáltjában, a concatenated_titled.csv második oszlopában, mint sérülékeny (0) vagy nem sérülékeny (1) sor. A duplikálás oka, hogy minden egyes újrafuttatásakor a concatenate.py újraírja a concatenated.csv-t ezáltal, ha a manuális változtatások nem kerülnének át más fájlba, akkor elvesznének. Az összefűzés oka pedig, hogy későbbiekben az egyes tokenek gyakorisága lesz fontos az összes .csv viszonylatában, nem csak a sajátjában.

Így a vectorizer.py már az összefűzött és felcímkézett concatenated_titled.csv-t dolgozza fel. A fájl legfőbb feladata, hogy olyan numpy fájlokat hozzon létre, amiket később az osztályozásért felelős fájl használni tud.

Kifejtve a főbb metódusait:

- __init__: A *Vectorizer* osztály konstruktora, ami paraméterként kér egy fájl nevet (*filename*), azoknak a tokeneknek a maximális számát, amelyeket megkülönböztetünk egymástól (*max_vocab*) és annak a maximális számát, hogy hány token van ténylegesen (*max_length*). Ez utóbbit a kód automatikusan számolja ki úgy, hogy későbbiekben az *X* vektorok ~10%-a csorduljon túl az értéken. A paraméterek változókba helyezésén kívül még inicializálva van egy számláló (*word_freqs*), két szótár (*word2index, index2word*), egy mátrix (*dataset*) és a kimeneti vektoroknak egy-egy változó (*X, y*).
- collectVocab: Metódus, ami ciklussal bejárja a *dataset* 0.oszlopának sorait, szóköz alapján szétválasztja a token sorozatokat és az egyes tokeneket az

előfordulási számukkal a *word_freqs* változóba teszi. A collection.Counter típusából adódóan biztosítja változón belül a csökkenő sorrendet.

- createLookupTables: Metódus, amely indexekkel keresőtáblákat készít a *word_freqs* tokenjeinek, ezek a *word2index* és az *index2word.*
- createVectors: Metódus, ami ciklussal végigmegy egyrészt a *dataset* 0.oszlopának szétdarabolt tokenjein, és a keresőtábla alapján a hozzájuk tartozó indexet az *X* változóba listázza. Amennyiben az *X* változó hossza kevesebb, mint a *max_length* értéke, a különbséget nullákkal tölti fel. Ha a hossza nagyobb, a különbséget levágja a lista végéről. Így a vektorok hossza egységes marad. Másrészt a *dataset* utolsó oszlopának bináris számait is kigyűjti, ezeket az *y* változóba.

```
X: [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0   20    4    5    7 1072    3 1073    4    8    2
  323    5    7  612    3  259  145  260  261    4    5    7  433   42
   16    3   55   38   17   54   19    7   23   24   16    2  323   16
    3   24   79  613   34   38   17   54   19    7   23  614   79  187]
y: 1.0
```

4.4-es ábra: az *X[40]* és a*z y[40]* eredménye; az *X* és *y* 40.eleme az concatenated_titled.csv 40.sorának felel meg, ami ekvivalens a 3.3-as ábrával felcímkézés után

- saveVectorAs: Metódus, ami az *X* és *y* változókat lementi egy-egy numpy fájlba.

A program utolsó és legfontosabb lépésében a trainer.py-on belül hajtja végre a program legfőbb feladatát: betanítja a választott algoritmust és ad egy becslést egy ismeretlen PHP kód SQL injectionbeli sebezhetőségével kapcsolatban.

Implementált metódusai:

- main: Metódus, amely egyrészt beolvassa parancssori argumentumokon keresztül a „toimportX” numpy fájt (-z), amiket a classifier2 metódusnak ad át, másrészt beolvassa hasonló módon (-x, -y) a „testX” „testy” numpy fájlokat is, amiket feloszt az algoritmus tanítására (*trainX, trainY*) és

]

tesztelésére (*testX, testY*) ~80~20% arányban. A felügyelt tanulás 5 fő algoritmusára van meghívva a classifier metódus. Az általa visszaadott metrikák segítségével a main összehasonlítja az algoritmusokat, aminek az eredményét a 3.6-os ábra táblázata mutatja. Ekkor automatikusan kiválasztja a legnagyobb értékhez tartozó osztályozót és meghívja rá a classifier2 metódust.

- test_bmodel: Metódus, amely visszaadja a paraméterként megadott algoritmus hatékonyságának számát egy [0;1] intervallumon a szintén paraméterként megadott teszt adatok által. Már említettem a korábbi fejezetben, hogy a Confusion Matrix Recall értéke a leghasznosabb, ha egy kód sérülékenységéről van szó, így ezt a metrikát használom a hatékonyságméréshez.
- classifier: Metódus, amely a paraméterben megadott tanító és tesztelő adatok alapján képzi és értékeli ki a szintén paraméterbeli osztályozót. Kimenete a test_bmodel által visszaadott metrika, valamint kiírja, hogy a ciklus hanyadik futásnál jár, a korábban említett metrikát és az osztályozót.
- classifier2: Metódus, amely a paraméterként megadott osztályozóval becslést ad a szintén megadott *X* vektor sérülékenységével kapcsolatban. Kiírja többek között az osztályozót, az osztályozás eredményét, annak a valószínűségét, hogy a vektor sérülékeny és a valószínűségét, hogy nem sérülékeny. A 3.6-os ábra utolsó három sora erre ad példát.

```
24      0.5     CalibratedClassifierCV(base_estimator=LinearSVC(max_iter=10000))
+-----------------------------------------------------------------+--------+
|                            classifier                           | recall |
+-----------------------------------------------------------------+--------+
|                           GaussianNB()                          |  0.71  |
|                     RandomForestClassifier()                    |  0.74  |
|                      KNeighborsClassifier()                     |  0.58  |
|      LogisticRegression(max_iter=10000, multi_class='ovr')      |  0.54  |
| CalibratedClassifierCV(base_estimator=LinearSVC(max_iter=10000)) |  0.51  |
+-----------------------------------------------------------------+--------+
Choosed classifier: RandomForestClassifier()
Result of classification: [0]
0.56
0.44
```

4.5-ös ábra: a trainer.py kimenete

A modellhez utolsó lépésként az asztali alkalmazás front-end oldaláról az first.fxml felhasználói bemenete biztosít ismeretlen adatokat, amelyeket a back-end firstController-e toimport_vulnerabilities.php néven ment el a többi PHP kód közé,

majd navigál át a PrimaryController-be. Itt a kód a vektorizálásig ugyan azokon a lépéseken megy végig, mint a többi, ám nem lesz összefűzve velük. Helyette önálló vektorokat kap, amiket a vecotirzer.py a „toimport" numpy fájlokba ment le. Ezeket olvassa be a main metódus és ad át a classifier2-nek. A front-end új kódok bekérésén kívül az eredmények kiíratásáért is felelős: a classifier2 által kiírt utolsó két sor valószínűségi számait a PrimaryController olvassa be és a hozzá tartozó primary.fxml szemlélteti kördiagram segítségével. Emellett további információkat is szolgáltat többek között az SQL injection veszélyeiről [17], a megelőzési módjairól [18], valamint megjeleníti a korábban megadott kódot és a program futási idejét.

Végezetül kitérnék a diagram.py-ra is, amely funkcióját tekintve ábrákat szolgáltat a dolgozat egyes részeinek szemléltetéséhez. Ilyen ábrák többek között a 2.1.1-es ábra, a 2.3.1-es ábrák, a 5.1.1-es ábra, közvetetten a 5.2.1-es ábra, amelynek futásai Confusion Matrix-beli értékeket jelölnek, és a 5.2.2-es ábra.

Főbb metódusai:

- main: Metódus, ami a trainer.py-hoz hasonlóan adatokat hív és különít el tanító, valamint tesztelő példákra. Mint fő metódus, hívja többek között a *plot_classifier_comparison, plot_classification, classifier és confusionm* függvényt.
- classifier: Metódus, amely a trainer.py-hoz hasonlóan a paraméterben megadott osztályozót betanítja a paraméterbeli tanító adatok segítségével, majd visszatér az osztályozóval.
- confusionm: Metódus, amely a *classifier* metódus által visszaadott osztályozóhoz tesztelő adatok és a *plot_confusion_matrix* segítségével normalizált és normalizálás nélküli Confusion Matrix-ot jelenít meg.
- plot_confusion_matrix [19]: Metódus, amely a Confusion Matrix építéséért felelős.
- plot_classification [20]: Metódus, amely a korábban tárgyalt 5 osztályozó „kalibráltságát" szemlélteti egy diagramon paraméterben adott tanító és teszt adatokkal.
- plot_classifier_comparison [21]: Metódus, amely a korábban tárgyalt 5 osztályozó pontosságát mutatja be különböző eloszlású adatokkal.

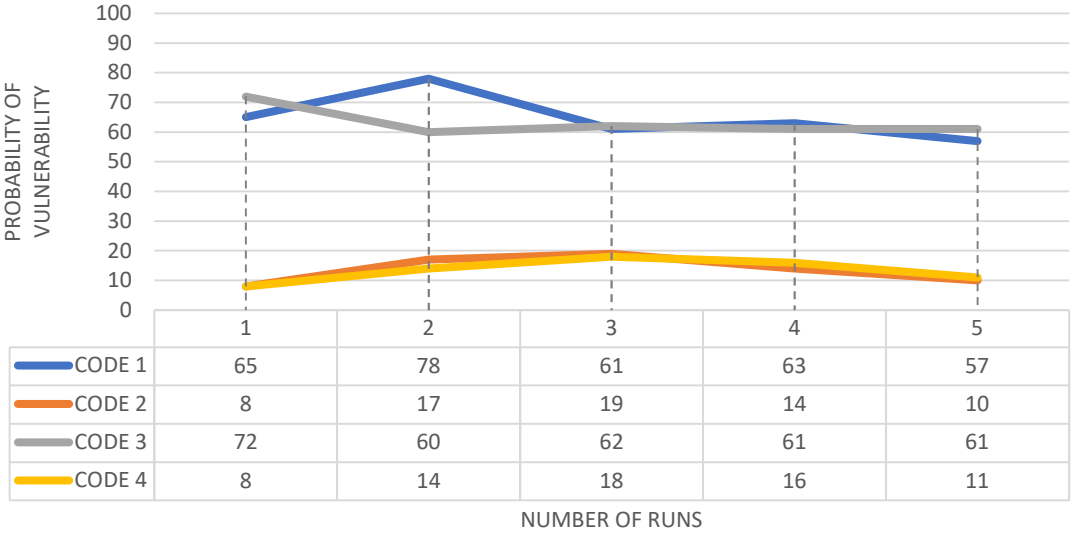
5. EREDMÉNY ÉS ANALÍZIS

5.1. Követelménymegvalósítás

A programot e fejezet írása során 4 kóddal teszteltem 5 futtatás során az eredmények szemléltetéséhez és analizálásához. Ezek között 2 sérülékeny és 2 nem sérülékeny példa van. A cél annak kiszűrése volt, hogy az egyes PHP kódokat hogyan étrékeli ki és hányadik futáskor ad elvárt értéket, amennyiben ad.

A 4.1-es ábrán láthatóak az eredmények: a „CODE 1” és a „CODE 3” azonos helyről, az irodalomjegyzék [22] forrásból származik és az 5 futás során a küszöbérték felett helyezkednek el a valószínűségi értékeik. Ez a két példa -ahogy a tanuló adatok egy része – nem igényelt előzetes ellenőrzést az elvárt osztály meghatározásához, mivel a projekt sérülékenynek lett implementálva. A „CODE 2” és „CODE 4” két nem sérülékeny példa a [23] forrásból, amelyek rendre PDO-t és MySQLi-t tartalmaznak, azok legáltalánosabb formájukban. Ez esetben a program 50% alatt jelöli az értékeiket, amely ekvivalens az elvárt működéssel.

Továbbá a futás számán is látható, hogy a helyes osztályt első alkalommal határozza meg: Az ötödik futásig a valószínűség kilengése rendre -8+13%, +11%, -12%, +10% osztályon belül. Ez által a program sikeresen szűri ki a sérülékeny és nem sérülékeny kódokat a legkorábbi futás alkalmával is.



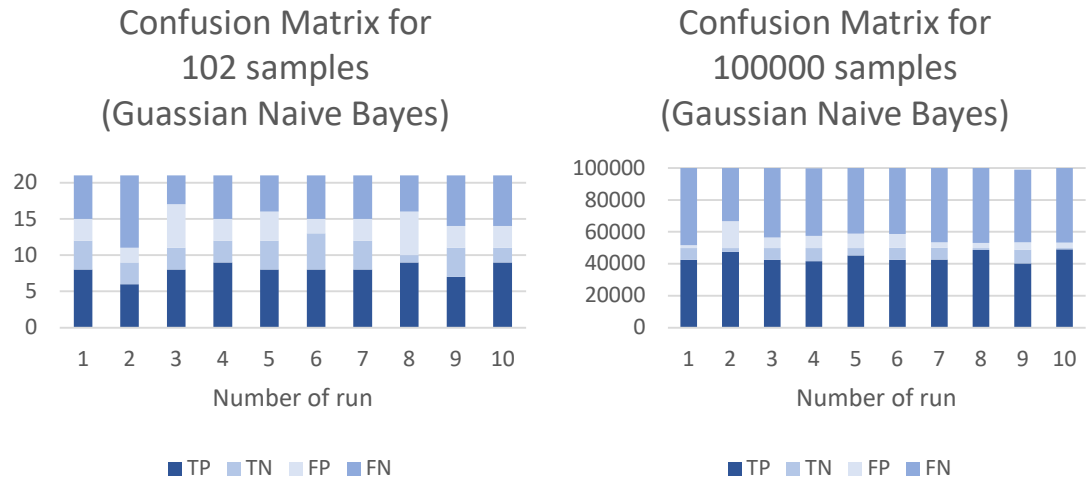
	1	2	3	4	5
CODE 1	65	78	61	63	57
CODE 2	8	17	19	14	10
CODE 3	72	60	62	61	61
CODE 4	8	14	18	16	11

5.1.1-es ábra: a program futásának eredménye 2 sérülékeny és 2 nem sérülékeny kódra

5.2. Erőforrás- és pontosságoptimalizálás

Ennek első lépése a megfelelő metrika kiválasztása volt, amely az osztályozók pontosságainak mérésére alkalmas. Ahogy a második fejezetben említettem, sebezhetőség meghatározásakor a rossz osztályozás magas költségekkel jár például, ha a program nem sérülékenynek címkézi azt a kódot, ami egyébként az. Így a metrikák közül a Recall bizonyult személyre szabottnak és egyben legkézenfekvőbbnek.

A felépített modell a tanuló adatok számának csekélysége ellenére is nagyobb mértékben ad igaz pozitív és hamis negatív példát. Az adatok számának növelésével természetesen ez az arány nőne, míg az igaz negatív és hamis pozitív aránya csökkenne, ezt szemlélteti a 5.2.1-es ábra. Ennek hatására az osztályozó precízebben és pontosabban tudná meghatározni az adatok osztályát, ez által úgymond jobban „kalibrált” lenne. A 5.2.2-es ábrán látható tisztán, hogy több adattal egyes osztályozók jobban illeszkednek arra az egyenesre, amit „tökéletesen kalibráltnak” nevezünk. Ekkor az osztályozás által adott valószínűség pontosan megegyezik a tényleges valószínűséggel. Ezesetben nincs igaz negatív és hamis pozitív adat. Az ideális és egyben irreális cél ezek megszűntetése, ám belátható, hogy a probléma minimalizálása sokkal reálisabb, mint annak megszűntetése. Ez által adott körülmények között (például csekély rendelkezésre álló idő) kézenfekvőbb a szükséges adatszám meghatározásához a program kívánt teljesítményét, pontosságát használni.



5.2.1-es ábra: az előrejelzett és tényleges értékek kombinációja 21 tesztadatnál (102 tanuló adat) és 99900 tesztadatnál (100000 tanuló adat)

Hogy összefoglaljam, a program megpróbálja a legjobb pontosságot elérni minimális erőforrás-mennyiséggel, emellett megfelelően választott osztályozóval. Ahhoz, hogy a releváns osztályozók közül a legpontosabb kerüljön kiválasztásra – adott metrika alapján –, a program ezen részének automatizálása által értem el. Ez végeredményképp 102 „erőforrás” és ~70%-os „pontosság”.

5.3. Összefoglalás

A dolgozatban az SQL Injection detektálás problémát gépi tanulási algoritmusok és természetes nyelvi feldolgozó rendszerek alkalmazásával közelítettem meg. Osztályozási módszerrel osztályozzuk a bejövő kódot sebezhető vagy biztonságos kódként, előzetes AST generálás, tokenizálás és vektorizálás segítségével. Öt gépi osztályozási algoritmust teszteltem tanuló adatokkal, amelyek a Logistic Regression, K-Nearest Neighbors, Linear Suppoet Vector, Gaussian Naive Bayes és Random Forest osztályozó. A bejövő kódot a legpontosabbal osztályoztam.

Az elért eredmények fényében a téma által megfogalmazott informatikai szakmai problémára a program megoldást nyújtott, mint SQL injection detektálási mechanizmus. Tehát kijelenthető, hogy a program a bemenetként megadott PHP kódra kimenetként sikeresen meghatározza azt, hogy milyen valószínűséggel tartozik a sérülékeny és nem sérülékeny osztályba, mindezt erőforrás- és pontosságoptimalizáltan.

Optimalizálást mellőzve, sokszorosan nagyobb tanuló adatbázissal és erőforrással ez a program is tovább fejleszthető, egyebek mellett még legalább 9 különböző sérülékenytípusra kiterjeszthető. Tovább tetőzve, új, soha nem látott támadások azonosítására kiképezhető. Tekintve, hogy ez egy aktuális probléma, ahogy a dolgozatom is aktív kutatási téma [24], ezek fejlesztésére van precedens.

Bizakodom, hogy a jövőben ez által fogok a dolgozat témájához vagy a bekezdésben felsoroltakhoz hasonló feladattal találkozni, mint alkalmazott kiberbiztonsági szakember és tudom hasznosítani a program és dolgozat megvalósítása során tanultakat.

Irodalomjegyzék

1. OWASP Top Ten
   https://owasp.org/www-project-top-ten/

2. Vidushi, Prof. S. Niranjan, 2015, A Review on SQL Injection, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) NCETEMS – 2015 (Volume 3 – Issue 10)
   https://www.ijert.org/a-review-on-sql-injection

3. SQL injection
   https://portswigger.net/web-security/sql-injection

4. SQL Injection
   https://www.w3schools.com/sql/sql_injection.asp

5. Error Based SQL Injection (SQLi), 2018
   https://beaglesecurity.com/blog/vulnerability/error-based-sql-injection.html

6. Raj Chandel, 2017, Beginner Guide to SQL Injection Boolean Based (Part 2)
   https://www.hackingarticles.in/beginner-guide-sql-injection-boolean-based-part-2/

7. Time Based Blind SQL Injection (SQLi), 2018
   https://beaglesecurity.com/blog/vulnerability/time-based-blind-sql-injection.html

8. Bogdan Calin, 2015, Blind Out-of-band SQL Injection vulnerability testing added to AcuMonitor
   https://www.acunetix.com/blog/articles/blind-out-of-band-sql-injection-vulnerability-testing-added-acumonitor/

9. Mike Shema, 2012, SQL Injection & Data Store Manipulation
   https://www.sciencedirect.com/topics/computer-science/prepared-statement

10. SQL Injection Prevention Cheat Sheet
    https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

11. A. Joshi and V. Geetha, "SQL Injection detection using machine learning," 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2014, pp. 1111-1115, doi: 10.1109/ICCICCT.2014.6993127.
    https://ieeexplore.ieee.org/document/6993127

12. Alex Kaechele, 2018, Classification
https://rpubs.com/alexkaechele/380330

13. Inés Roldós, 2020, NLP, AI, and Machine Learning: What's The Difference?
https://monkeylearn.com/blog/nlp-ai/

14. AST explorer
https://astexplorer.net/

15. php-parser
https://github.com/glayzzle/php-parser

16. Kar D., Panigrahi S., Sundararajan S. (2015) SQLiDDS: SQL Injection Detection Using Query Transformation and Document Similarity. In: Natarajan R., Barua G., Patra M.R. (eds) Distributed Computing and Internet Technology. ICDCIT 2015. Lecture Notes in Computer Science, vol 8956. Springer, Cham.
https://doi.org/10.1007/978-3-319-14977-6_41

17. SQL Injection
https://owasp.org/www-community/attacks/SQL_Injection

18. SQL Injection Prevention Cheat Sheet
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

19. Confusion matrix
https://scikit-learn.org/0.18/auto_examples/model_selection/plot_confusion_matrix.html

20. Comparison of Calibration of Classifiers
https://scikit-learn.org/stable/auto_examples/calibration/plot_compare_calibration.html#sphx-glr-auto-examples-calibration-plot-compare-calibration-py

21. Classifier comparison
https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

22. VulnerableApp-php
https://github.com/SasanLabs/VulnerableApp-php

23. How can I prevent SQL injection in PHP?
https://stackoverflow.com/questions/60174/how-can-i-prevent-sql-injection-in-php

24. Mishra, Sonali, "SQL Injection Detection Using Machine Learning" (2019). Master's Projects. 727. https://doi.org/10.31979/etd.j5dj-ngvb

Nyilatkozat

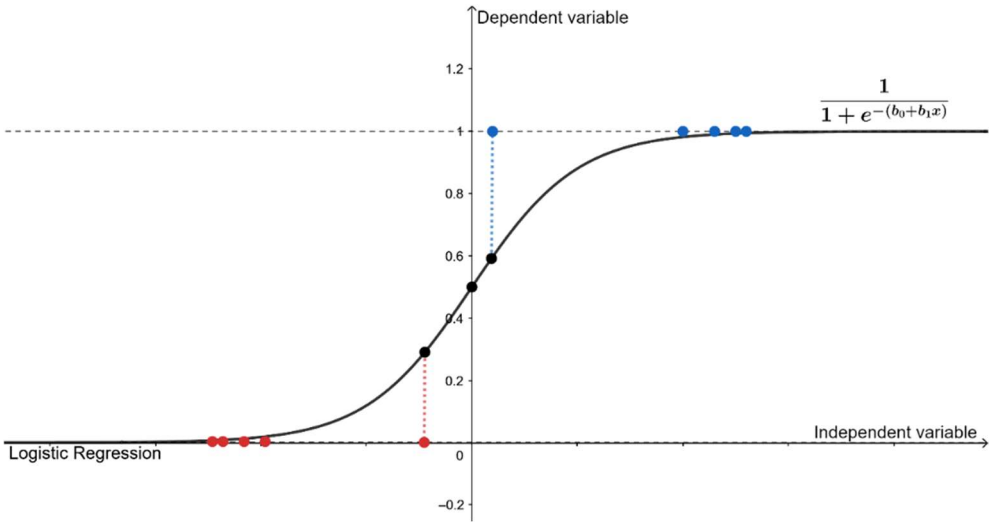
Alulírott Stiller Marianna programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, programtervező informatikus BSc diploma megszerzése érdekében. Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök stb.) használtam fel. Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

2021. .....

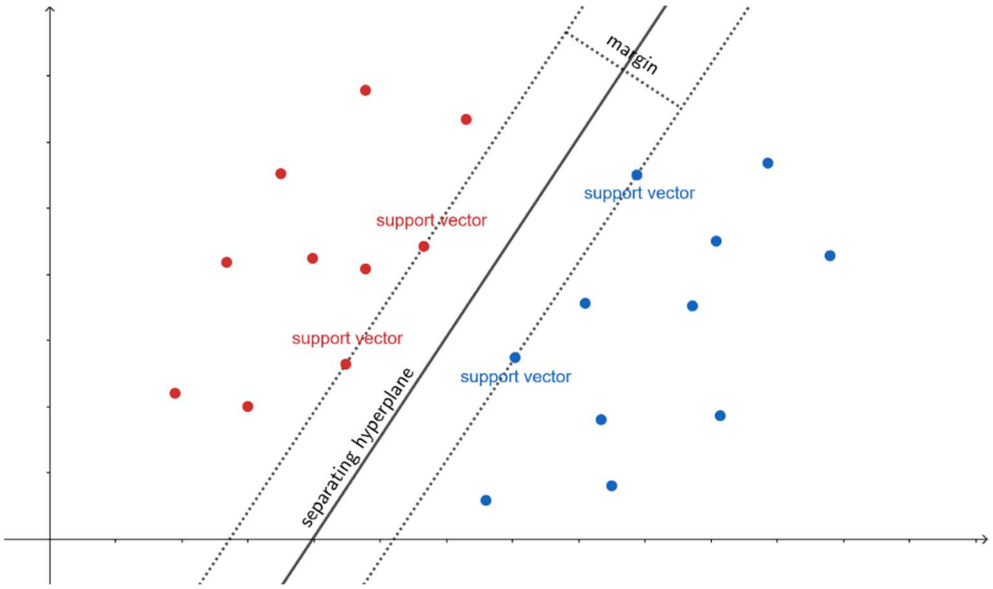
Köszönetnyilvánítás

Szeretnék köszönetet mondani egykori gyakorlat vezetőmnek, Tóth Lászlónak és előadómnak, Vidács Lászlónak, akik bevezettek az információbiztonság világába és segítettek a program implementálása és a szakdolgozat megírása során. Bízom benne, hogy ezzel olyan úton indítottak el, amin még évtizedek múlva is boldogan járhatok.

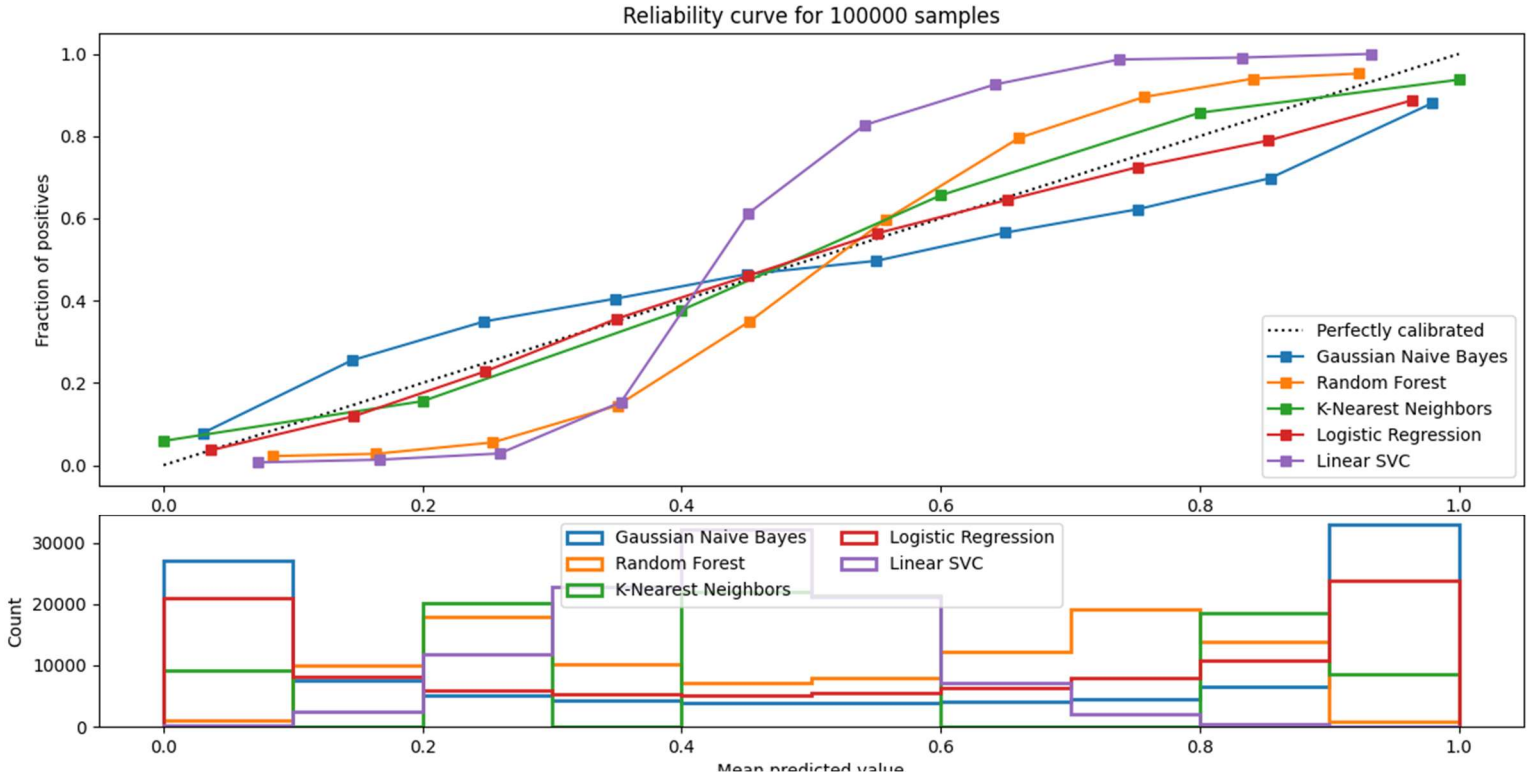
Mellékletek



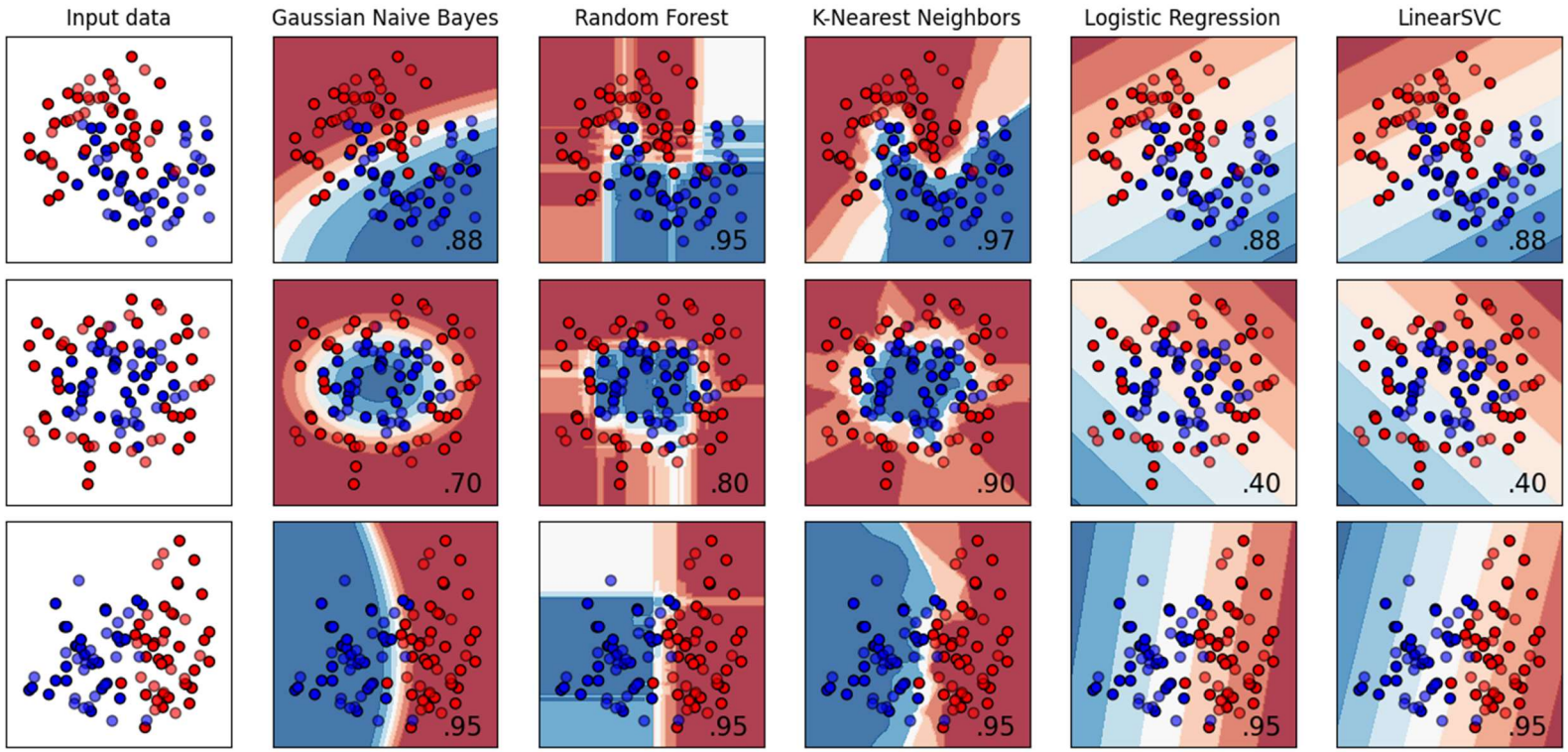
2.2.1.3-as ábra: a Logistic Regression Classifier osztályozásának szemléltetése



2.2.3.1-es ábra: a Linear Support Vector Classifier osztályozásának szemléltetése



5.2.2-es ábra: osztályozók „kalibrálásának” összehasonlítása a diagram.py-ból, 100000 tanuló adatnál



2.1.1-es ábra: az osztályozók pontossága (a négyzetek jobb alsó sarkában) különböző eloszlású adatok esetén a diagram.py-ból