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: |  | Témavezető: |  |
|  | Stiller Marianna |  | Dr. Vidács László |  |
|  | programtervező informatika szakos hallgató |  | 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:

Minták által tanított program képes reális becslést adni egy kód SQL injectionbeli sebezhetőségével kapcsolatban.

A megoldási mód:

PHP kódokból generálok absztrakt szintaxis fákat, amelyeket felcímkézek, mint sérülékeny és nem sérülékeny példák. Ezek átesnek egy vektorizáláson, majd egy kifejezett modell 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:

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

Elért eredmények:

A program felhasználói felületen a bemenetként megadott PHP kódra kimenetként meghatározza azt, hogy a kód milyen valószínűséggel sérülékeny.

Kulcsszavak:

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

## Tartalomjegyzék

[Feladatkiírás 2](#_Toc72158194)

[Tartalmi összefoglaló 3](#_Toc72158195)

[Tartalomjegyzék 4](#_Toc72158196)

[**BEVEZETÉS 5**](#_Toc72158197)

[1. SQL INJECTION 6](#_Toc72158198)

[1.1. Típusai 6](#_Toc72158199)

[1.1.1. Sávon belüli (in-band, classic) SQLi 6](#_Toc72158200)

[1.1.2. Vak (blind, inferential) SQLi 7](#_Toc72158201)

[1.1.3. Sávon kívüli (out-of-band) SQLi 8](#_Toc72158202)

[1.2. Megelőzési módjai 8](#_Toc72158203)

[**2. GÉPI TANULÁS 11**](#_Toc72158204)

[2.1. Felügyelt tanulás 11](#_Toc72158205)

[2.2. Osztályozás 13](#_Toc72158206)

[2.2.1. Logistic Regression 13](#_Toc72158207)

[2.2.2. K-Nearest Neighbors (K-NN) 13](#_Toc72158208)

[2.2.3. Linear Support Vector (LinearSVC) 14](#_Toc72158209)

[2.2.4. Gaussian Naive Bayes 15](#_Toc72158210)

[2.2.5. Random Forest 15](#_Toc72158211)

[2.3. Confusion Matrix 16](#_Toc72158212)

[**3. PROGRAM FELÉPÍTÉSE 18**](#_Toc72158213)

[**4. EREDMÉNY ÉS ANALÍZIS 23**](#_Toc72158214)

[4.1. Követelménymegvalósítás 23](#_Toc72158215)

[4.2. Erőforrás- és pontosságoptimalizálás 24](#_Toc72158216)

[4.3. Összefoglalás 25](#_Toc72158217)

[Irodalomjegyzék 26](#_Toc72158218)

[Nyilatkozat 28](#_Toc72158219)

[Köszönetnyilvánítás 29](#_Toc72158220)

## 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 helyezzenek a „használók” elé.

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. A dolgozatban ki fogom fejteni többek között ezt a fajta sérülékenységet, a tanulási módszereket és a kettő ötvözetét egy programon keresztül, amely kimutatja 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 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--* [1] 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--* [1]*.* A lekérdezés eredménye az adatbázis összes felhasználójának neve és jelszava.

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

## Típusai

* + 1. Sávon belüli (in-band, classic) SQLi

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*[2]. Így a tényleges lekérdezés *SELECT \* FROM Users WHERE UserId = 105; DROP TABLE Susppliers*[2]*;* lesz. Ez a támadási forma a pontosvessző SQL béli funkcióját használja ki halmozott lekérdezések indításához.

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'* [3] 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. Vak (blind, inferential) SQLi

Az következtetéses befecskendezés esetén - ellentétben a sávon belüli injektálással - hosszabb ideig tarthat, amíg a támadók kihasználják, 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 --+* [4]egy lekérdezést generál: *SELECT title, description, doby FROM items WHERE id=1’ AND 1=0*[4]. 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 --+*[4]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)* [5]. 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)*[5], amely esetén ha a kiszolgáló válasza legalább 15 másodpercet vesz igénybe, megállapíthatjuk, hogy ez az adatbázis-kiszolgáló a MySQL 5.x verzióját futtatja.

* + 1. Sávon kívüli (out-of-band) SQLi

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'+--+*[6]. Dekódoláskor a *%20* és a *%5c* a szóköznek és a 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\' –*[6], amely az alábbi lekérdezést eredményezi: *SELECT \* FROM product WHERE id=1;EXEC master..xp\_dirtree ’\\test.attacker.com\’ --*[6]. Ezzel a támadó a *test.master.com* webhelyre kényszeríti a DNS kérést, a támadás sikeres.

## 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.



1.2.1-es ábra: példák elkészített utasításokra [7]

**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*).



1.2.2-es ábra: Java példa tárolt eljárásra [8]

**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.



1.2.3-as ábra: Java példa adatbázis tábla neveinek ellenőrzésére [8]

**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.

1. GÉPI TANULÁS

A gépi tanulás az adatok mintázatának automatikus felfedezésének technikája. Forráskód biztonsági kontextusban a gépi tanulást használják arra, hogy meghatározzák a forráskód sérülékeny vagy nem sérülékeny kódként történő besorolásának legpontosabb és leghatékonyabb módját a digitális eszközök legjobb védelme érdekében.

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.

## 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, valamint a megoldandó probléma típusától. A 2.3 alfejezeten kerül kifejtésre az osztályozási algoritmusok teljesítményének mérése, amely lehetőséget ad azok összehasonlítására és teljesítményorientált szelektálására.

1. 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.

1. 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.

1. 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.

1. 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. Többek között kétségbevonhatatlan fontosságának okán alakult ki több kategória 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.

## 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).

## 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 *b0* és *b1* pedig a konstansokat.

2.2.1.1-es-ábra: a lineáris regresszió képlete

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

2.2.1.2-es ábra: 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.

## 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. A 2.2.2.1-es ábra többek között szemlélteti, amikor a 3 legközelebbi szomszéd közül 2 db (67%) tartozik az 1-es osztályba és csak 1 db (33%) a 0-ás osztályba. Ekkor az algoritmus az 1-eshez rendeli az új esetet. Továbbá azt az esetet is bemutatja, amikor a 9 legközelebbi szomszéd közül 3 db (33%) tartozik az 1-es osztályba és 6 db (67%) a 0-ás osztályba. Így a K értékének megválasztása már jóval fontosabbá válik, ezt szemlélteti három példán keresztül a 2.2.3-as ábra.



2.2.2.2-es ábra: a K-NN algoritmus pontossága a K érték függvényében [9]

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.

Bár emellett is a legegyszerűbb osztályozások egyike, de csakúgy, mint szinte mindegyik más, a K-NN is az általa alkalmazott mélyen gyökerező matematikai elméletek miatt működik. A K-NN megvalósításakor az adatpontok jellemző vektorokká vagy matematikai értékekké alakítása után az algoritmus megtalálja a pontok közötti távolságot. A távolság megtalálásának leggyakoribb módja az euklidészi távolság kiszámítása.

2.2.2.3-as ábra: az n-dimenziós euklidészi távolság képlete

## 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 egyértelműen elválasztja a különböző osztályok tagjainak halmazát. Ehhez sokféle hipersík választható. A cél egy olyan sík megtalálása – amelyet a 2.2.3.1-es ábra szemléltet -, amely rendelkezik a maximális margóval, vagyis a maximális távolsággal mindkét osztály adatpontja között.

Az optimális sík megtalálása esetén () egy új pont () osztályozása () már roppant egyszerű, az algoritmus a lineáris regresszió kimenetét vizsgálja. Mivel az SVM a küszöbértékeket 1-re () és -1-re () állítja, megkapjuk azt az értéktartományt ([-1,1]), amely margóként működik. Ha a kimenet nagyobb, mint 1 (), akkor azonosítjuk egy osztállyal (), és ha a kimenet kisebb, mint -1 (), akkor egy másik osztállyal ().

## Gaussian Naive Bayes

A Naiv 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 Naiv Bayes-modellek meglepően jól teljesítettek, még olyan összetett feladatoknál is, ahol egyértelmű, hogy az erős a függetlenségi feltételezések hamisak.

2.2.4.1-es ábra: Naive Bayes tétele

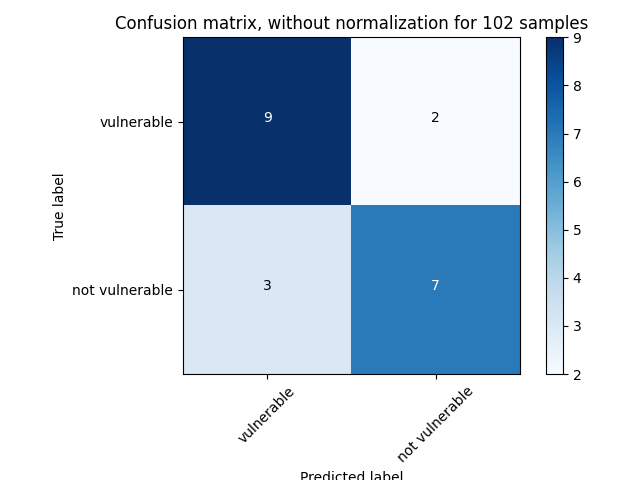
Háromféle Naive Bayes osztályozó létezik, a normál (Gauss-) eloszlás alapján történő osztályozáshoz a Gaussian Naive Bayes-t használják, ekkor Naive Bayes tétele az új adattömb minden szavához () ad egy valószínűségi értéket az adott osztályra () vonatkozóan, amelyek a számítás végén összegzésre kerülnek.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

## 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.

## Confusion Matrix



2.3.1-es ábra: Confusion Matrix 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 helytelenül jósolja meg a pozitív osztályt
* hamis pozitív (false positive, FP): eredmény, ahol a modell helytelenül jósolja meg a negatív osztályt
* hamis negatív (false negative, FN): eredmény, ahol a modell helyesen jósolja meg a negatív osztályt

**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: .

**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 . képlettel történik.

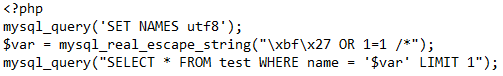
**F-1 score**

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 képlettel történik.

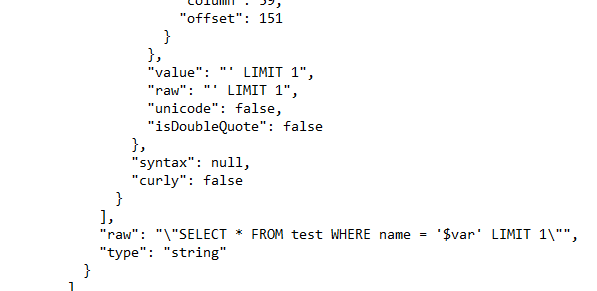
1. PROGRAM FELÉPÍTÉSE

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 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.



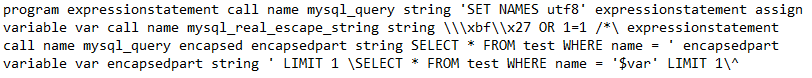
3.2-es ábra: pdo1.php, egy nem sérülékeny kód a tanulo\_adatok mappában

Az 57 darab sérülékeny és 45 darab nem sérülékeny kód legelső lépésben AST generáláson esik át, amihez az irodalomjegyzék [10] elemét használtam. Ez egy manuális folyamat volt, de a parser.js a front-end által bekért kódra ezt már automatizálja az irodalomjegyzék [11] eleme segítségével. A PHP fájlokhoz generált AST fák az AST mappábában közel azonos névvel helyezkednek el .txt kiterjesztéssel.



3.3-es ábra: pdo1\_ast.txt, a 3.1-es ábra kódjának AST megfelelője

A tokenizer.py ezekkel dolgozik tovább. 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 ^.

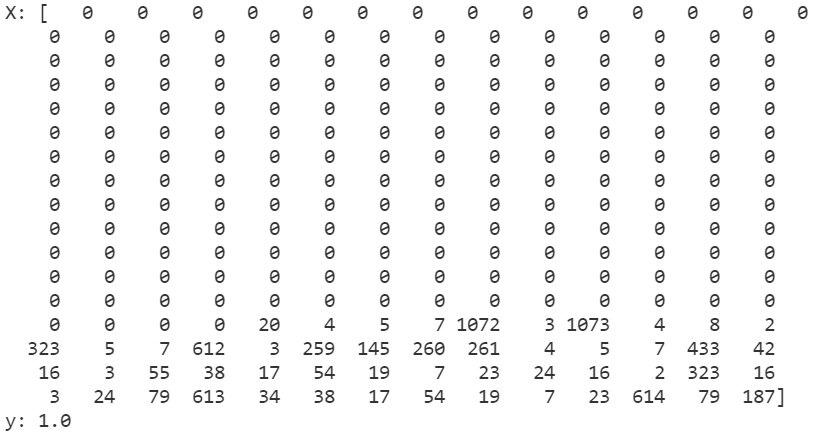


3.4-as ábra: pdo1\_ast.csv, a nodes.py kimenete, a 3.2-es ábra bemenetre

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.



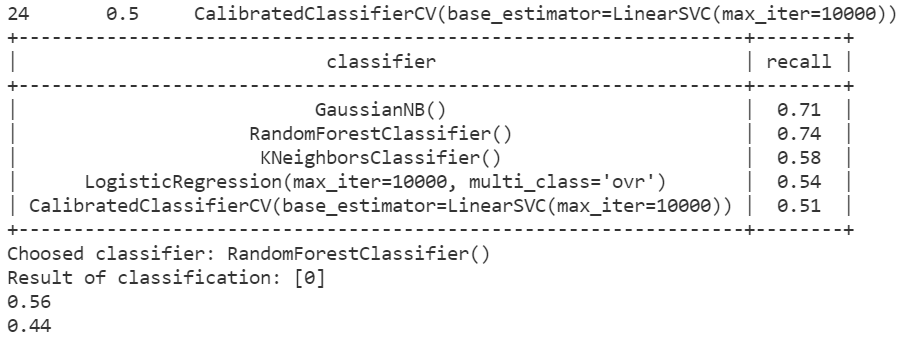
3.5-es ábra: az *X[40]* és a*z y[40]* eredménye; az *X* és *y* 40.eleme az concatenated\_titled.csv 40.sorának felel meg, ami ekvivalens a 3.3-as ábrával felcímkézés után

* 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.



3.6-os 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, a megelőzési módjairól, valamint megjeleníti a korábban megadott kódot és a program futási idejét.

1. EREDMÉNY ÉS ANALÍZIS

## 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 [12] 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 [13] 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.

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

## 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 4.2-es ábra. Az ideális és egyben irreális cél az utóbbiak 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. 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”.

4.2-es ábra: az előrejelzett és tényleges értékek kombinációja 21 teszt adatnál (102 tanuló adat) és 99900 teszt adatnál (100000 tanuló adat)

## Összefoglalás

A programom implementálása után és a felügyelt tanulási modell alapos megismerése alapján már tudom, hogy a gépi tanulás eredményeit több dolog befolyásolja: a tanuló adatok száma, a pozitív-negatív példák aránya, az algoritmus típusa, metrika stb. Ám a végeredmény szemlélteti leginkább, hogy a befolyásoló tényezők mind az intervallum pozitív oldalán vannak.

A dolgozat összefoglalójában leírtak alapján 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.

## Irodalomjegyzék

1. SQL injection

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

1. SQL Injection

https://www.w3schools.com/sql/sql\_injection.asp

1. Error Based SQL Injection (SQLi)

https://beaglesecurity.com/blog/vulnerability/error-based-sql-injection.html

1. Beginner Guide to SQL Injection Boolean Based (Part 2)

https://www.hackingarticles.in/beginner-guide-sql-injection-boolean-based-part-2/

1. Time Based Blind SQL Injection (SQLi)

https://beaglesecurity.com/blog/vulnerability/time-based-blind-sql-injection.html

1. 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/

1. Mike Shema 2012. SQL Injection & Data Store Manipulation

https://www.sciencedirect.com/topics/computer-science/prepared-statement

1. SQL Injection Prevention Cheat Sheet

https://cheatsheetseries.owasp.org/cheatsheets/SQL\_Injection\_Prevention\_Cheat\_Sheet.html

1. Alex Kaechele 2018. Classification

<https://rpubs.com/alexkaechele/380330>

1. AST explorer

https://astexplorer.net/

1. php-parser

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

1. VulnerableApp-php

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

1. How to prevent SQL-injection when using MySQLi

https://stackoverflow.com/questions/60174/how-can-i-prevent-sql-injection-in-php

## Nyilatkozat

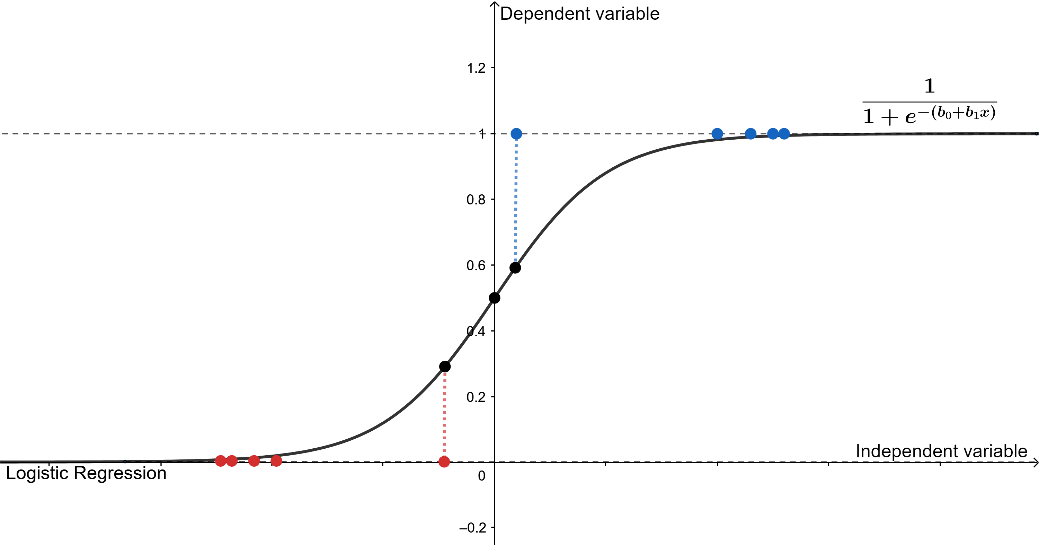
Alulírott Stiller Marianna programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, programtervező informatikus BSc diploma megszerzése érdekében. Kijelentem, hogy a dolgozatot más szakon 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.

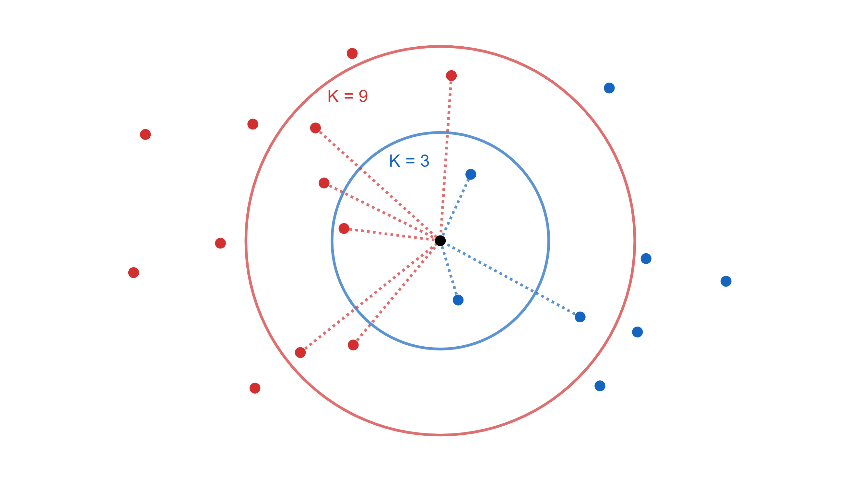
## Köszönetnyilvánítás

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.

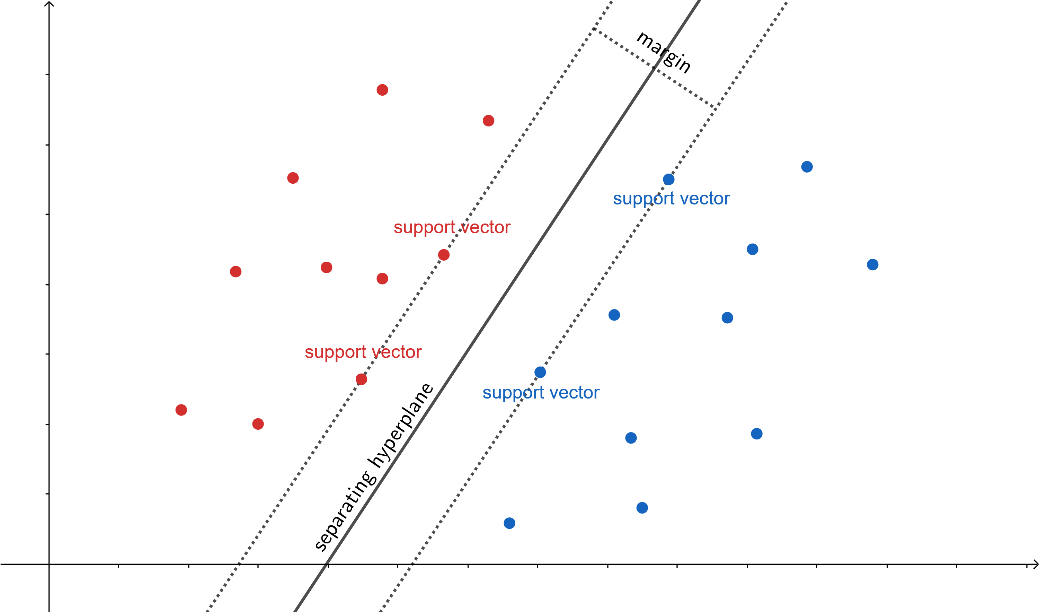
**Mellékletek**



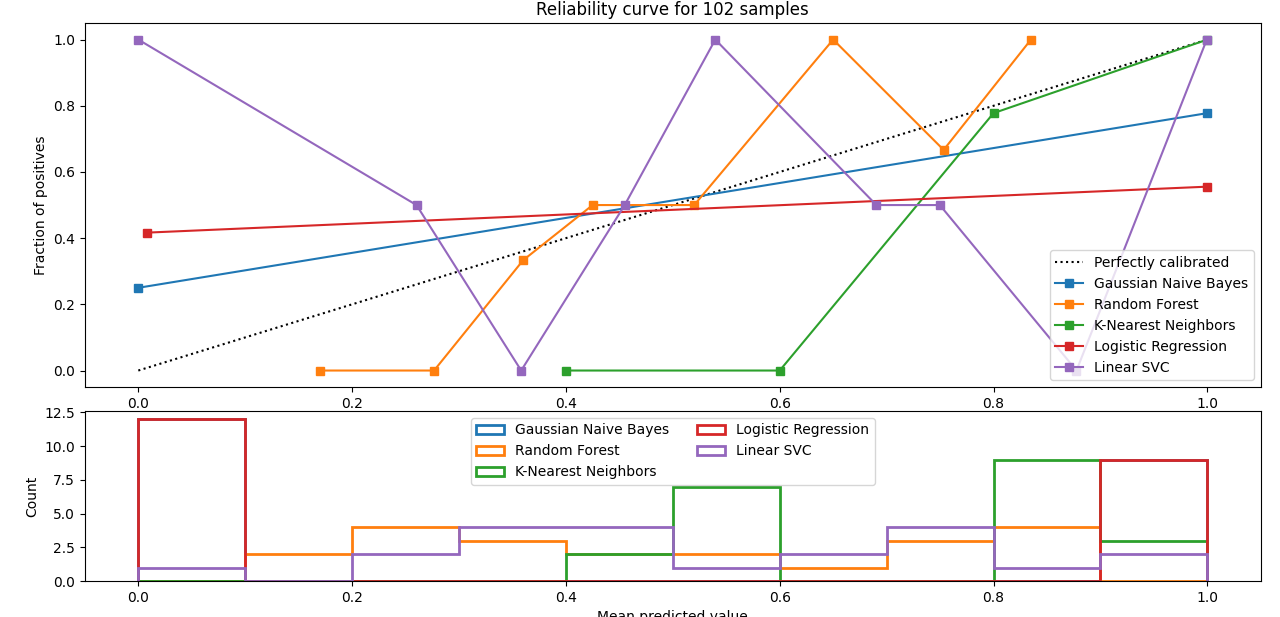
2.2.1.3-as ábra: a Logistic Regression TODO



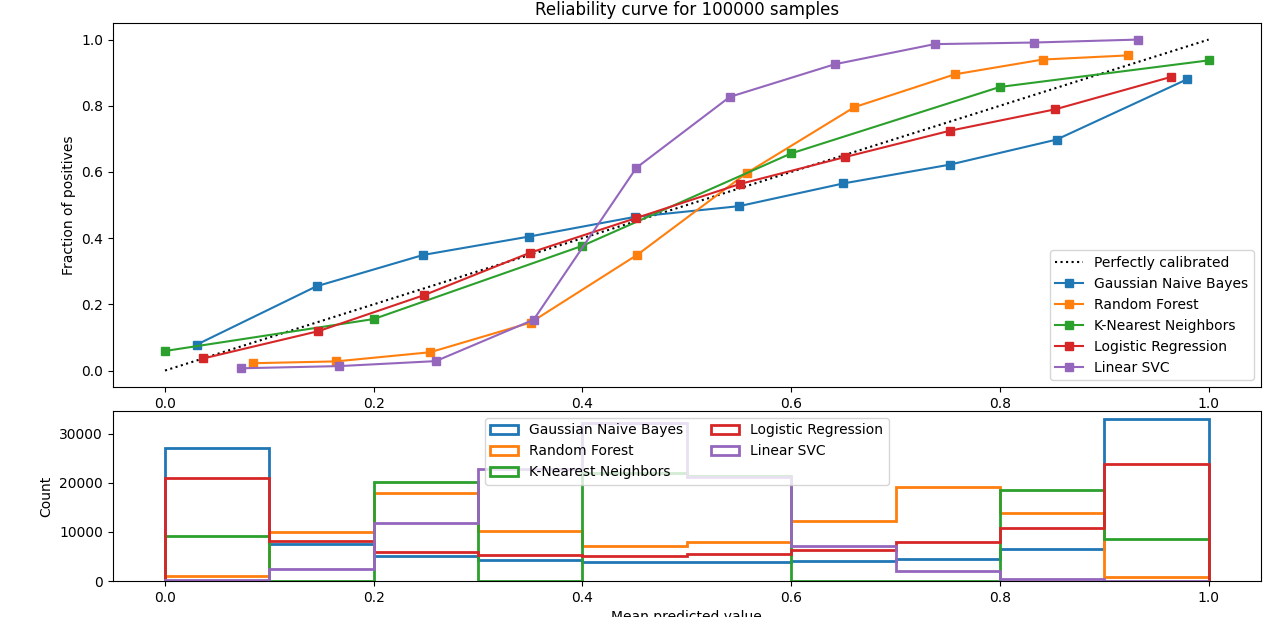
2.2.2.1-es ábra: a K-Nearest Neighbors TODO



2.2.3.1-es ábra: a Linear Support Vector TODO



TODO



TODO