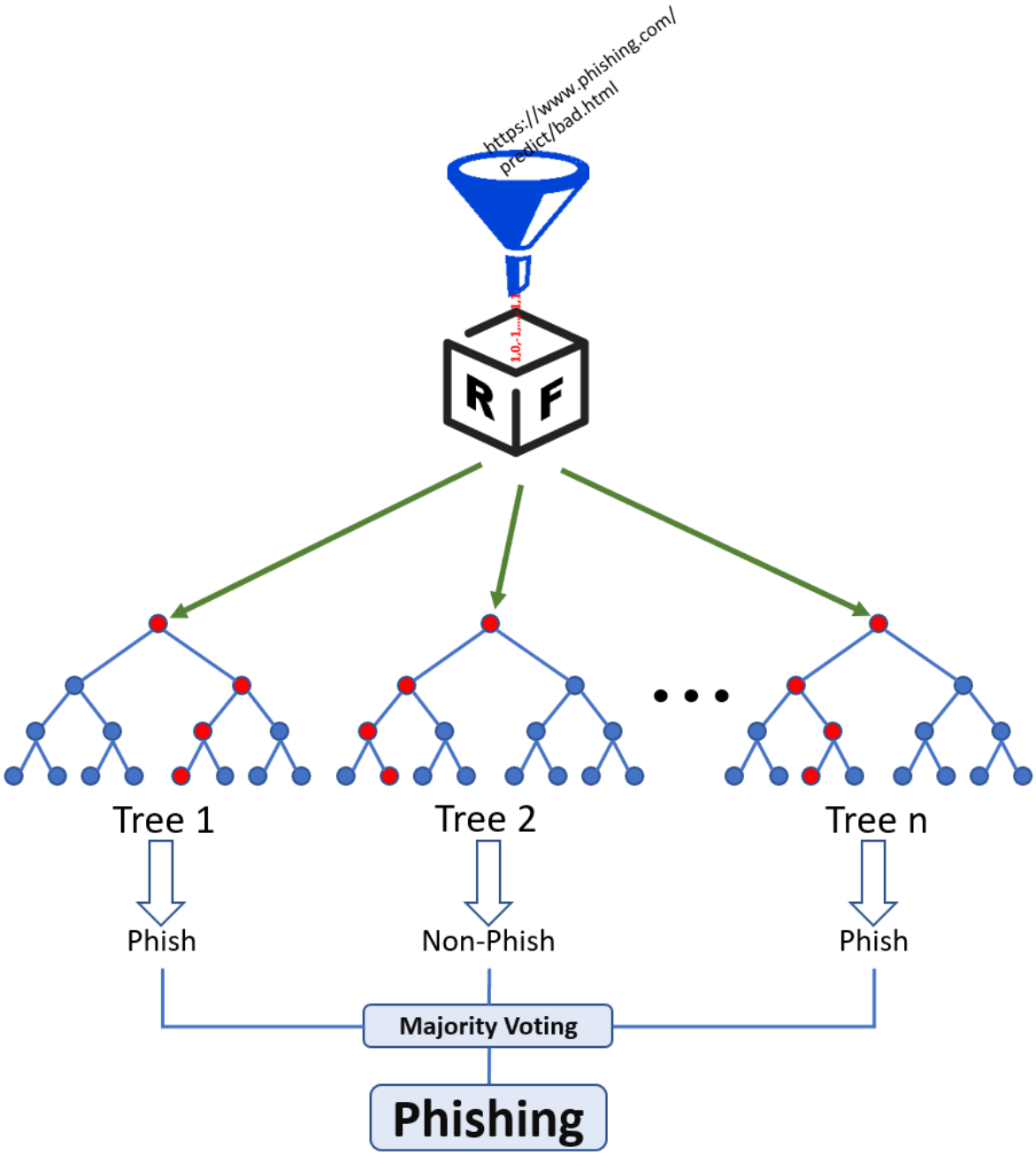
Klassificering af phishing ud fra et URL



Afgangsprojekt på Datamatiker-uddannelsen af Michael Sander Klan

Vejleder: Claus Terp
Uddannelsesinstitution: CPH Business Lyngby
Afleveringsdato: 11. jan. 2021
Eksamineringsdato: 25. - 27. jan. 2021

Abstracts

This report looks at identifying phishing by analyzing attributes of an URL and the underlying webpage. I have used the results of these to create a classification model in machine learning using Random Forest. This is developed in Python and utilizes Pythons good support of machine learning and access to external resources. Finally, it showcases a concept for using the developed modules to prevent accidental accessing phishing sites.

Forord

Denne rapport er udført som et 30 ECTS speciale for Datamatikere ved CPH Business Lyngby. Deltager i gruppen er Michael Sander Klan. Det primære arbejde blev udført i projektperioden nov. - jan i eget hjem, men ide til opgaven blev skabt ud fra praktikforløbet i virksomheden Statens IT i afdelingen DART (Detection And Response Team), ud fra den opgave, jeg blev stillet dér.

Noget af mit arbejde i praktikperioden har været forarbejde for at kunne lave denne opgave; I praktikperioden har jeg gennemgået indrapporterede mails fra brugere, og trukket links ud, der skulle analyseres med henblik på klassifikation omkring de var phishing eller ej. Jeg har i praktikperioden også læst op på karakteristika omkring phishing og påbegyndt på 6 målepunkter, der kunne bruges videre i denne opgave. Det er: 1. Using the IP Address, 2. Long URL to Hide the Suspicious Part, 3. Using URL Shortening Services “TinyURL”, 4. URL’s having “@” Symbol, 5. Redirecting using “//”, 6. Adding Prefix or Suffix Separated by (-) to the Domain) Se i (se referenceliste 18.1).

I forbindelse med denne opgave er der en række personer, jeg gerne vil takke for deres hjælp;

Jeg vil gerne takke mine kontaktpersoner i Staten IT for mit praktikforløb og for at vise mig, at der er behov for at identificere phishing, og at phishing-angreb i virksomheden stadig er meget aktuelt.

Jeg vil også gerne takke min vejleder Claus Terp for god feedback og konstruktive forslag, som har hjulpet mig i min videre proces, og for at fastholde mig i strukturen i min opgave

Sidst, men ikke mindst vil jeg gerne takke min forældre for deres tid til at afse ugentlige samtaler og sparring med mig, for at holde min motivation oppe og takke for brug af deres hjem som arbejdskontor.

Jeg forudsætter, at læseren af rapporten har kendskab til, hvad der svarer til mindst et fjerde-semester-niveau på datamatikeruddannelsen

Indholdsfortegnelse

1. Introduktion til projektet 1
   1.1. Målgruppe 1
2. Introduktion til virksomheden 2
   2.1. Statens IT 2
   2.2. Problemstilling 2
3. Opgaven 2
   3.1. Problemformulering 2
   3.2. Hypotese 3
4. Fordele ved phishing-program for virksomheden 3
5. Kravspecifikation 3
   5.1. Funktionelle krav 3
   5.2. Ikke-funktionelle krav 4
   5.3. Afgrænsning af krav 4
6. Valg af teknologi 4
   6.1. Python 4
   6.2. Python-moduler anvendt i projektet 5
   6.3. Machine learning 5
   6.4. Valg af klassificeringsmodel 5
   6.5. Fordele ved Random Forest som klassificeringsmodel 9
   6.6. Ulemper ved Random Forest som klassificeringsmodel 9
7. Valg af arkitektur 9
   7.1. Moduler 9
   7.2. Flows 10
8. Design 13
9. Sikkerhedsaspekter 13
10. Implementering 13
   10.1. Om målepunkter 13
   10.2. Beskrivelse af 30 målepunkter 14
      10.2.1 Address Bar based Features (1-12) 14
      10.2.2 Abnormal Based Features (13-18) 15
      10.2.3 HTML and JavaScript based Features (19-23) 16
      10.2.4 Domain based Features (24-30) 17
   10.3. Beskrivelse af kode 17

10.3.1 Favicon 17
10.3.2 Request_URL 18
10.3.3 Valg af klassificeringsmodel 20
10.3.4 Koncept eksempel på anvendelse 24
11. Test 25
11.1 GetPhishingFeatures testet med non-phishing URL: 26
11.2 GetPhishingFeatures testet med phishing URL 26
11.3 TrainClassifier testet med phishing URL 27
12. Evaluering og konklusion 28
12.1. Evaluering 28
12.2. Konklusion 28
12.3. Uhensigtsmæssigheder 28
12.4. Videreudviklingsmuligheder 29
13. Projektets set-up 30
14. Valg og tilpasning af udviklingsmetode 31
14.1. Proces omkring projektets mål 31
14.2. Ændring af udviklingsmetode 31
14.3. Scrum 31
14.4. Andre metoder 31
15. Planlægning og overblik over tidsforløb 32
15.1. Første tidsplan 32
15.2. Ændret tidsplan 32
16. Dokumentation af og refleksion over proces 33
16.1. Overordnet om sprints og tidsestimering: 33
16.2. Beskrivelse af indhold i sprints: 33
16.2.1. Sprint 1: 33
16.2.2. Sprint 2: 33
16.2.3. Sprint 3: 34
16.2.4. Sprint 4 (varede ca. 2 uger): 34
16.2.5. Sprint 5: 34
16.2.6. Sprint 6 (varede 1½ uge): 35
16.2.7. Sprint 7: 35
16.2.8. Sprint 8: (sidste sprint) 35
17. Refleksioner over proces 36
17.1. Det dårlige ved processen 36
17.2. Det gode ved processen 36

17.3. Konklusion på proces 37
18. Referencer 38

1.

1. Introduktion til projektet

Phishing er en form for cyberangreb, hvor svindlere prøver at lokke fortrolige oplysninger ud af folk; Det kan f.eks. være en anmodning fra deres bank eller en note fra personer i deres virksomhed - og at klikke på et link eller downloade en vedhæftet fil.
Formålet med phishing er at franarre modtageren oplysninger, der kan bruges til at opnå økonomisk gevinst - f.eks. kontooplysninger, pengeoverførsler og ransomware.
Det er et af de ældste typer cyberangreb, der går tilbage til 1990'erne, og det er stadig en af de mest udbredte og skadelige, hvor phishing-meddelelser og -teknikker bliver stadig mere sofistikerede.
Det, der gør phishing så effektivt, er efterligningen af e-mails fra pålidelige kilder; Angriberne får modtageren til at tro, at henvendelsen kommer fra en pålidelig afsender, som modtageren kender eller har handlet med. Dette gør, at det kan ofte være svært at gennemskue forskellen på en legitim og en phishing e-mail, da man normalt ikke tager sig tid til at nærlæse alting hvis e-mailen antages at komme fra en person eller et firma, man kender og har tillid til. Man søger derfor at udvikle programmer, der kan håndtere og undersøge mails for eventuelle phishing angreb for at stoppe brud på it-sikkerheden.

Hvordan kan man så beskytte sig mod disse phishing forsøg? Jeg vil i dette projekt kigge på, hvordan man kan gøre opmærksom på et eventuelt phishing forsøg, udarbejdet ud fra en problemstilling fra mit praktikophold.
Ved en række konkrete krav til mit program finder jeg frem til relevante teknologier og arkitektur, hvorved jeg laver et program, der udnytter de karakteristika, som links i phishing e-mailene har og forsøge at kunne klassificere dem udelukkende ved disse.Efter forklaring af udvalgte kode-dele gennemgår jeg test af programmet og evaluerer på mit produkt, også med overvejelser omkring videreudvikling.
Dernæst gennemgår jeg min proces gennem projektet, med beskrivelse af dets set-up, mine arbejdsmetoder og planlægning. Dette dokumenteres ved min forløb i sprints, hvor jeg afslutter med at evaluere min proces. Til sidst følger en liste over de referencer, jeg har brugt gennem forløbet.

1.1. Målgruppe

Denne opgave er rettet til folk, der arbejder med it-sikkerhed og typisk indenfor mail, som er interesserede i at forstå, hvordan man kan begrænse phishing-angreb på et tidligt tidspunkt. Jeg er selv interesseret i IT - sikkerhed og for mig har det været vigtigt, at jeg i min hovedopgave beskæftiger mig med et emne, som jeg fremadrettet kunne tænke mig at arbejde med. Phishing er stadig et meget aktuelt emne, og vil fortsat være det i mange år fremover. Mit håb er, at folk kan bruge mit implementerbare program som inspiration og afsæt til et mere udviklet phishing-system og dermed til en bredere målgruppe.

PRODUKT - afsnit

2. Introduktion til virksomheden

2.1. Statens IT

Statens IT er den danske stats it-driftsafdeling, organiseret som en styrelse under Finansministeriet. De leverer it-services til 16 ministerområder og 4 andre større statslige institutioner - svarende til ca. 28.000 medarbejdere i staten. Dette enorme bagland og den nødvendig grad af fortrolighed pga. statsligt arbejde fordrer en høj grad af it -sikkerhed. Afdelingen DART,hvor jeg var i praktik, tager sig fra de mere komplicerede IT- sager, oftest omkring it-sikkerhed.

2.2. Problemstilling

Statens IT har oprettet en Phishing knap i Outlook, hvor brugerne kan rapportere om potentielle phishing mails og links, hvor man også modtager den originale mail. Det er dog store mængder data, der rapporteres, og det er derfor tidskrævende at gennemgå manuelt. Overordnet ønsker virksomheden derfor en proces bestående af nogle moduler, der kan lette arbejdsgangen for den it-medarbejder, der skal håndtere og behandle indrapporterede formodede Phishing e-mails

3. Opgaven

3.1. Problemformulering

Problemstillingen favner bredt og kan omfatte flere specifikke egenskaber omkring phishing. For at opnå et færdigt og implementerbart program indenfor projektperiodens tidsramme har jeg valgt at fokusere på linket, uanset i hvilken sammenhæng det optræder, og kigge på sandsynligheden for phishing.

Min problemformulering for min projektopgave lyder således:

Ud fra et givent URL, kan man sandsynliggøre om et link er phishing og med hvor stor sandsynlighed?

Herunder har jeg lavet følgende arbejdsspørgsmål for at arbejde målrettet med min problemformulering:

Hvilke checks og metoder vil være relevante for at kunne identificere phishing?

- Er der specifikke kendetegn ved phishing URL'er, der kan udpege det som et phishing link?
- Kan man sammenligne et givent link med et register af kendte phishing links, for at finde sandsynligheden for om det er phishing?

3.2. Hypotese

Identifikation af de links, der som regel modtages som forsøg på phishing, vil være med til at begrænse antallet af succesfulde angreb.
Da der ikke er en enkel måde at entydigt identificere disse farlige links, er det min ide, ud fra forskellige karakteristika ved phishing-links, at kunne sandsynliggøre et Phishing forsøg.
Da hver af disse egenskaber i sig selv er ikke nok til at afgøre, om et URL er phishing eller ej, vil netop kombinationen give en bedre chance for at identificere og opfange phishing.
Udgangspunktet må være, at jo flere egenskaber man tester på, desto større nøjagtighed i klassificering vil man kunne opnå.
Målet er at komme med det bedste bud på, om det er phishing og samtidig give en indikation på, hvor sandsynligt dette resultat er.

4. Fordele ved phishing-program for virksomheden

En generelt mere effektiv arbejdsgang omkring håndtering af phishing e-mails kan give både arbejdsmæssige og økonomiske fordele for virksomheder.
Ved at implementere programmet i dette projekt kan man erstatte store dele af den manuelle identificering, som foregår i dag i virksomheder som f.eks. Statens IT. Da brugerne aktivt selv skal identificere og markere et aktivt phishingangreb, kan man i mange tilfælde stoppe et fejltrin omkring phishing link tidligt i forløbet. Man kan dermed eliminere et trin i processen og frigøre værdifuld arbejdstid til andre opgaver.
Man kan i dag tilkøbe systemer, hvor der typisk betales pr. bruger for analyse af links, om det er phishing eller ej. Ved at lade dette system indgå kan der opnås stor økonomisk gevinst for virksomhederne. For en it-medarbejder, der normalt skulle undersøge disse phishing-forsøg, vil arbejdsmængden også blive væsentligt begrænset.

5. Kravspecifikation

Krav til systemet er afgørende, da det hjælper med et solidt udgangspunkt for det videre projekt. Opstillede krav hjælper med at finde frem til de nødvendige opgaver, og med udgangspunkt i min problemformulering har jeg opstillet følgende krav:

5.1. Funktionelle krav

Systemet skal kunne -

- hive målepunkter ud fra et givent URL
- læse en liste URL'er og lave et datasæt af målepunkter
- læse disse målepunkter og danne et machine learnings - træningssæt til klassificering af phishing eller ej
- tage en vilkårlig URL og svare tilbage omkring phishing eller ej, og event. give liste af målepunkter, der indikerer om det er en phishing eller ej

5.2. Ikke-funktionelle krav

Da systemet skal kunne bruges i et realtids-flow, er det afgørende for anvendeligheden af system, at det ikke tager for lang tid at køre. Performance er et typisk ikke-funktionelt krav. Jeg holdt mig til disse ikke-funktionelle krav:

- Valg af machine learning for at finde metode, der giver største træfsikkerhed
- Optimering af den valgte metode

5.3. Afgrænsning af krav

Det er nødvendigt fra starten af at sætte nogle rammer for et projekt, særlig med begrænset tidshorisont. En af mine oprindelige ideer var at anvende sandboxing, da det er en måde, hvorpå man kan undersøge om et site opfører sig sikkert. Efter at have undersøgt denne mulighed var det for omfattende både tidsmæssigt og i forhold til mine kompetencer, så det valgte jeg fra. Det samme gælder en ide, jeg havde med at lave en browser-ekstension, men mit valgte koncept har en bredere anvendelse og ses anvendt af professionelle løsninger.

6. Valg af teknologi

I dette afsnit kigger jeg på, hvilke teknologier jeg har anvendt for at lave systemet. Jeg kommer ind på de forskellige komponenter der indgår, samt hvordan jeg har valgt at opdele systemet i moduler. Der forventes at læseren har et vist kendskab til de delkomponenter, som benyttes. Jeg gennemgår desuden, hvordan jeg er kommet frem til de valg, der er gjort og hvorfor.

6.1. Python

Virksomheden havde ingen præferencer eller ønsker i forhold til programmeringssprog, og jeg var derfor frit stillet til at vælge system. Mit ønske til læringen var, at jeg gerne ville programmere i Python. Jeg synes, at Python er nemmere at læse og generelt mere tilgængeligt, og så er det godt til at manipulere større mængder data.

Dette projekt vil arbejde med datasæt af en vis størrelse og analysere og bearbejde disse, derfor faldt valget naturligt på Python. Dette kombineret med at Python har en ret nem måde at benytte sig af eksterne services, mange af disse som er indkapslet i moduler. Dette er også tilfældet for moduler til Machine Learning, hvor klassificering var tiltænkt som værende det, der skal give en smartere tilgang til løsningen af hovedformålet med projektet.

For Python programmører er mange af disse standardmoduler noget, der kendes til, og jeg benytter mig af følgende: **numpy**, **pandas**, **datetime**, **re**, **argparse**, og **sys**. Derfor vil jeg ikke beskrive disse nærmere.

Jeg har valgt at bruge VScode som min IDE, fordi jeg har en præference for denne, og den har en god integration til Python.

6.2. Python-moduler anvendt i projektet

Efterfølgende moduler benyttes også, og jeg har uddybet min brug af disse i følgende:

requests – benyttes til at hente siden, som et URL refererer til.

beautifulsoup4 – benyttes til at analysere HTML'en og dets elementer.

googlesearch (**search**) – search er en del et større bibliotek, hvor jeg kun benytter mig af denne til at forespørge Google Index.

OpenSSL og **ssl** – benyttes til at tjekke certifikater for et givent URL.

whois – benytter til at hente data om domænet i URL'et fra DNS-databaserne.

Til brug for klassificeringen, klargøring, og test af data benyttes flere moduler fra biblioteket scikit-learn.

Der er gode funktioner til at opdele data i træningssæt og testsæt (**train_test_split**), inden jeg bygger og tester de forskellige modeller til at finde den bedste. Til dette benytter jeg **accuracy_score**.

Jeg har gjort brug af følgende klassificeringsmodeller: **tree**, **svm**, **RandomForestClassifier**, **GradientBoostingClassifier**, og **KNeighborsClassifier**.

Til sidst benyttes **pickle** til at gemme og gen-indlæse valgte klassificeringsmodeller.

6.3. Machine learning

Machine Learning dækker over mange forskellige aspekter af dataanalyse. Det er oplagt at dén gren af Machine Learning, som vi skal bruge, er en såkaldt klassificeringsmodel med udfald af phishing/non-phishing.
Bestemmelse af om det er phishing kan ikke udelukkende gøres ved enkelte eller få målepunkter; Men via en klassificeringsmodel, der tager højde for *alle* de målepunkter der er data i. kan systemet komme med godt estimat og samtidigt give et estimat på, hvor godt det bud er.
Styrken ved disse modeller er, at man ved et begrænset udvalg af målepunkter kan komme frem til et resultat, der med stor sandsynlig er korrekt, og at modellerne kan trods manglende data stadig benyttes.

6.4. Valg af klassificeringsmodel

Jeg har valgt at kigge på 10 forskellige klassificeringsmodeller for at vælge den model, der passer bedst til opgaven.

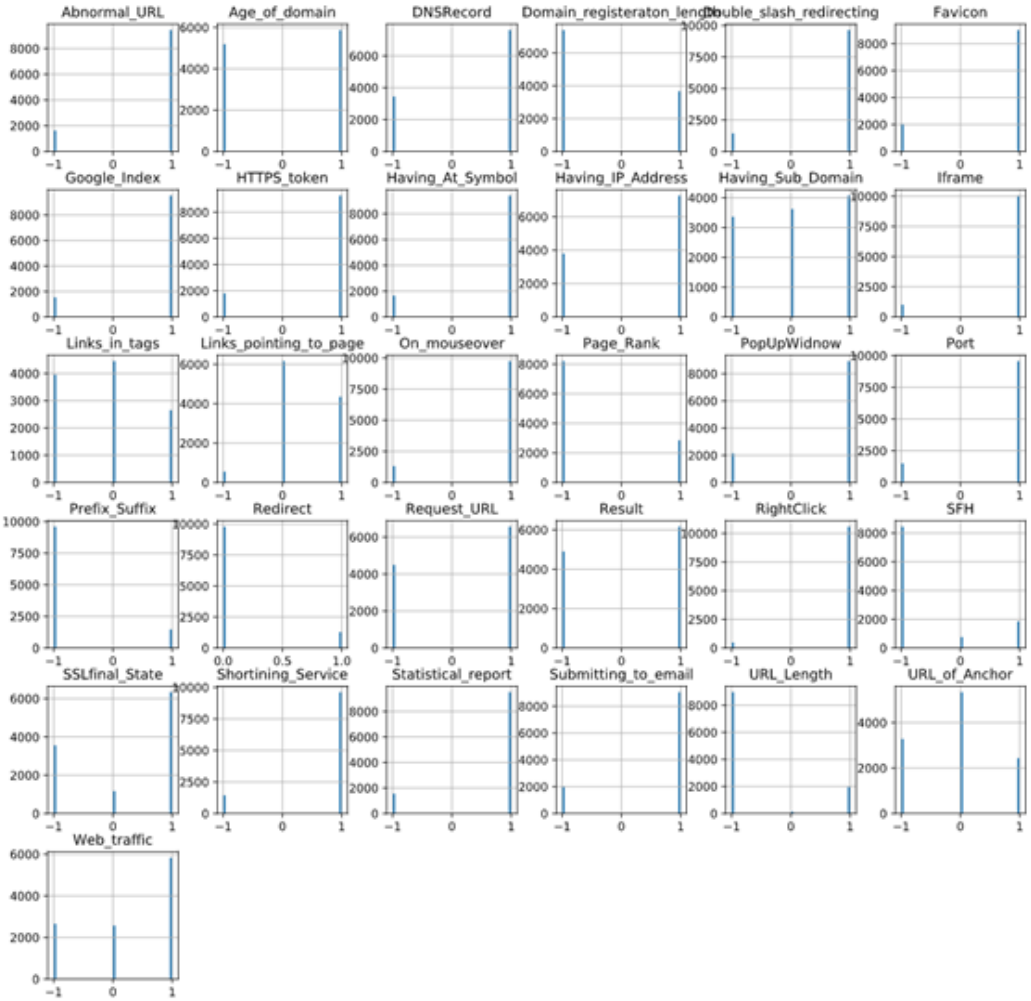
Næsten alle phishing målepunkterne har et binært svar "phishing" eller "legitim", enkelte har også "mistænkelig".

I min søgen efter lister over phishing sites, er jeg faldet over et datasæt (se referenceliste 18.1) som har de samme målepunkter, som jeg har valgt at have. Jeg har derfor brugt dette datasæt i min analyse af, hvilken klassificeringsmodel der passer bedst. Datasættet har desuden med sine 11.055 rækker en pæn blanding af phishing og non-phishing rækker, men desværre ikke hvilket URL, der er tale om, så jeg har ikke kunnet benytte dette datasæt til at teste målepunkterne.

Jeg startede med at lave en Jupyter Notebook, hvor jeg indlæste datasættet og så på hvad det indeholdt. Efter at have sat lidt overskrifter på målepunkterne, kunne jeg se hvordan de enkelt målepunkter fordelte den 11.055 rækker (se nedenstående figur 1).

Fordelingen af data-rækkerne er phishing: 4.898 og non-phishing: 6.157, dvs "næsten" ligeligt fordelt. Det fremgår også af **Result**-grafen. Det kan være værd at bemærker at måle punkter som **Prefix_Suffix** og **URL_Length** giver mange falske positive udslag og målepunkter som **port**, **iframe**, og **doubleslash** giver lavt udslag for phishing.
Desuden er det interessant at se, at der på målepunktet **Redirect** ikke er noget der markeres som phishing, kun suspicious og non-phishing.

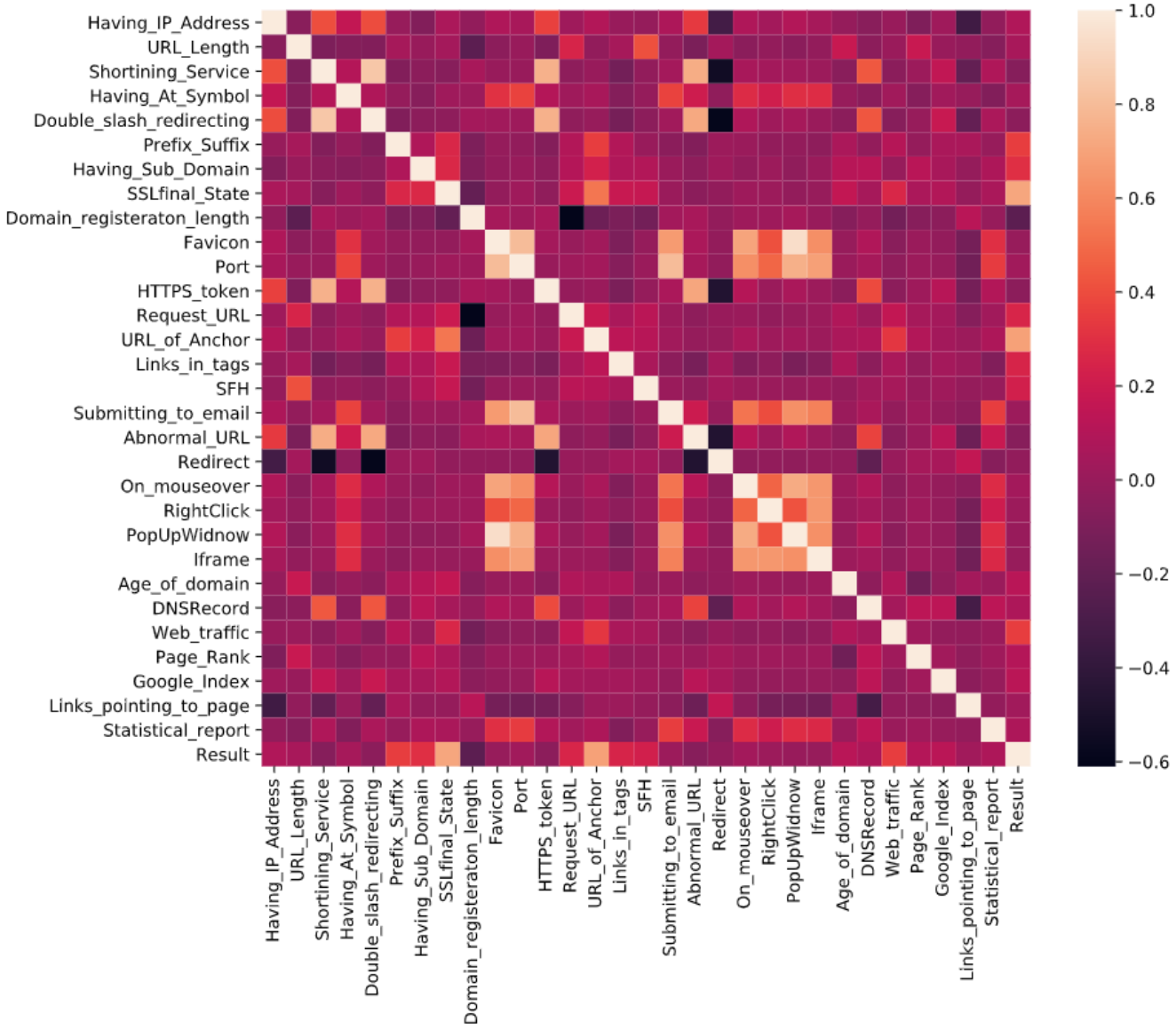
Disse afvigelser kan evt. betyde at målepunkterne ikke bliver ofte brugt af phishere, eller også skal målepunkterne være mere nuanceret.



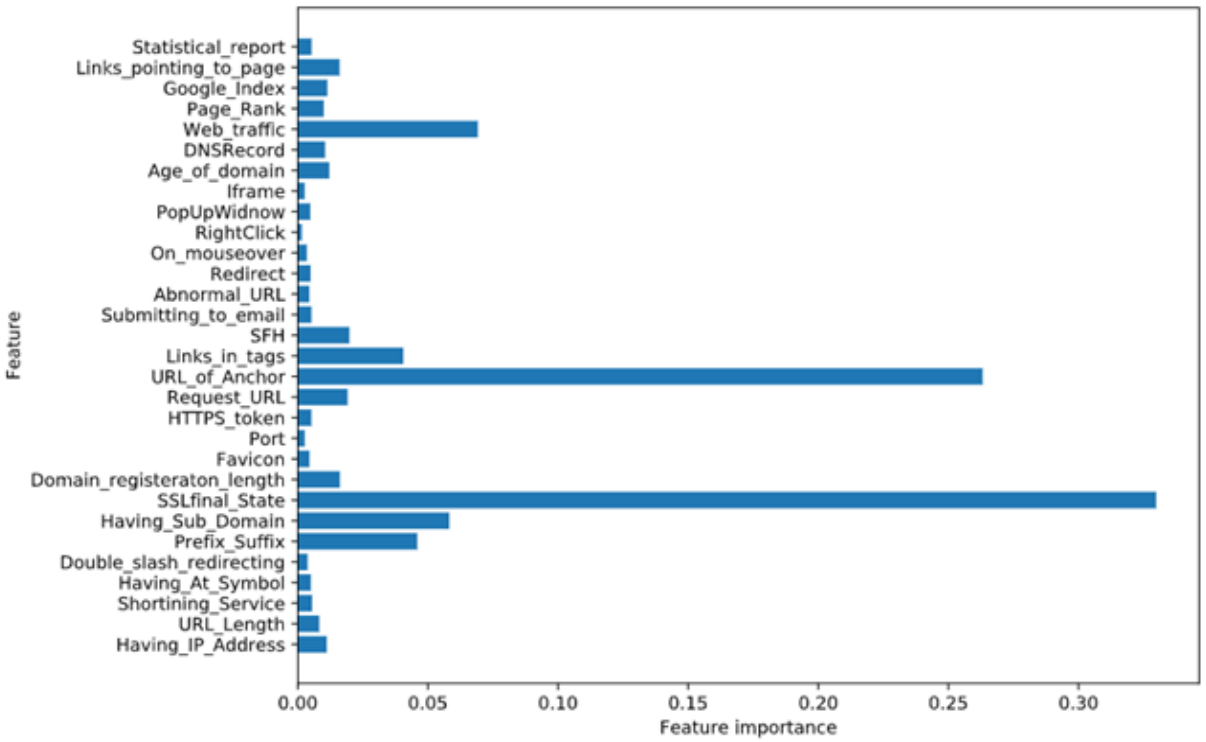
Figur 1 Hyppighed af målepunkter

For at prøve at forstå om der var sammenhænge mellem de forskellige egenskaber, prøvede jeg at plotte et heatmap over sammenhængene mellem de forskellige egenskaber (se nedenstående figur 2). Selvom der er enkelte, som fremtræder samtidigt, synes jeg ikke at der var noget nævneværdig grund til, at det skulle være en trend.

Når jeg ser på den nederste linje, **Result**, kan det se at 2 af egenskaberne er lidt mere fremtrædende. Dette kan også ses i næste figur (se figur 3).



Figur 2 Heatmap af de forskellige målepunkter i forhold til hinanden



Figur 3 Signifikans af de forskellige målepunkter

Dernæst prøvede jeg 10 forskellige klassificeringsmodeller for at finde den bedste, der passede til opgaven – både mht. resultat (Accuracy), processeringstid (at danne modellen), og tid den var om at forudsige et resultat (PredictTime).

Det gav følgende resultat:

> **ProcTime:2.43 PredictTime:0.26 - Random forest: Accuracy on test data: 0.968**
> ProcTime:0.02 PredictTime:0.00 - Decision tree: Accuracy on test data: 0.963
> ProcTime:0.98 PredictTime:0.15 - SVC with linear kernel: Accuracy on test data: 0.927
> ProcTime:0.96 PredictTime:0.24 - SVC with rbf kernel: Accuracy on test data: 0.945
> ProcTime:0.92 PredictTime:1.29 - Custom SVC with ovo linear: Accuracy on test data: 0.927
> ProcTime:0.77 PredictTime:0.23 - Custom SVC with ovo rbf: Accuracy on test data: 0.945
> ProcTime:2.66 PredictTime:0.58 - NuSVC: Accuracy on test data: 0.913
> ProcTime:2.35 PredictTime:0.56 - One Class SVM: Accuracy on test data: 0.474
> ProcTime:0.06 PredictTime:1.07 - K nearest neighbours: Accuracy on test data: 0.937
> ProcTime:0.81 PredictTime:0.01 - GradientBoostingClassifier: Accuracy on test data: 0.947

Her er det tydeligt, at Random Forest (RF) og Decision Tree (DT) er de 2 modeller, der giver den mest nøjagtige forudsigelse. Desuden er DT virkelig hurtig, når det gælder forudsigelsen, så det vil (antageligvis) være naturligt at vælge den.

De 2 modeller er meget lig hinanden: Begge bygger på beslutningstræer, hvor DT er en simpel træstruktur (og derved lettere at forstå) og RF kombinerer flere beslutningstræer (som er tilfældigt udvalgt – heraf navnet). Men der er en ulempe ved DT, hvor den har en tendens til at blive noget mere unøjagtig ved datasæt, som afviger meget fra træningsdatasættet. Dette håndterer RF bedre, så med mindre at PredictTime er vigtig for brug af min phishing-detection-rutine, er det bedre at vælge RF.

6.5. Fordele ved Random Forest som klassificeringsmodel

Denne klassificeringsmodel kan løse to typer af problemer, klassificering og regression, og giver gode resultater på begge fronter.

En af fordelene ved Random Forest, som jeg finder meget anvendelig, er evnen til at håndtere store datasæt med mange målepunkter. Den kan håndtere tusindvis af målepunkter og identificere de mest betydningsfulde af dem, så man kan betragte det som en måde at reducere antallet af målepunkter. Desuden kan modellen vise vigtigheden af de enkelte målepunkter, hvilket kan være en meget praktisk ved analyse af f.eks. tilfældige datasæt.

Den har en effektiv metode til estimering af manglende data på, samtidigt med at den bibeholder nøjagtighed i dens resultat, når en stor del af dataene mangler.

Den har metoder til at afbalancere fejl i datasæt, hvor klasserne er ubalancerede.

Ovenstående egenskaber kan også benyttes til uklassificerede data, hvilket fører til ikke-supervised klyngedannelse, datavisninger og detektion af rand-data.

6.6. Ulemper ved Random Forest som klassificeringsmodel

Modellen giver gode og nøjagtige resultater ved klassificering, men er knap så god til regressionsproblemer.

Random Forest kan føles som en black box-tilgang for en med en statistisk tilgang - man har meget lidt kontrol over, hvad modellen gør. Man kan i bedste fald prøve forskellige parametre og tilfældige random-seeds.

7. Valg af arkitektur

Jeg har valgt en forholdsvis simpel strukturel opbygning af mit system, som ikke involverer flere maskiner eller databaser. Her kommer jeg ind på de moduler, systemet består af, og de flows der er mellem dem.

7.1. Moduler

Arkitekturen består af 3 Python programmoduler, som udgør de 3 flows, der skal til for at kunne tilbyde en vurdering af, om det er phishing.

Det første modul er GetPhishingFeatures.py, hvis primære opgave er at danne et resultatet for hvert målepunkt i en form, som er tilpasset den klassificeringsmodel jeg har valgt at benytte. Dette resultat gemmes i en tekstfil, som skal bruges af næste modul. Dette er tænkt at skulle køres som en enkeltstående kørsel uafhængig af resten af systemet, som kan køres igen og igen som man får fat i de nyere og bedre phishing lister. Der er desuden test-funktionalitet indbygget i modulet.

Det andet modul, TrainingClassifier.py, kan også køres uafhængigt af resten af systemet, og har til formål at indlæse den resulterende fil, som første modul dannede, og danne en Random Forest klassificeringsmodel og gemme denne igen i en fil til brug for sidste modul. Der er også her test-funktionalitet indbygget i modulet.

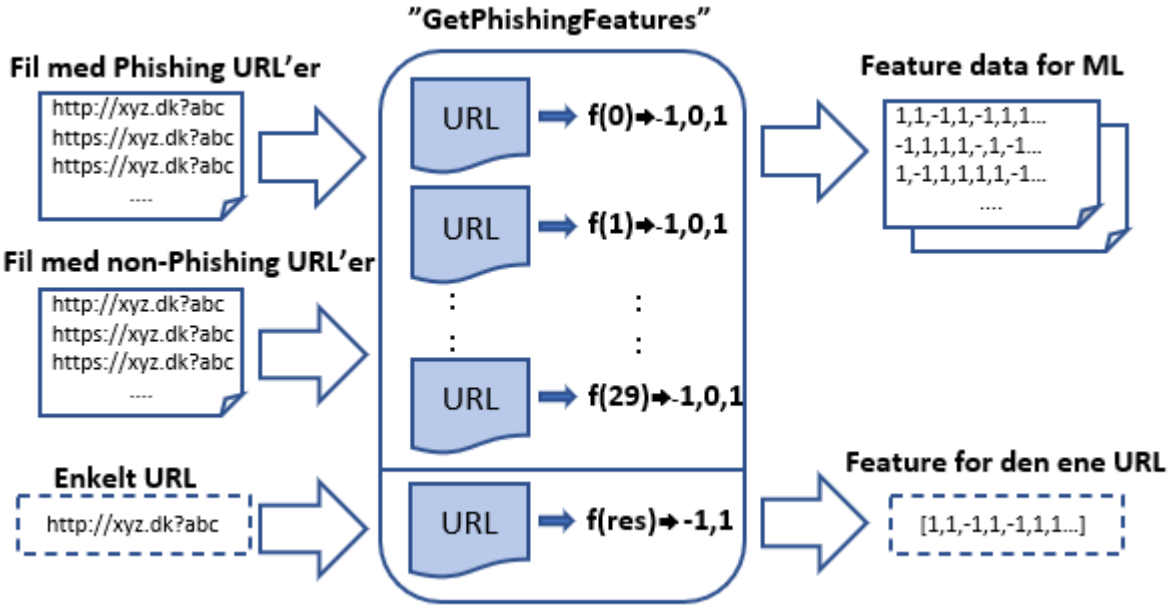
Det tredje og sidste modul, api-mini.py, er tiltænkt som et koncept, der viser hvordan man kan lave en service, som kan tilbyde andre systemer såsom e-mail servere, web-servere, mm. en mulighed for at teste og blokere for et phishing link. Det er noget, man typisk ser i professionelle løsninger, hvor man f.eks. får sat sit mailsystem op til at pakke alle links ind i et kald til denne service. så et link til f.eks. https://www.dr.dk bliver erstattet af et link som https://tjekforphishing.dk/?url=https%3A%2F%2Fwww.dr.dk. Resultatet af dette er, at hvis linket ikke bliver klassificeret som phishing, omdirigeres man automatisk til det oprindelige site, ellers bliver man gjort opmærksom på, at man har klikket på et phishing-link.

Da jeg primært har valgt at fokusere på at løse problemstillingen med en simpel og effektiv struktur, og at der på dette stadie ikke er krav til at skulle indgå i et større eksisterende systemkompleks, giver det ikke mening at kigge på alternative opbygninger. Havde der været eksterne krav, ville de sikkert stille krav til både valg af teknologi og arkitektur.

7.2. Flows

Denne opgaves system kan opdeles i 4 flows:

1. Dannelse af målepunktsdata ud fra kendte lister af URL'er med phishing og non-phishing data, og samle disse i en fil til træning af klassificeringsmodellen.
2. Indlæsning af denne fil med målepunktsdata, foretag træning af RF modellen for derefter at gemme modellen i en fil til brug i det sidste flow.
3. Indlæsning af den færdigtrænede model og derefter indlæse et enkelt URL eller en fil af URL'er til at forudsige, om de er phishing. Ved input fra en fil lægges resultatet i en fil.
4. En hjemmeside, der kan tage et URL, som returnerer om det er phishing eller omdirigerer til angivne side.



Figur 4 Flow af GetPhishingFeatures

Det første modul tager 2 forskellige parametre og kan kaldes på 2 måder.
Den første måde kaldes modulet med en input fil som parameter sammen med værdien for om det er phishing eller ej.

python GetPhishingFeatures.py -i [filenavn.ext] [verdict]

[filenavn.ext] er navnet på inputfilen og [verdict] er værdien 1 for non-phishing og værdien -1 for phishing. Filen skal indeholde de URL'er, man vil have værdierne for målepunkterne, en URL pr. linje. Disse URL'er læses ind én ad gangen og testes for phishing egenskaber, og outputtet fra modulet skrives ned i en fil, der får navnet [filnavn].out. Sidste værdi i output linjen er denne værdi [verdict].

Den anden måde er at kalde modulet med en enkelt URL, hvor output skrives ud på terminalen.

python GetPhishingFeatures.py -u [url] [verdict]

Det betyder, at output for hver URL vil bestå af 31 tal (-1 for phishing, 0 for suspicious eller 1 for legitim), hvor hvert tal repræsenterer hvert af de 30 målepunkter, og det sidste tal er om det er phishing eller ej, svarende til den [verdict] man angav.

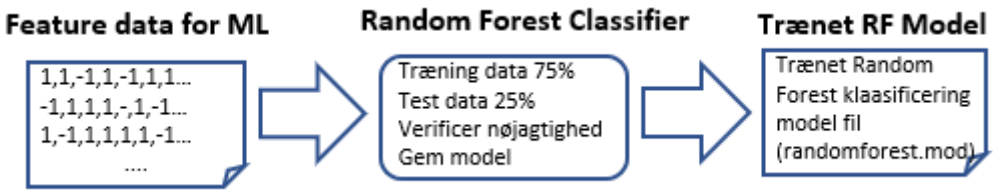
Vi kan se, hvordan vi kan skabes data til et træningssæt ved at køre modulet med 2 forskellige type filer, en med phishing URL'er og en med legitime URL'er:

python GetphishingFeatures.py -i PhishingFilename.txt -1

som skaber en fil ved navn **PhishingFilename.out** og ved

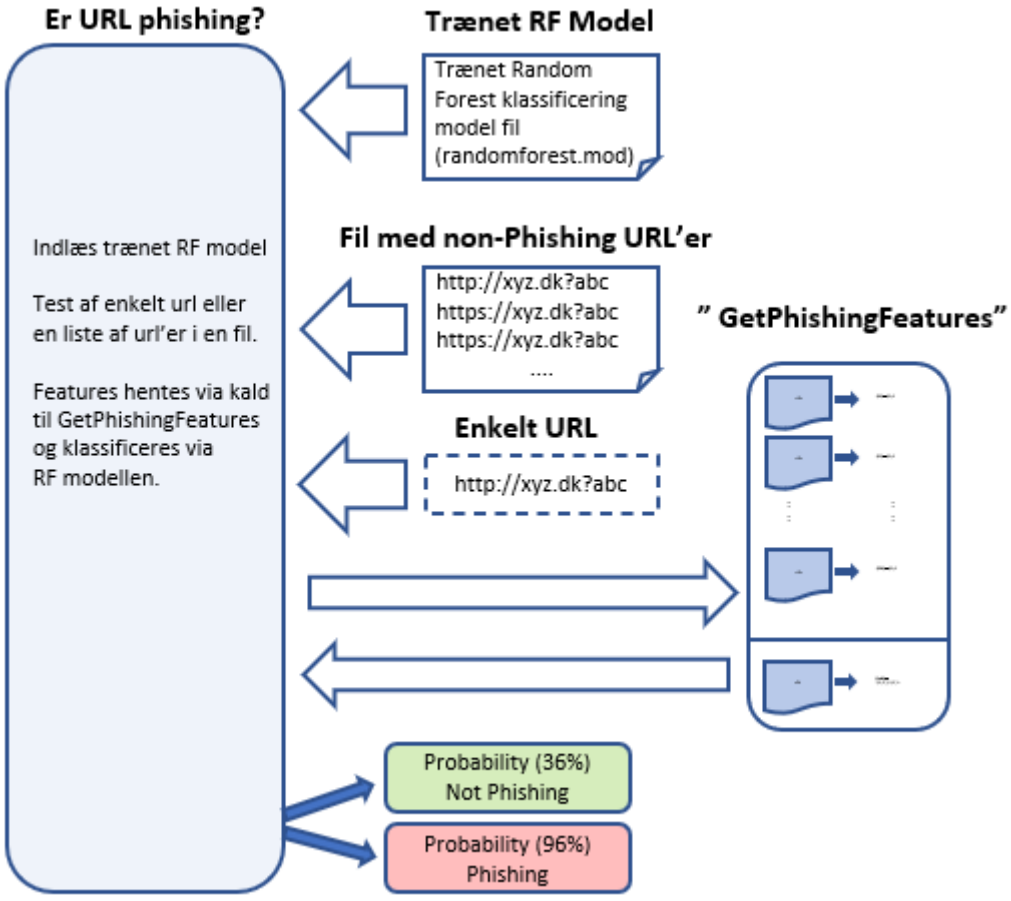
python GetphishingFeatures.py -i NonPhishingFilename.txt 1

som skaber en fil ved navn **NonPhishingFilename.out**.
Disse 2 filer lægges sammen i en fil, og vil være input til næste modul.



Figur 5 Flow af træning af klassificeringsmodel

I ovenstående diagram ser vi resultatet fra figur 4 blive brugt som input i TrainClassifier.py, hvor vi træner modellen med vores data ved brug af Random Forest Classifier(RF).
Vi får da en trænet RF model som resultat.



Figur 6 Flow af forudsigelser af URL(er)

I ovenstående diagram ser vi processen af at indsætte et URL eller en fil af URL'er, som vi gerne vil vide om er phishing eller ej, og vores træningssæt fra figur 5.
Der kaldes GetPhishingFeatures, hvor URL'et eller filen med URL'er bliver processeret til en string af (1,-1, og 0'er) eller en fil med (1,-1, og 0'er).

Dette bliver "predictet" med vores trænet RF model, der da giver os en sandsynlighed på om vores URL, eller for hvert URL i vores fil af URL'er, er phishing eller ej og hvor stor sandsynlighed der er for det givne svar er korrekt

8. Design

Målet i design er at lave en designmodel, der beskriver og hjælper til implementering af softwaren.
I forhold til målgruppe og arkitektur har design ikke været en fremtrædende faktor i dette projekt. Brugergrænsefladen er begrænset, da programmet blot viser en chance i procenter, om et URL er phishing og dernæst en kort beskrivelse af, hvordan programmet kommer frem til dens konklusion/beregning. Alt bliver gjort og vist i en terminal eller skrevet ned i filer.

Designet i denne opgave er simpelt, fordi det består af 3 moduler:

1. modul skal hente målepunkter ud af de valgte URL'er, og gemme værdierne af målepunkter for hver af de enkelte URL'er.
2. modul skal tage data fra 1. modul og indlæse dem i en ML-klassifikation, for at kunne få den til at forudse phishing eller ej.
3. modul skal gøre brug af 2. moduls færdig brændte klassifikationsmodel til at komme med en vurdering af, om en URL er phishing eller ej.

Det er forventningen, at disse moduler primært skal køre som baggrundsjobs uden brugerflade.

9. Sikkerhedsaspekter

Da mit program sigter mod at være et internt modul til at analyse data, bør man altid overveje sikkerheden omkring det.
De data, der hentes, er ikke personhenførbare og dermed ikke GDPR-problematiske. De informationer, der er i systemet, er anonymiserede data og de personer, der har brugt URL'et/erne, bliver ikke registrerede.
Der er ingen database, autentifikation, autorisation, e -mails, eller konfidentialitet.
Jeg vurderer derfor, at der ikke er brug specifikke sikkerhedsløsninger.

10. Implementering

Her vil jeg mere detaljeret gennemgå de elementer, der udgør det at kunne identificere de egenskaber, som kendetegner phishing. Dermed får man en dybere indsigt i, hvordan man kan identificere et phishing forsøg.

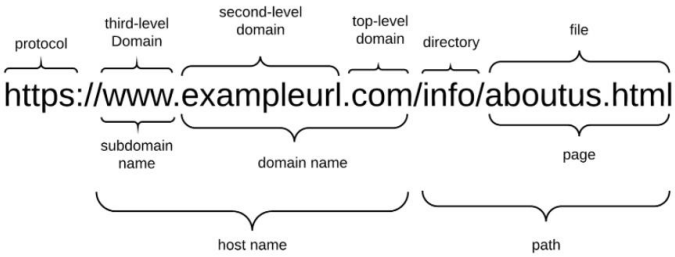
10.1. Om målepunkter

Ved at søge på internettet har jeg fundet flere steder, hvor der fremhæves forskellige egenskaber ved en hjemmeside, som kan afsløre om det er et Phishing site eller ikke. Et af de dokumenter, som bedst beskriver disse og har de fleste målepunkter er "Phishing Websites Features" (se referencelisten 18.1).

.
Disse målepunkter kan opdeles i 4 kategorier:

- Adressen på sitet (Uniform Resource Locator - URL)
- Afvigelser af "normal" brug af egenskaber
- HTML og JavaScript baserede egenskaber
- Domæne baserede egenskaber

Opbygningen af et URL:



Figur 7 Bestanddele af et URL

I dokumentet er beskrevet 30 forskellige målepunkter, som jeg har valgt at implementere alle så godt det er muligt, ud fra antagelsen om at jo flere målepunkter desto mere nøjagtig en klassificering.

Herunder en kort beskrivelse af hvert målepunkt - nogle også med et eksempel. Den engelske overskrift er bibeholdt for reference årsager til både kilde og program.

10.2. Beskrivelse af 30 målepunkter

10.2.1 Address Bar based Features (1-12)

URL'et er det første trin, man kan foretage en analyse af for at afgøre, om det er phishing eller ej. Webadresser til phishing-domæner har nogle specielle særpræg, som man måler på, og nedenfor er nogle af disse adresse-baserede funktioner, som jeg har valgt.

1. Using the IP Address
Et link, der indeholder en IP adresse i stedet for et domænenavn, benyttes af phishing sider da det koster at oprette domænenavnet og phishing sites udskiftes ofte. Her et eksempel "http://112.69.5.420/gib_money_pls.html".

2. Long URL to Hide the Suspicious Part
Lange URL'er kan benyttes til at skjule de dele, som phishing ikke ønsker, man skal opdage. Analyser af længden på lange phishing URL'er starter ved ca. 54 eller deromkring. Samtidigt kan den lange URL skjules ved at have en anden tekst, som vises.

3. Using URL Shortening Services "TinyURL"
URL shortening er en service, der forkorter URL'er (via omdirigeringer) og som kan give yderligere statistik om brugen af linket. Dette giver derved mulighed for at gemme den reelle destination. Et eksempel "https://t.co/dkRMsmJdI9?amp=1" som omdirigerer til "https://www.youtube.com/watch?v=NHTpZV-Dcw4".

4. URL's having "@" Symbol
Et "@" i et URL gør, at visse browsere ignorerer alt før symbolet, hvilket kan få brugere til at tro det er et legitimt link. Et eksempel er "dr.dk/nyhedder@phishingsite.ru/steal_yo_money".

5. Redirecting using "//"
Et andet lignende trick er benytte "//" midt i et URL, hvilket får browseren til at omdirigere til det, der kommer efter - eksempelvis "http://www.legitimate.com//http://www.phishing.com".

6. Adding Prefix or Suffix Separated by (-) to the Domain
Analyser har vist, at der sjældent benyttes "-" i legitime URL'er, og phishing udnytter dette ved at indføre et "-" og kan få linket til at ligne den rigtige adresse.
Eksempelvis "http://www.Confirme-paypal.com/" som ikke ejes af PayPal.

7. Sub Domain and Multi Sub Domains
I samme boldgade benytter phishing sig af at have en eller flere sub-domæner og derved skjule, at det ikke er det legitime site. Så ved at tælle antallet af "." i domænenavnet kan disse identificeres.

8. HTTPS (Hyper Text Transfer Protocol with Secure Sockets Layer)
Alle med fokus på sikkerhed er overgået til at benyttes HTTPS, for at kryptere trafikken. Derfor ser vi også, at phishing gør dette for at give indtryk af at det er et "sikkert" site. Så ved at spørge til på hvor gamle certifikater for et site er, kan man identificere nye og mindre sikre sites.

9. Domain Registration Length
Ud fra antagelsen at phishing sites udskiftes ofte, kan man også se på hvornår et domænenavn udløber, og dermed igen identificere nye og mindre sikre sites

10. Favicon
For at fremstå som et legitimt site benytter phishing sig af at vise favicon fra det rigtige site (favicon vises i fanebladet i browsere). Så ved at fremhæve dem, der ikke henter dette fra samme site, antager vi at disse er phishing.

11. Using Non-Standard Port
I de tilfælde hvor der eksplicit ikke benyttes standard porte til HTTP (port 80) eller HTTPS (port 443), kan det give phishing flere muligheder for at udnytte dig. Eksempel på dette er "http://www.dr.dk:21/notlegal".

12. The Existence of "HTTPS" Token in the Domain Part of the URL
Ved at tilføje "HTTPS" som en del af domænenavnet kan det fremstå som et mere sikkert link - eksempel "http://https-www-paypal-it-webapps-mpp-home.soft-hair.com".

10.2.2 Abnormal Based Features (13-18)

Denne gruppe af egenskaber går på atypiske eller overdreven brug af elementer på hjemmesiden. Nedenfor er dem, jeg har valgt i denne kategori.

13. Request URL
Nogle phishing sider refererer til de legitime sites for at hente billeder og andre ressourcer for at få siden til at virke legitim, så ved at skele til hvor stor en procentdel der refererer til andre sites, kan denne egenskab fremhæves.

14. URL of Anchor
Et andet kendetegn er, at phishing benytter sig af links, der ikke refererer til nogen side. Så ved at kigge på hvor stor en andel af disse links udgør, kan denne egenskab fremhæves.

15. Links in <Meta>, <Script> and <Link> tags
I samme boldgade kan man ved at tjekke andelen af <Meta>, <Script> og <Link>'s, som refererer til andre sites.

16. Server Form Handler (SFH)
Hvis den aktivitet,der udføres ved at "submitte" en form er tom eller refererer til et andet site, kan det være tegn på phishing.

17. Submitting Information to Email
På hjemmesider der benytter "mail() eller "mailto:" kan benyttes til at sende data til Phishing, så ved at tjekke dette kan det fremhæve denne egenskab.

18. Abnormal URL
Oftest indgår host-navnet; der står i WHOIS databasen; i URL'et, så ved at tjekke dette fremhæves denne.

10.2.3 HTML and JavaScript based Features (19-23)

I denne kategori tjekkes der for ting i HTML og JavaScript, som ofte ses på phishing sider.

19. Website Forwarding
Ved analyse af legitime hjemmesider ser man oftest højst én omdirigering, hvor phishing sider godt kan komme op på 4 eller flere omdirigeringer for at skjule sig.

20. Status Bar Customization
For at skjule den faktiske destination ved klik på en hjemmeside, benytter phishing sig af at ændre teksten i statusbaren, som normalt viser destinationen ved klik. Så der skal tjekkes for dette.

21. Disabling Right Click
Phishing sider har ofte højreklik funktionen slået fra, så brugere ikke kan se sourcekoden for skjule dets indhold.

22. Using Pop-up Window
Legitime hjemmesider spørger ikke ofte om personlig info i et popup vindue, men som er noget der er set på phishing sider.

23. IFrame Redirection
Phishing er set benytte en iframe uden ramme for at det ligner, at du er inde på den legitime hjemmeside. Så endnu et målepunkt.

10.2.4 Domain based Features (24-30)

Denne kategori fokuserer på de eksterne oplysninger, man kan finde ud fra domænenavnet.

24. Age of Domain
Ud fra antagelsen at phishing sites udskiftes ofte, kan man også se på hvor gammel et domænenavn er, og dermed igen identificere nye og mindre sikre sites.

25. DNS Record
Hvis en hjemmeside ikke kan genkendes på WHOIS databaser, er det en klar indikator for phishing.

26. Website Traffic
Ved opslag i ranking-databaser - såsom Alexa databasen - kan man antage, at en lav ranking betyder at det er mindre legitime sider - og dermed en phishing egenskab.

27. PageRank
PageRank er endnu en ranking service, der viser hvilke sider der anses for vigtige, også denne kan bruges til at fremhæve de mindre vigtige/"legitime" sider.

28. Google Index
Ofte er phishing sites ikke indekseret af Google, hvorfor dette også er et godt målepunkt.

29. Number of Links Pointing to Page
Analyser viser, at der er næsten ingen links der peger på Phishing sites, så jo flere sider der refererer til en given side, desto mere legitim fremstår den.

30. Statistical-Reports Based Feature
Der tilbydes en del services på nettet, med lister over phishing URL'er og IP adresser - såsom f.eks. PhishTank og StopBadware. Ved at foretage opslag i disse kan endnu et målepunkt laves. Det er services man skal købe for at få adgang, hvorfor jeg ikke har valgt at implementere denne, men er klart noget man vil nemt vil kunne implementere hvis man vælger at benytte systemet i produktion.

I min analyse af disse målepunkter har jeg benyttet en fil med 11.055 målepunktsdata, som er lånt fra samme sted som jeg fik listen over målepunkter (se referenceliste 18.1).

1 = non-phishing, -1 = phishing, 0 = suspicious (det er kun 6 af målepunkterne hvor suspicious er en mulighed)

10.3. Beskrivelse af kode

Jeg har valgt at fremhæve 4 forskellige kodeeksempler, hvor jeg gennemgår hvad de laver og hvordan de fungerer.

10.3.1 Favicon

Her tjekker jeg, om hjemmesiden et favicon og hvis den har det, om det bliver hentet fra samme site som domænet.

Jeg starter med at finde alle links, der har “href”, og tjekker derefter efter ordet “favicon”.
Jeg sammenligner da domænet og det refererede domæne i favicon-referencen:

- Hvis det matcher, er det formodentlig ikke phishing, da det så bare er en hjemmeside med et favicon, den henter fra sig selv.
- Hvis det ikke matcher, er det formodentligt et phishing site, der henter favicon fra den legitime side, den prøver at imitere denne.
- Hvis der i stedet står et “.” som første tegn betyder det, at favicon’et bliver hentet lokalt.
- Hvis der ikke er noget favicon overhovedet, er dette målepunkt irrelevant og defaulter derfor til not phishing.

Der er sikkert andre måder, hvorpå man kan definere et favicon på, men jeg tror ikke at phishing sites bruger meget tid på at benytte disse.

```
# Har Favicon { 1,-1 } (-1 er phishing)
    try:
        for link in soup.find_all('link', href=True):
            if ("favicon" in str(link)):
                first_split = link['href'].split("//")
                # Test om der er http/https
                if ("http" in first_split[0]):
                    second_split = first_split[1].split("/")
                    # Hvis domænenavne forskellig fra url-domæne -
gottem phishing
                    if (domain == second_split[0]):
                        res[9] = 1
                    else:
                        res[9] = -1
                # Hvis første tegn er "." så er den en lokal reference
                elif (first_split[0][0] == "."):
                    res[9] = 1
            else:
                # Ingen favicon - ikke phishing
                res[9] = 1
    except:
        res[9] = 1
```

10.3.2 Request_URL

I denne del af koden tjekker jeg, hvor mange eksterne objekter der bliver refereret til i forhold til det totale antal af objekter, der refereres til.

Først og fremmest tjekker vi, om siden er loaded med BeautifulSoup uden fejl. Vi kan ikke tjekke, hvis vi ikke har loadet siden korrekt. Såfremt den ikke er loaded korrekt, defaulter vi til at siden ikke er phishing, da målepunktet er irrelevant hvis det ikke kan tjekke ordentligt.

Jeg gennemløber BeautifulSoup strukturen og finder alle reference til objekter af typen “img”. Hvis dens “src” refererer til en ekstern side, tæller vi disse op, samtidigt med at vi tæller op for total antal “src”. Til dette benyttes regular expression, som kan være kryptiskt at læse og skrive, men som er en meget effektiv måde at finde de enkelte elementer af en URL.
Jeg har ikke selv skrevet dette regex udtryk, da jeg ikke har erfaring i dette endnu. Så regex-udtrukket har jeg fundet ved en Google søgning.

På samme måde findes antallet for “audio” og “embed”, og det samlede forhold mellen eksterne referencer og total antal referencer indgår i vurderingen, om det er phishing.
Jeg har valgt at benytte samme grænseværdier for, om det er phishing, suspicious og legitimt, som angivet i det dokument, som har været basis for mine målepunkter (se referenceliste 18.1).

```
# Hvor mange referencer til eksterne objekter i forhold til total antal
- Request_URL  { 1,0,-1 } (-1 er phishing)
    if (soup):    # Siden skal være hentet
        no_ref = 0
        no_ext_ref = 0
        for script_tag in soup.find_all('img', src=True):
            try:
                src_domain = re.findall(

r"^(?:https?:\/\/)?(?:[^@\/\n]+@)?(?:www\.)?([^:\/\n]+)",
script_tag["src"])[0]
                if not(src_domain in ["."， ".."] or src_domain ==
domain):
                    no_ext_ref += 1
                no_ref += 1
            except:
                pass
        for script_tag in soup.find_all('audio', src=True):
            try:
                src_domain = re.findall(

r"^(?:https?:\/\/)?(?:[^@\/\n]+@)?(?:www\.)?([^:\/\n]+)",
script_tag["src"])[0]
                if not(src_domain in ["."， ".."] or src_domain ==
domain):
                    no_ext_ref += 1
                no_ref += 1
            except:
                pass
        for script_tag in soup.find_all('embed', src=True):
            try:
```

```
                src_domain = re.findall(

r"^(?:https?:\/\/)?(?:[^@\/\n]+@)?(?:www\.)?([^:\/\n]+)",
script_tag["src"])[0]
                if not(src_domain in ["."， ".."] or src_domain ==
domain):
                    no_ext_ref += 1
                no_ref += 1
            except:
                pass

        try:
            percentage = no_ext_ref / float(no_ref) * 100
            if percentage < 22.0:
                res[12] = 1
            elif 22.0 <= percentage < 61.0:
                res[12] = 0                     # Mistænkelig
            else:
                res[12] = -1
        except:
            res[12] = 1
    else:
        res[12] = 1
```

10.3.3 Valg af klassificeringsmodel

Det er interessant at se, hvor gode de forskellige klassifikationsmodeller er til at forudsige det rigtige resultat. De fleste modeller har hver deres styrker og svagheder, men man kan ikke se bort fra, at Random Forest (RF) og Decision Tree (DT) er de 2 modeller, der skiller sig ud.
I mit Jyniper Notebook test program (t.ipynb) testede jeg alle 10, som jeg havde taget ud som kandidater. Opbygningen og træning af de forskellige modeller er meget ens, hvilket det gjorde det nemt at sammenligne dem, og de fik alle det samme træningsdatasæt og samme testdatasæt. Ved hjælp af en en timer-funktion loggede jeg den tid, de var om at træne modellen (fit) og den tid, de var om at forudsige på testdatasættet (predict), og derefter skrev jeg ud hvor nøjagtige forudsigelserne var. Nedenstående kode viser koden for dette og de resultater.
RF var den model, der klart var den mest nøjagtige, men det er også værd at bemærke, at DT var noget hurtigere med ikke så ringe en præcision. Derfor begyndte jeg at skrue på nogle de af de få parametre der er til RF, for at se om jeg kunne forbedre tiden, den var om at forudsige (PredictTime). Jeg prøvede forskellige værdier på **n_estimators**, **max_depth** og **max_leaf_nodes**.

En kort beskrivelse af disse er:
Ved at begrænse en af dem, påvirker man hvor stor en effekt de andre har.
Jeg måtte derfor forsøge mig frem:

- **n_estimators** er hvor mange træer der skal laves. Her er det tydeligt at se, jo flere træer desto længere tid tog det. Men det var ikke sådan, at man uden videre kunne øge denne parameter og få mere præcision. Det var derimod et spørgsmål om at have så lille et antal uden at det nævneværdigt gik ud over præcisionen. Her kom jeg frem til 25.
- **max_depth** er max dybde på træerne. Her fandt jeg frem til, at en dybde på 15 var bedst.
- **max_leaf_nodes** danner træer med max dette antal noder. Denne parameter er en af måderne, man kan undgå under- og overfitting, hvilket går ud over præcisionen. Her kom jeg frem til, at en værdi på 15.000 gav det bedste resultat.

I og med at disse træer vælges tilfældigt, vil man kunne se, at tiderne og præcisionen varierer fra kørsel til kørsel.

Nedenstående kode viser hvordan jeg tester det de forskellige klassificeringsmodeller, for at finde frem til den bedste, som optimerer præcisionen samtidigt med at den er hurtig.
For hver model laver jeg en tidsmåling på processeringstiden (den tid det tager at lave modellen) og forudsigelsestiden (den tid det tager at finde modellens klassificering om det er phing eller non-phishing).
Det var interessant at se, hvor gode de forskellige klassifikationsmodeller er til at forudsige et resultat og hvor præcis denne var.

```
# Random Forest
tic1 = time.perf_counter()
forest = RandomForestClassifier(n_estimators=25, max_depth=15,
max_leaf_nodes=15000)
forest.fit(X_train, y_train)
toc1 = time.perf_counter()
tic2 = time.perf_counter()
y_test_forest = forest.predict(X_test)
acc_test_forest = accuracy_score(y_test,y_test_forest)
toc2 = time.perf_counter()
print(f"ProcTime:{toc1 - tic1:0.2f} PredictTime:{toc2 - tic2:0.2f} -
Random forest: Accuracy on test data: {acc_test_forest:.3f}")

# Decision tree
tic1 = time.perf_counter()
dtree = tree.DecisionTreeClassifier()
dtree.fit(X_train, y_train)
toc1 = time.perf_counter()
tic2 = time.perf_counter()
y_test_dtree = dtree.predict(X_test)
acc_test_dtree = accuracy_score(y_test,y_test_dtree)
```

```
toc2 = time.perf_counter()
print(f"ProcTime:{toc1 - tic1:0.2f} PredictTime:{toc2 - tic2:0.2f} -
Decision tree: Accuracy on test data: {acc_test_dtree:.3f}")

# Linear SVC classifier
tic1 = time.perf_counter()
svclinear = svm.SVC(kernel='linear')
svclinear.fit(X_train, y_train)
toc1 = time.perf_counter()
tic2 = time.perf_counter()
y_test_svclinear = svclinear.predict(X_test)
acc_test_svclinear = accuracy_score(y_test,y_test_svclinear)
toc2 = time.perf_counter()
print(f"ProcTime:{toc1 - tic1:0.2f} PredictTime:{toc2 - tic2:0.2f} -
SVC with linear kernel: Accuracy on test data:
{acc_test_svclinear:.3f}")

# RBF SVC classifier
tic1 = time.perf_counter()
svcrbf = svm.SVC(kernel='rbf')
svcrbf.fit(X_train, y_train)
toc1 = time.perf_counter()
tic2 = time.perf_counter()
y_test_svcrbf = svcrbf.predict(X_test)
acc_test_svcrbf = accuracy_score(y_test,y_test_svcrbf)
toc2 = time.perf_counter()
print(f"ProcTime:{toc1 - tic1:0.2f} PredictTime:{toc2 - tic2:0.2f} -
SVC with rbf kernel: Accuracy on test data: {acc_test_svcrbf:.3f}")

# Ovo Linear SVC classifier
tic1 = time.perf_counter()
svcovolin = svm.SVC(decision_function_shape='ovo', kernel='linear')
svcovolin.fit(X_train, y_train)
toc1 = time.perf_counter()
toc2 = time.perf_counter()
y_test_svcovolin = svcovolin.predict(X_test)
acc_test_svcovolin = accuracy_score(y_test,y_test_svcovolin)
toc2 = time.perf_counter()
print(f"ProcTime:{toc1 - tic1:0.2f} PredictTime:{toc2 - tic2:0.2f} -
Custom SVC with ovo linear: Accuracy on test data:
{acc_test_svcovolin:.3f}")

# Ovo RBF SVC classifier
```

```
tic1 = time.perf_counter()
svcovorbf = svm.SVC(decision_function_shape='ovo', kernel='rbf')
svcovorbf.fit(X_train, y_train)
toc1 = time.perf_counter()
tic2 = time.perf_counter()
y_test_svcovorbf = svcovorbf.predict(X_test)
acc_test_svcovorbf = accuracy_score(y_test,y_test_svcovorbf)
toc2 = time.perf_counter()
print(f"ProcTime:{toc1 - tic1:0.2f} PredictTime:{toc2 - tic2:0.2f} -
Custom SVC with ovo rbf: Accuracy on test data:
{acc_test_svcovorbf:.3f}")

# NuSVC classifier
tic1 = time.perf_counter()
nusvc = svm.NuSVC()
nusvc.fit(X_train, y_train)
toc1 = time.perf_counter()
tic2 = time.perf_counter()
y_test_nusvc = nusvc.predict(X_test)
acc_test_nusvc = accuracy_score(y_test,y_test_nusvc)
toc2 = time.perf_counter()
print(f"ProcTime:{toc1 - tic1:0.2f} PredictTime:{toc2 - tic2:0.2f} -
NuSVC: Accuracy on test data: {acc_test_nusvc:.3f}")

# OneClassSVM classifier
tic1 = time.perf_counter()
oneclass = svm.OneClassSVM()
oneclass.fit(X_train, y_train)
toc1 = time.perf_counter()
tic2 = time.perf_counter()
y_test_oneclass = oneclass.predict(X_test)
acc_test_oneclass = accuracy_score(y_test,y_test_oneclass)
toc2 = time.perf_counter()
print(f"ProcTime:{toc1 - tic1:0.2f} PredictTime:{toc2 - tic2:0.2f} -
One Class SVM: Accuracy on test data: {acc_test_oneclass:.3f}")

# K nearest neighbours algorithm.
tic1 = time.perf_counter()
knear = KNeighborsClassifier(n_neighbors=5, algorithm='ball_tree')
knear.fit(X_train, y_train)
toc1 = time.perf_counter()
tic2 = time.perf_counter()
y_test_knear = knear.predict(X_test)
```

```
acc_test_knear = accuracy_score(y_test,y_test_knear)
toc2 = time.perf_counter()
print(f"ProcTime:{toc1 - tic1:0.2f} PredictTime:{toc2 - tic2:0.2f} - K
nearest neighbours: Accuracy on test data: {acc_test_knear:.3f}")

# Gradient boosting classifier
tic1 = time.perf_counter()
grad = GradientBoostingClassifier()
grad.fit(X_train, y_train)
toc1 = time.perf_counter()
tic2 = time.perf_counter()
y_test_grad = grad.predict(X_test)
acc_test_grad = accuracy_score(y_test,y_test_grad)
toc2 = time.perf_counter()
print(f"ProcTime:{toc1 - tic1:0.2f} PredictTime:{toc2 - tic2:0.2f} -
GradientBoostingClassifier: Accuracy on test data:
{acc_test_grad:.3f}")
```

Output fra ovenstående kørsel:

```
ProcTime:0.19 PredictTime:0.02 - Random forest: Accuracy on test data: 0.967
ProcTime:0.02 PredictTime:0.00 - Decision tree: Accuracy on test data: 0.957
ProcTime:0.98 PredictTime:0.13 - SVC with linear kernel: Accuracy on test data: 0.929
ProcTime:0.74 PredictTime:0.23 - SVC with rbf kernel: Accuracy on test data: 0.947
ProcTime:0.95 PredictTime:1.31 - Custom SVC with ovo linear: Accuracy on test data: 0.929
ProcTime:1.24 PredictTime:0.40 - Custom SVC with ovo rbf: Accuracy on test data: 0.947
ProcTime:3.57 PredictTime:0.57 - NuSVC: Accuracy on test data: 0.917
ProcTime:2.52 PredictTime:0.65 - One Class SVM: Accuracy on test data: 0.485
ProcTime:0.06 PredictTime:1.20 - K nearest neighbours: Accuracy on test data: 0.941
ProcTime:0.75 PredictTime:0.01 - GradientBoostingClassifier: Accuracy on test data: 0.949
```

10.3.4 Koncept eksempel på anvendelse

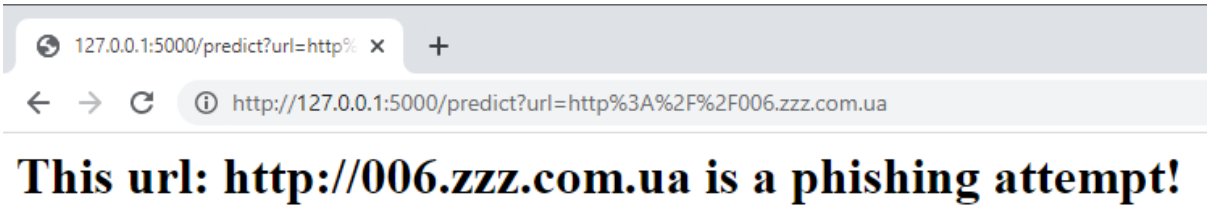
Min ide med systemet er, at det kunne indgå som en del af sikkerhedstjek i et mailsystem. Derfor har jeg lavet et meget enkelt Flask modul (api-mini.py), som tilbyder at man kan kalde dette modul for at få tjekket en URL for phishing. Modulet er udelukkende tænkt som et eksempel på, hvordan man kunne bruge mit system og kan/skal ikke bruges som det er udformet her.
Det følgende er noget professionelle købe systemer også gør brug af.
Forudsætningen er, at mailsystemet indkapsler alle links med til api-mini, som f.eks. hvis et link i en mail så sådan her ud:

http://006.zzz.com.ua

skal det erstattes med:

http://127.0.0.1:5000/predict?url=http%3A%2F%2F006.zzz.com.ua

Resultatet vil i dette tilfælde være



Figur 8 Resultat af api-mini.py

da der er tale om et phishing-link.
Hvis mit system havde klassificeret det som non-phishing, ville browseren blot blive omdirigeret til det oprindelige link.

```
# En route for at teste url for phishing
@app.route('/predict', methods=['GET'])
def predict():
    query_parameters = request.args
    url = unquote(query_parameters.get('url'))
    print(url)
    features = np.array(gpf(url, 1))[:-1]
    inp_data = features.reshape(1, -1)

    # Forudsig om URL er Phishing
    pred = classifier.predict(inp_data)
    print(pred[0])
    if (pred[0] == -1):
        return '<h1>This url: ' + url + ' is a phishing attempt!</h1>'
    else:
        return '<h1>To be redirected to ' + url + '</h1>'    #
Udkommenter denne
        # return redirect(url)                                # og fjern
kommentar på denne

# Start med at hente Random Forest Classifier modellen op
classifier = RandomForestClassifier(
    n_estimators=500, max_depth=15, max_leaf_nodes=15000)
classifier = pickle.load(open('randomforest.mod', 'rb'))
```

11. Test

Test af de enkelte modulers funktionalitet er til for at sikre, at de virker hensigten. Der er valgt ikke at lave unit-tests eller lignende, da resultat af en test af en URL kan give forskellige resultater i takt med, at siderne på internettet ændrer sig.

Gennem hele forløbet er der aktivt blevet lavet disse tests på modulerne, for at løbende være sikker på at det virker som forventet.

11.1 GetPhishingFeatures testet med non-phishing URL:

Jeg har testet GetPhishingFeatures ved at give den et URL, som jeg ved ikke er phishing, og et URL der er phishing.
Som udgangspunkt forventer jeg, at legitime sider ikke giver phishing udslag ved nogle målepunkter, da de ikke er phishing. Dog kan der være gode grunde til, hvorfor nogle hjemmesider giver phishing udslag på enkelte målepunkter.
Ved at skrive:
python GetPhishingFeatures.py -u https://www.dr.dk 1

Forventer jeg at få URL skrevet ud på terminalen, efterfulgt af 31 "1","-1", eller "0"'er, hvor den sidste er 1 (da vi gav verdict "1" i input), og forventer meget målepunkter med "-1".

Jeg får resultatet:
https://www.dr.dk
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, 1, 1, 1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
skrevet ud på terminalen.

Jeg vil gennemgå de målepunkter, hvor den giver phishing udslag og forklare hvorfor:

Målepunkt 13 - Request URL: Da dr.dk er en nyhedsside, giver det mening at der bliver hentet billeder, referencer eller andet fra steder på nettet. Derfor gives der phishing udslag her.

Målepunkt 17 - Submitting_to_email: hvis vi går ind på siden, højreklikker og trykker "view source", og derefter søger efter "mailto" kan vi se der står:
```
<p>Du kan kontakte lytternes og seernes redaktør i DR, Jesper
Termansen, på<!-- --> <a style="color:inherit"
href="mailto:lsrred@dr.dk">lsrred@dr.dk</a></p>
```
Hvilket er, som man kan se, en mulighed at kontakte en medarbejder i DR via e-mail, hvilket ikke er phishing.

11.2 GetPhishingFeatures testet med phishing URL

Jeg gør det samme med et URL, jeg ved er phishing:
OBS: faktisk phishing link. Jeg ved ikke om det stadig er aktivt, men undgå stadig at følge linket:
"http://aerinlangworthy.com/wp-content/uploads/invoice/office19/4345a7b9-9a63-4910-a426-customer-IDPP00C749/authnow/?client_id=4345a7b9-9a63-4910-a426-35363201d503"
Og med verdict "-1", så:
python GetPhishingFeatures.py -u http://aerinlangworthy.com/wp-content/uploads/invoice/office19/4345a7b9-9a63-4910-a426-customer-IDPP00C749/authnow/?client_id=4345a7b9-9a63-4910-a426-35363201d503 -1

Jeg forventer at der kommer phishing udslag fra flere målepunkter.

Jeg får resultatet:
http://aerinlangworthy.com/wp-content/uploads/invoice/office19/4345a7b9-9a63-4910-a426-customer-IDPP00C749/authnow/?client_id=4345a7b9-9a63-4910-a426-35363201d503
[1, -1, 1, 1, 1, 1, 1, -1, 1, 1, 1, 1, 1, -1, 1, 0, -1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1]

Målepunkt 2 - URL_length: Det kan ses tydeligt, at URL'et er langt over 54 tegn, hvilket derfor giver udslag på dette målepunkt.

Målepunkt 8 - SSLfinal_State: Her kan vi se, at URL'ens certifikat er mindre end et år for at udløbe. Det er ukendt, om certifikatet faktisk er udløbet, eller om udsteder er trusted. Som vist i figur##(referer til diagram med signifikante målepunkter) er dette målepunkt et at de målepunkter, der har stor betydning for om et URL er phishing eller ej.

Målepunkt 13 - Request_URL: Her kan vi se, at få eller ingen referencer peger på dette URL. Dette er også et af de mere signifikante målepunkter, der dikterer om et URL er phishing

Målepunkt 17 - Submitting_to_email: Her kan vi se, at der bliver brugt "mail" eller "mailto" i URL'ets soruce kode

Målepunkt 18 - Abnormal_URL: Her kan vi se, at dets domænenavn ud fra "whois" ikke indgår i URL'et.

TrainClassifier testet med non-phishing URL
Jeg har testet TrainClassifier ved at indtaste et URL, jeg ved ikke er phishing (https://www.dr.dk).

Jeg indtaster: "python TrainClassifier.py -u https://www.dr.dk"

Jeg forventer at få et verdict, om det er phishing (-1) eller ej (1), fulgt af sandsynligheden af hvor korrekt det estimat er, hvor "1" er 100% og "0,01" er 1%

Jeg får: "[1] [[0.01 0.99]]"
Hvilket fortæller os,at det ikke er phishing med 99% sikkerhed,
og at det er phishing med 1% sikkerhed.
Med et verdict at det **ikke** er phishing.

11.3 TrainClassifier testet med phishing URL

Jeg tester TrainClassifier igen ved at give den et URL, jeg ved er phishing:
(http://aerinlangworthy.com/wp-content/uploads/invoice/office19/4345a7b9-9a63-4910-a426-customer-IDPP00C749/authnow/?client_id=4345a7b9-9a63-4910-a426-35363201d503"
)
Jeg forventer igen at få et verdict, om det er phishing (-1) eller ej (1), fulgt af sandsynligheden af hvor korrekt det estimat er.

vi får resultatet:” [-1] [[0.506 0.494]]”
hvilket fortæller os, at det er Phishing med 50,6% sikkerhed,
og at det ikke er phishing med 49,4% sikkerhed.
Med et verdict at det **er** phishing.

Testene forløb som forventet, med forventede resultater.
En bemærkning er at TrainClassifier testet med et phishing URL kun gav meget lidt over 50% phishing, hvilket kan vise hvor overbevisende phishing hjemmesider kan være.

12. Evaluering og konklusion

12.1. Evaluering

Status på mit endelige produkt er, at jeg overordnet har fået implementeret alle mine opstillede funktionelle og ikke-funktionelle krav i mit program.

Jeg kan nu hive 30 forskellige målepunkter ud fra et URL. Nogle få af dem er ikke fuldt implementeret, men jeg har givet grund til hvorfor og hvordan dette kan gøres som tidligere beskrevet i rapporten. Nogle af målepunkterne kan arbejdes videre med eller forbedres, men dette har ikke muligt for mig givet min tidsbegrænsning og/eller mit kompetenceniveau.
Programmet kan indlæse en liste af URL og danne et træningssæt ud fra dette. Træningssættet kan bruges til at måles op mod andre givne URL’er og komme frem til, om det givne URL er phishing eller ej.

Jeg har testet forskellige metoder for databehandling, og både på deres præcision og hastighed er jeg kommet frem til et “ideelt” valg, som jeg endvidere har optimeret til at give den bedste hastighed og præcision, der var mulig.

12.2. Konklusion

Jeg synes det er lykkedes at identificere phishing forsøg med god præcision ud fra et URL ved at analysere flere af dets kendetegn og siden bag ved.
Konklusionen er, at man kan sandsynliggøre om et URL er phishing eller ej, og med hvor stor sandsynlighed dette udsagn er korrekt

12.3. Uhensigtsmæssigheder

Det er selvfølgelig en mulighed for, at et meget gennemtænkt og godt konstrueret phishing forsøg vil undgå en del af disse målepunkter og vil blive dømt til at være lav nok ”procent for ikke at være phishing” til ikke at blive markeret som det.
Jeg har benyttet alle de målepunkter i dokumentet (se referenceliste 18.1). Jeg har ikke erfaring nok omkring phishing til at kunne tilføje mine egne aktuelle målepunkter. En videreudvikling vil være, hvis der er flere målepunkter, skal de selvfølgelig også implementeres.
Hvis dette program skal færdiggøres ude i den “virkelige verden”, vil jeg helt klart inkludere personer med større kompetence, så programmet kan være så nøjagtigt og omfattende så

muligt. De primære programmeringsproblematikker, der har været i projektet, har således også været pga. de begrænsninger, som mit kompetenceniveau giver.

12.4. Videreudviklingsmuligheder

Som jeg har forsøgt at lave et simpelt forsøg på, ville dette program kunne blive implementeret som en browser-extension, hvor det vil gøre det let for brugeren at personalisere og bruge det. Det kunne også bruges som en slags "anti-phishing program" i stil med med et anti-virus program.

Jeg kunne godt kunne have tænkt mig at bruge mere tid på at se på optimering af den tid, det tager at få svar tilbage på test af en URL. Hvis det f.eks. skal indgå som en del at beskyttelse af links i e-mail, er det bedst hvis den ekstra kontrol der nu lægges til alle links tager så kort tid som muligt.
Det er meget tydeligt, at det er omkring identificeringen af alle målepunkterne der tager mest tid. At få Random Forest (RF) modellen til at komme med et resultat tager næsten ingen tid. Det at træne modellen har ingen betydning, da det kan gøres uden gene for brugeren.
Det kunne være interessant at se, hvor god RF ville være med færre målepunkter og så få eller ingen eksterne kontroller (f.eks. WHOIS, Google Index, Alexa, m.fl.), da disse kan tage tid - specielt hvis de ikke svarer (time-out). Hvis man kunne bibeholde nøjagtigheden til at genkende phishing ved udelukkende at kigge på URL'ets opbygning og måske på indholdet af selve hjemmesiden, vil jeg gætte på at man kan forbedre svartiden med en faktor 10.

PROCES - afsnit

13. Projektets set-up

Dette projekt er anlagt i en mindre skala, tilpasset forholdene omkring det forudgående praktikforløb og det faktum, at gruppen består af én mand. Da der kun er min egen faglige kompetence at trække på, har jeg skullet være opmærksom på usikkerhed omkring produktet i forhold til størrelse, kvalitet og/eller performance. Det gav også en større projektrisiko i tilfælde af sygdom og dermed usikkerhed om aflevering til tiden.

Foruden projektets størrelse og risikofaktorer har jeg også måtte være realistisk omkring projektets begrænsede faglige kompleksitet pga. mit kompetenceniveau; Jeg har ikke så stor erfaring som udvikler, og jeg vurderer at mit softwareudviklingsniveau er på "Applying" *(Anvende)*, se figur 9.

Figur 9 Kompetenceniveau

Denne type opgave består essentielt i analyse, strukturering og kategorisering. Jeg har således udfordret mig selv omkring mine kompetencer i denne opgave. Da jeg er alene om projektet, har jeg også måtte præstere på alle kompetenceniveauer i mere eller mindre grad. Python som programmeringssprog var velkendt for mig, men de fleste øvrige teknologier og services var nye for mig, så jeg har også i projektperioden måtte læse op på flere områder omkring phishing.

14. Valg og tilpasning af udviklingsmetode

14.1. Proces omkring projektets mål

I min praktikperiode havde virksomheden Statens IT en del krav til opgaven, som bredte sig over flere områder, så opgaven var ikke så specifik i starten. Det var derfor svært at honorere alle ønsker som ene mand på projektet, og derfor blev det accepteret, at jeg fokuserede på virksomhedens indrapporterede mails, hvor jeg trak relevante URL'er ud, som jeg kunne bruge til at trække data ud af, og lave målepunkter ud fra disse. Min praktikvejledning i Statens IT blev desværre ret begrænset pga. flere interne mentorskift, reduktion i personale + indskrænkninger pga. Corona-restriktioner, og jeg nåede derfor ikke nær så langt som ønsket. Efter afslutning på min praktikperiode måtte jeg derfor revurdere målene for mit projekt og derfor forenkle kravene.

14.2. Ændring af udviklingsmetode

Grundet min utilfredsstillende arbejdsproces i praktikken måtte jeg også evaluere min procesmodel, som primært havde været plandreven omkring et udviklingsprojekt. Det havde indtil da været svært for mig at estimere, hvor lang tid det tog at udvikle de enkelte målepunkter, fordi en del af de eksterne services, som målepunkterne gør brug af, var nye for mig. Derfor valgte jeg en agil udviklingsmetode til min projektperiode, hvor jeg ville få en mere fleksibel arbejdsproces, som kunne tillade ændringer både i krav og teknologi.
Det gav mig også mulighed for et iterativt forløb, hvor jeg løbende kunne vurdere, hvor langt jeg kunne nå med projektet.

14.3. Scrum

En af de mest anvendte agile udviklingsmetoder er Scrum. I en Scrum-model struktureres arbejdet i iterationer, som kaldes 'sprints'. Længden af et sprint kan variere, men er mellem 1- 4 uger. Scrum foregår oftest i mindre grupper, hvilket umiddelbart ikke var muligt for mig som en-mandsgruppe. Belært af praktikperiodens begrænsede vejledning vidste jeg dog, at jeg havde behov for feedback og sparring til at foretage løbende prioritering af mine opgaver. Jeg valgte derfor at alliere mig med mine forældre, som havde tid og overskud til give mig jævnlig konstruktiv sparring på min opgave. Jeg lavede således min egen 'pseudo'-Scrum-model.

14.4. Andre metoder

Jeg har også været inspireret af enkelte af principperne fra udviklingsprocesserne Unified Process (UP) og Extreme Programming (XP); Systemudviklingen foregik i relativt korte forløb, hvor hver iteration resulterede i et 'færdigt' program - gerne en forbedret og virkende version af mit program. Jeg havde en adaptiv planlægning, hvor næste forløb blev overordnet tilrettelagt efter evaluering og feedback efter hvert sprint. Krav til systemet var mulige at omprioritere, hvis nødvendigt - herunder også en sortering af opgaver med størst værdi. Slutteligt lavede jeg et simpelt design, hvor man stadig kan bygge godt software.

Ved også at vælge disse udvalgte dele af agile udviklingsmetoder kunne jeg minimere nogle af min udfordringer omkring begrænsede kompetencer og mine risikofaktorer ved at være en én-mandsgruppe.

15. Planlægning og overblik over tidsforløb

15.1. Første tidsplan

Min praktikperiode forløb fra 3. aug. til 9. okt. 2020. Derefter lavede jeg praktikrapport til eksamen d. 30. okt. Umiddelbart efter, i starten af november 2020, lavede jeg en overordnet plan for mit afgangsprojekt for at prioritere mine opgaver. Jeg havde til 11. jan. 2021, dvs. 10 uger i alt.

Oprindelig var min tidsramme og disposition over opgaver således:

Medium/svære målepunkter - 3 uger
Meget svære målepunkter - 2 uger
Machine learning - 2 uger
Automatisering - 2 uger
Rapportskrivning og sammenfatning af dokumentation - 1 uge

Evalueringen af min proces under praktikken nødvendiggjorde dog, at jeg også brugte energi og tid på at revurdere min tilgang til projektet som før beskrevet. I den første uge af projektperioden (sprint 0) skrev jeg derfor på projektets introduktion og problemstilling for nærmere at analysere og redefinere mine ønsker for opgaven og dermed få en mere præcis problemformulering end hidtil. Ved at opstille en hypotese blev jeg også mere skarp på, hvad jeg ville arbejde med i opgaven.

15.2. Ændret tidsplan

Efter 1. sprint på 1 uge ændrede jeg min tidsplan pga. ændret problemformulering *og* sygdom. Jeg valgte at droppe automatisering helt, da det ikke længere passede til projektformuleringen. Jeg kunne dermed bruge mere tid på programmering omkring målepunkter og have mere fleksibilitet omkring estimering til at lave Machine Learning og skrive rapport.

Den nye tidsplan og disposition over opgaver blev således:
Medium-svære målepunkter - 3 uger
Meget svære målepunkter - 2- 3 uger
Machine Learning og test - 1- 2 uger
Rapportskrivning og sammenfatning af dokumentation - 1 uge

Mit mål var at blive færdig med programmeringen senest efter de næste 8 uger, så jeg i hvert fald havde minimum 1 uge til at skrive rapport færdig. Hvert sprint blev planlagt til at vare ca. 1 - 2 uger, hvor jeg som udgangspunkt havde aftalt at møde til evaluering og

sparring på mandag eftermiddage. Løbende dokumentation og rapportskrivning samtidig med opgaverne var stadig nødvendig for at nå i mål.

16. Dokumentation af og refleksion over proces

16.1. Overordnet om sprints og tidsestimering:

At udvikle et program stringent efter SCRUM metode som én mandsgruppe er ikke muligt, men jeg har brugt elementer fra SCRUM ved lave ugentlige sprints og brugt min forældre som neutral part, der kunne hjælpe mig i mine refleksioner og synliggøre min prioriteringer i projektet. Jeg har især brugt retrospektive for at kunne forbedre min arbejdsmetode og -resultater. Det har fastholdt mig i dokumentation af mine sprints og min videre proces.

Jeg har også søgt faglig sparring i min omgangskreds, som har kommunikeret med mig i løbet af mine sprints for at holde min motivation oppe.

Jeg har foretaget estimater på, hvor meget jeg skulle arbejde for at nå i mål med projektet. I de første 4 - 5 uger arbejdede jeg ca. 5 - 6 timer dagligt 5 dage om ugen, men særlig i 2. sprint og midtvejs i processen måtte jeg omprioritere og sætte flere mandetimer ind, så jeg har derefter arbejdet ca. 7 timer dagligt i 6 - 7 dage ugentligt.

16.2. Beskrivelse af indhold i sprints:

16.2.1. Sprint 1:

Målet for sprint 1: At få sat mit udviklingsmiljø op og arbejde hen imod at få mit grundprogram til at fungere, hvor jeg kan indtaste et URL, køre det igennem de 6 målepunkter jeg har indtil nu (lavet i praktikperioden) og få programmet til at virke.
Retrospektive: Det positive ved dette sprint var, at jeg nåede at sætte miljø op og fik min basis i orden. Det negative var klart, at jeg blev syg i næsten en uge og derfor ikke kunne fokusere på mit mål omkring at få programmet til at køre med svar omkring URL. Denne dårlige start nødvendiggjorde, at jeg måtte omprioritere min oprindelige plan og skulle bruge mere tid i næste sprint for at indhente noget af min tabte tid.

16.2.2. Sprint 2:

Mål for sprint 2: At sætte flere arbejdstimer ind i løbet af ugen (6 - 7 timer dagligt) , så jeg når følgende mål for næste uge: Ligesom sidste uge, at give programmet et URL og få et svar tilbage. Jeg vil også undersøge muligheden for sandboxing på længere sigt.
Retrospektive: Jeg har nu et enkelt program der kan processere et URL med 6 målepunkter og give mig et svar på % del phishing eller ej. Jeg nåede således det essentielle mål for sprint 2, og det var tilfredsstillende, at selve programmeringen lykkedes for mig. Jeg har undersøgt sandboxing i forhold til mit projekt, og set i forhold til tidsestimering har jeg besluttet at undlade dette.

16.2.3. Sprint 3:

Målet for sprint 3: Jeg har 30 målepunkter, jeg gerne vil nå til projektet i alt og har lavet 6 indtil videre. Mit mål for næste uge er at nå 5 af disse målepunkter, da jeg vurderer, at opstarten bliver sværest.
Retrospektive: Reelt var mit 3. sprint kortere end et normalt sprint på en uge og jeg nåede næsten i mål med 4 målepunkter, så i forhold til sprintets varighed er jeg tilfreds med mine mål. Min arbejdsstruktur blev også forbedret, da jeg i 3. sprint besluttede at arbejde væk fra hjemmet de fleste dage for at bevare fokus og ikke at lave for mange uvedkommende aktiviteter.

16.2.4. Sprint 4 (varede ca. 2 uger):

Målet for sprint 4: At nå 8 - 10 målepunkter for at holde min tidsplan. Da Jeg i slutningen af 3. sprint fandt nogle gode referencer (henvisning til hvilke?) til videre håndtering, har jeg en forhåbning at nå mere end 10. Jeg vil fortsat arbejde væk fra hjemmet de fleste dage, da jeg bedre kan koncentrere mig.
Retrospektive - Jeg nåede desværre kun 4 målepunkter; jeg undervurderede opgaverne som tog længere tid end forventet; Hver opgave omkring et målepunkt er mere kompliceret end jeg troede, og jeg skulle finde flere nye eksterne services end forventet. Jeg må derfor omprioritere, og vælger derfor fremover at udvælge målepunkterne efter sværhedsgrad og ikke i rækkefølge som beskrevet. Jeg overvejer at droppe nogle af de mest komplicerede målepunkter, da jeg er ikke sikker på at kunne nå alle 30 målepunkter inden for projektperioden. Det giver derfor ikke mening at sætte et mål med at nå et bestemt antal målepunkter for hvert sprint, men at se på omfanget af hvert målepunkt.
Jeg brugte også tid på at omstrukturere den eksisterende koden til flere dele, så koden er mere overskuelig. Formålet var differentiere koden, så man kan håndtere den bedre.
Derudover har jeg også lavet en overordnet disposition for den skriftlige del af opgaven, så jeg ikke kun fokuserer på at lave kode, men også får skrevet tekst til min opgave.

16.2.5. Sprint 5:

Målet for sprint 5: Min plan er at lave flere medium-svære målepunkter i starten af dette sprint, og de sværeste målepunkter venter jeg med at håndtere til slutningen af sprintet, til jeg har bedre overblik, om jeg kan nå dem eller skal prioritere, hvad jeg kan nå. I forhold til selve opgaveskrivning vil jeg begynde at have mine spredte notater mere struktureret på skrift og sætte dem ind i min disposition.
Retrospektive: Jeg har nået 3 medium og 3 svære målepunkter, hvilket er mere end jeg forventede. Det skyldes, at jeg i dette sprint ikke har behøvet at sætte mig ind i ny teknologi, og har derfor kunnet udnytte tiden bedre til at programmere. Mit program kører ligeledes fint med de ekstra målepunkter, så jeg opnår en fortsat forbedret version som tiltænkt. Jeg har fået startet på notater til opgaveskrivning omkring de tekniske afsnit under ‘produkt’ og fået et bedre overblik over, hvad og hvordan jeg skal skrive opgaven, hvilket jeg foreløbig er tilfreds med.
Jeg har nu et fornyet håb om at nå de sidste 10 målepunkter som oprindeligt planlagt.

16.2.6. Sprint 6 (varede 1½ uge):

Målet for sprint 6: Ved at arbejde alle ugens dage vil jeg forsøge at nå igennem de sidste 10 målepunkter ud af 30. Jeg vil ligeledes starte på Machine Learning og finde ud af hvilken slags, jeg vil bruge.
Derudover vil jeg gerne nå at skrive notater til tekst til de fleste afsnit i opgaven, så jeg får skrevet ca. en ⅓ - del af opgaven.
Retrospektive: Jeg har nået igennem alle 30 målepunkter, og det er *meget* tilfredsstillende at have nået mit oprindelige mål. Ligeledes har jeg fundet den mest præcise Machine Learning til mit formål, og har det fået til at fungere med programmet.
Jeg har også skrevet notater og skitser til ca. 14 sider af rapporten, hvilket jeg er godt tilfreds med. Jeg er godt tilfreds med min indsats, som også har krævet en massiv arbejdsindsats, hvor jeg stort set ikke har lavet andet end at sove, spise og arbejdet på mit projekt.

16.2.7. Sprint 7:

Målet for sprint 7: Da jeg har arbejdet intenst det foregående sprint, vil jeg gerne skrue en smule ned for mit arbejdstempo og -timer for at få have energi til at lave en ordentlig rapport. Jeg vil derfor prøve at gå tilbage til 6 - 7 timers arbejdsdag og 6 dages arbejdsuge. Jeg vil arbejde med at lave diagrammer og grafer til mine rapport og få skrevet på manglende afsnit. Jeg vil også prøve at nå at lave test til mit program.
Retrospektive: Jeg har fundet diagrammer og grafer, hvilket hjælper til bedre overblik over essentielle afsnit. Andre personer har læst korrektur på det, jeg har skrevet indtil videre, og tilbagemeldingen er at jeg skal beskrive mere, så sammenhængen i projektrapporten er tydelig. Så jeg har brugt og skal bruge mere tid på at skrive end forventet. Det er ikke, fordi skriveprocessen er svær, men det tager lang tid at lave præcise formuleringer, samtidig med at jeg skal forholde mig til den anbefalede struktur til rapporten. Jeg har derfor ikke kunnet skrue så meget ned for mit arbejdstempo som ønsket, men da mit program reelt er færdigt, er jeg dog ikke så presset som i forrige sprint. Jeg er ikke begyndt på endelige tests endnu, hvilket jeg skal nå i næste og sidste sprint.

16.2.8. Sprint 8: (sidste sprint)

Målet for sprint 8: Resten af den foreløbige rapport skal rettes igennem, og så mangler jeg at lave test, at skrive kommentarer til udvalgte kode-dele, og skrive evaluering på produkt- og procesdel. Jeg vil stile mod at være helt færdig 2 dage før aflevering, så jeg kan få læst korrektur på min rapport og have tid til at rette den, da det var min erfaring fra 7. sprint, at det tager længere tid end forventet.
Retrospektive: Jeg har fået lavet det sidste af rapporten, som tog noget længere tid end forventet. Jeg har i det sidste sprint løbende fået læst korrektur af flere, så i stedet for at vente med korrektur til sidst har jeg gjort det undervejs; Det har fungeret godt, fordi der var en del rettelser og så har jeg skullet ikke stresset over det. Jeg har både fået IT-kyndige og andre personer til at læse det igennem og give feedback.

17. Refleksioner over proces

17.1. Det dårlige ved processen

Jeg fik en dårlig start på projektet med sygdom, og det var til tider svært at holde motivationen oppe - især da min tidsplan begyndte at skride, fordi jeg ikke nåede de ønskede mål med mine målepunkter. Jeg skal stadig blive bedre til planlægge; jeg har ikke været god til at lave sprintmøder alene, og jeg har haft svært ved at estimere mit tidsforbrug. Jeg ville gerne have lavet et lidt større anlagt projekt og var måske for ambitiøs i starten, men det var også spændende med de mange muligheder og retninger, som opgaven kunne indeholde.
Som en konstant underliggende faktor har jeg været ret isoleret i både min praktik- og projektperiode; fagligt har jeg manglet sammenspil og feedback, og jeg har heller ikke haft mulighed for arbejdsfordeling i min en-mandsgruppe. Menneskeligt har jeg pga. Corona-epidemien ikke fået den energi, som man får ved samvær med sin omgangskreds.
Hvis jeg kunne gøre hele processen om, ville det optimale være at jeg både i praktik- *og* projektperioden havde haft gruppemedlemmer at lave projektet med, - opgaven kan fint deles op mellem flere personer. Det er hårdt at lave et projekt alene; både mentalt at være alene om alle alle opgaver, men der er også større risiko på andre parametre, som jo blev aktuelt for mig med f.eks. sygdom i starten.

17.2. Det gode ved processen

Det gode og bedste i processen var, at jeg faktisk nåede i mål med mine omdefinerede mål, og jeg synes, at jeg har lavet et godt produkt.
Jeg har fagligt lært meget; Min læringskurve har været stejl og jeg har lært rigtig meget på kort tid. Jeg kan mærke, at jeg er blevet bedre til at programmere set i forhold til mit oprindelige kompetenceniveau. Trods en presset projektperiode er jeg stadig interesseret i it-sikkerhed og vil fortsat gerne beskæftige mig med emnet.
Jeg har lært, at den agile metode passer mig godt, især med korte sprints i stedet for længerevarende opgaver, da man nemmere kan ændre opgavens opbygning og tilpasse den til at løse opgavens problem.
Jeg er ikke fan af alle punkter i XP eller Scrum, men synes at nogle udvalgte af dem kan være ekstremt gode at benytte sig af.
Jeg synes, at sprintmøder kan være langtrukne, men jeg kan mærke at jeg klarer mig bedre, når jeg har et mål at gå efter for hvert sprint.
Jeg er glad for, at jeg allierede mig med nogle personer (mine forældre), da det er nemmere at formulere sine overvejelser, når man har nogen at snakke med. Jeg er blevet bedre til at bede om hjælp og panikkede derfor ikke. Og så brugte jeg min stædighed konstruktivt til at blive færdig.
Jeg klarede mig således godt ved at bruge en del af min læring fra min praktik-proces til at gøre det bedre i min projektproces. Jeg tænkte også i alternative løsninger som at flytte fysisk arbejdssted. Så jeg har formået at ændre nogle af ting, som har været udfordringer for mig tidligere.

17.3.Konklusion på proces

Min proces har været udfordrende og presset, da jeg har arbejdet alene på projektet og jævnligt har måtte omprioritere pga. tidspres. Jeg har trods dette formået at opnå målene for projektet, fordi jeg fra starten af har været opmærksom på nogle af mine svagheder omkring arbejdsstruktur og har kompenseret for dette ved at være fleksibel i min arbejdsmetode. Jeg er derfor overordnet tilfreds med min lære- og arbejdsproces i dette projekt.

18. Referencer

18.1 Til brug for valg af phishing egenskaber har jeg primært brugt denne beskrivelse "Phishing Websites Features" fra 13-06-2015 skrevet af:

- Rami M. Mohammad, School of Computing and Engineering, University of Huddersfield, Huddersfield, UK.
- Fadi Thabtah, E-Business Department, Canadian University of Dubai, Dubai, UAE.
- Lee McCluskey, School of Computing and Engineering, University of Huddersfield, Huddersfield, UK.

Samt deres datasæt til brug for analyse.'Training Dataset.arff' samt beskrivelse 'Phishing Websites Features.docx' fra: **https://archive.ics.uci.edu/ml/machine-learning-databases/00327/**

18.2 Inspiration hentet fra Kaggles hjemmesider om emnet Phishing og analyse af data. Kaggle er et selskab under Google som tilbyder holder konkurrencer, har artikler, datasæt, mm. indenfor machine learning.
https://www.kaggle.com/shubha23/identification-of-phishing-websites
https://www.kaggle.com/akashkr/phishing-url-eda-and-modelling

18.3 "Towards data science" er selskab der tilbyder et site der har alle mulige artikler om forskellige emner, heriblandt phishning, machine learning m.m.
https://towardsdatascience.com/phishing-domain-detection-with-ml-5be9c99293e5

18.4 "Scikirt-learn"'s officielle hjemmeside, for at forstå deres implementering af klassifikationsmodeller.
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html?highlight=random%20forest#sklearn.ensemble.RandomForestClassifier

18.5 Hentede inspiration og eksempler på kode mht. Flask.
https://flask.palletsprojects.com/en/1.1.x/quickstart/
https://programminghistorian.org/en/lessons/creating-apis-with-python-and-flask

18.6 Analytica Vidya som tilbyder kurser. Læst en artikel skrevet af Abhishek Sharma, 12. maj 2020 for at bedre at forstå Decision Tree og Random Forest classifiers.
https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/#:~:text=Each%20node%20in%20the%20decision,to%20generate%20the%20final%20output.&text=The%20Random%20Forest%20Algorithm%20combines,to%20generate%20the%20final%20output.

18.7 Endnu en artikel læst - skrevet af Niklas Donges 16. juni 2019 - for bedre forståelse af Random Forest og hvordan den fungerer.
https://builtin.com/data-science/random-forest-algorithm

18.8 Lister over phishing links til brug for at danne træningsdata.

https://github.com/mitchellkrogza/Phishing.Database

18.9 Hjemmesider til hjælp for at tweake regular expressions til ønskede formål.
https://regexr.com/
https://regex101.com/

18.10 Hjælp til at beskrive fordelene ved anvendelsen af ML i forskellige områder.
https://blog.innofactor.com/dk/machine-learning-derfor-er-det-en-fordel#:~:text=Machine%20Learning%20kan%20let%20bruge,for%20at%20finde%20relevante%20variabler.

18.11 Youtube videoer forståelse af ML og RF.
https://www.youtube.com/watch?v=eM4uJ6XGnSM
https://www.youtube.com/watch?v=nKW8Ndu7Mjw&list=PLIivdWyY5sqJxnwJhe3etaK7utrBiPBQ2&index=2
https://www.youtube.com/watch?v=3kYujfDgmNk