CHALMERS

GÖTEBORGS UNIVERSITET

Jämförelser mellan ”bill of materials” för anläggningsmaskiner

Examensarbete inom högskoleprogrammet Datateknik

TOMAS ALANDER
ALEXANDER BRATIC

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg 2023
www.chalmers.se

Examensarbete 2023

Jämförelser mellan ”bill of materials” för anläggningsmaskiner

TOMAS ALANDER
ALEXANDER BRATIC

GÖTEBORGS
UNIVERSITET

CHALMERS

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
Göteborgs Universitet
Göteborg 2023

Jämförelser mellan ”bill of materials” för anläggningsmaskiner
TOMAS ALANDER
ALEXANDER BRATIC

Handledare: Sakib Sistek, Institutionen för Data- och Informationsteknik
Examinator: Lars Svensson, Institutionen för Data- och Informationsteknik

Examensarbete 2023
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Skriven i LaTeX
Göteborg 2023

Jämförelser mellan ”bill of materials” för anläggningsmaskiner
Examensarbete inom högskoleingenjörsprogrammet i datateknik
TOMAS ALANDER
ALEXANDER BRATIC
Institutionen för data- och informationsteknik
Chalmers Tekniska Högskola
Göteborgs Universitet

Sammanfattning

I samarbete med företaget Diadrom Systems AB skapades olika metoder för att hantera, analysera och jämföra en stor datamängd av ”bill of materials” (BOM) för olika anläggningsmaskiner. Detta utförs i syfte att kunna minska onödig produktvariation bland maskiner och att kunna identifiera maskindelar som gör maskinexemplar eller modeller speciella. I rapporten beskrivs hur olika metoder testades för att göra övergripande jämförelser på hela datamängden, där kvantifierbara skillnader mellan modeller kunde identifieras. Även skillnader inom samma modell kunde beräknas för att identifiera vilka maskiner som sticker ut. En enkel applikation med grafiskt användargränssnitt skapades för att låta användare välja två maskiner, navigera deras struktur och få skillnader och likheter markerade på en detaljerad komponent nivå. Även mer specifika undersökningar av maskinernas hytter och antalet komponenter som innehåller bly utfördes efter önskemål från industriverksamma.

Abstract

In collaboration with the company Diadrom Systems AB, different methods were created to handle, analyze and compare a large data set of ”bill of materials” (BOM) for different heavy equipment vehicles. This is done with the aim of being able to reduce unnecessary product variation among machines and to be able to identify machine parts that make machine individuals or models special. This report describes how different methods were tested to make overall comparisons on the entire data set, where quantifiable differences between models could be identified. Differences within the same model could also be calculated to identify which machines stand out. A simple graphical user interface application was created to allow users to select two machines, navigate their structure, and have differences and similarities highlighted at a detailed component level. More specific examinations of the machines’ cabins and the number of components containing lead were also carried out at the request of key users.

Nyckelord: bill of materials, BOM, maskininlärning, t-SNE, Jaccards index, anläggningsmaskiner.

Förord

Detta är ett examensarbete för kandidatexamen på högskoleingenjörsprogrammet i datateknik vid sektionen för Data- och informationsteknik på Chalmers Tekniska Högskola. Det skrevs inom ramen för kursen LMTX38 under vårterminen 2023.

Författarna vill rikta ett stort tack till Diadrom Systems AB och vår handledare Henrik Fagrell som försett oss med resurser, kontakter, insikter och inspiration för att kunna utföra detta arbetet. Tack för förslaget till arbetet, idéer till intressanta problemområden och all avsatt tid till handledning. Vi vill även rikta ett stort tack till vår handledare Sakib Sistek vid Chalmers för hans vägledning under hela arbetets gång. Tack för hjälpen med att komma igång med arbetet, hjälpen med rapportskrivandet, all avsatt tid till handledning och alla goda samtal som följt med. Avslutningsvis vill vi tacka familj och vänner för deras viktiga stöd under arbetets gång.

Tomas Alander & Alexander Bratic, Göteborg, Juni 2023

Beteckningar

API	Application Programming Interface
BOM	Bill of materials
CSV	Comma Separated Values
DIU	Data Integration Unit
DTM	Document Term Matrix
JSON	JavaScript Object Notation
PIN	Product Identification Number
PLM	Product Lifecycle Management
RU	Request Unit
SDK	Software Development Kit
SVD	Singulärvärdesuppdelning
t-SNE	T-distributed Stochastic Neighbor Embedding

Innehåll

Figurer

1
Introduktion

I detta kapitel introduceras arbetets bakgrund, följt av dess mål och syfte. Slutligen ges viktiga frågeställningar och avgränsningar.

1.1 Bakgrund

”Bill of materials” (BOM) är en produkts strukturerade materialförteckning och kan skapas eller användas vid flera olika stadier av dess livscykel. Den kan hjälpa ett företag med upphandling av material och planering för kostnader eller lagerplats. Då en komponent i en produkt ska ersättas kan BOM användas för att identifiera delar som påverkas [1].

För vissa produkter kan det vara intressant att analysera skillnader och likheter i deras innehåll och struktur, för att identifiera komponenter av intresse. Metoder för detta kan utnyttjas till felsökning eller till att förbättra framtida produkter. Eventuellt kan dessa skillnader reduceras för att minska överflödig variation bland modeller och därmed spara på resurser.

Vid analys av en större mängd data, kan jämförelser mellan maskiner och dess komponenter vara en tidskrävande uppgift. Därför behövs det program och metoder för att automatisera och effektivisera jämförelser mellan BOM. Detta ska utvecklas på uppdrag av Diadrom Systems AB, som är ett företag med kunskap inom diagnostisk åt bilbranschen, inbyggda system och it-system. Företaget har bidragit med 7544 filer av BOM för olika moderna anläggningsmaskiner i detta syfte.

1.2 Syfte

Syftet med att identifiera olikheter bland en mängd maskiner är att minska produktvariationen. Att reducera onödig variation kan vara fördelsamt ekonomiskt och minska på resurser som används vid tillverkning. Att identifiera unika delar för de olika maskinerna kan utnyttjas vid felsökning eller när man vill veta varför en viss maskin eller modell presterar bättre eller sämre än andra. Efter samtal med industriverksamma lyftes även behovet av att i detalj jämföra maskinernas hytter, då dessa möjligtvis har stor produktvariation. Att identifiera komponenter som innehåller bly i BOM strukturen kan göra det lättare att ersätta eller återvinna dessa.

1.3 Mål

Lämpliga metoder ska användas för att jämföra hela datamängden av BOM-filer. Filerna ska också jämföras med varandra parvis för att identifiera skillnader och likheter. Vilka maskinmodeller datamängden består av ska identifieras och jämförelser mellan modeller och mellan maskiner inom samma modell ska utföras. Jämförelser av maskinernas hytter och antalet komponenter som innehåller bly ska utföras och en enkel applikation med grafiskt användargränssnitt ska skapas som en del av arbetet.

1.4 Frågeställningar

Följande frågor kommer att behandlas i denna rapport.

- Hur kan maskinexemplar jämföras med BOM, genom automatiserade metoder, för att förbättra kontrollen av produktvariation?
- Finns det skillnader på en mer detaljerad nivå för hytter och går dessa delar att jämföras separat?
- Kan användargränssnittet i applikationen ge en snabb och enkel förståelse av detaljerade skillnader mellan två maskiner?

1.5 Avgränsningar

Vidare undersökning och analys av kostnader för olika maskindelar och deras miljöpåverkan står utanför arbetets omfattning.

Aspekter som design, inloggning för användare och att kunna lägga till maskiner i databasen via applikationen är utanför arbetets omfattning.

2

Teknisk bakgrund

Under projektets gång användes olika tekniker och mjukvara för att utföra arbetet. Dessa beskrivs och förklaras i detta stycke.

2.1 Bill of materials

En ”bill of materials” (BOM) är en strukturerad materialförteckning. Den kan komma i olika former av struktur, innehålla olika typer av information och vara gjord för olika stadier av en produkts livscykel, så som vid design, tillverkning, produktion eller färdig produkt. Den kan innehålla information som kostnad för de olika materialen eller hur de ska användas [1]. För det här arbetet används och analyseras BOM:s av anläggningsmaskiner som färdig produkt. Strukturen är på flera nivåer så att en komponent i maskinens första led kan bestå av en egen lista av komponenter som i sin tur kan ha flera nivåer. Kan kallas förälder-barn struktur eller trädstruktur. Varje komponent har en kvantitet och enhet, men ingen information om kostnad eller liknande. Varje BOM kommer i detta fallet med en avsatt marknad (land) för den färdiga maskinen.

2.2 Product Lifecycle Management

Inom industrin är Product Lifecycle Management (PLM) en process för att hantera en produkts hela livscykel, från framtagning och design till tillverkning, service och slutligen demontering och återvinning. Genom användning av olika programvaror behandlas och hanteras information om produktens livscykel för att stödja alla processer som är involverade i framtagning, tillverkning och slutligen återvinning av produkten.

Genom att lagra data kan information om produkten sparas, hanteras och sökas. Syftet med ett PLM-system är att effektivisera varje steg i produktutvecklingsprocessen. Genom att spara information från alla steg i utvecklingsprocessen kan företaget säkerställa att all nödvändig information är tillgänglig för alla inblandade parter. Detta bidrar till att spara tid och resurser, vilket i sin tur leder till en snabbare produktionscykel och ökad lönsamhet.

Ur ett miljö- och hållbarhetsperspektiv kan PLM bidra till att minska avfall och miljöpåverkan genom hela produktens livscykel. Genom att noggrant dokumentera

produktens livscykel kan företaget identifiera områden där resurser kan sparas, avfall kan minskas, och återvinningen kan förbättras. PLM-data kan också användas för att minska produktens klimatpåverkan genom att aktivt minimera användningen av skadliga råmaterial samt reducera energikonsumtionen vid transport och produktion. PLM ökar inte bara effektiviteten och lönsamheten för ett företag, det är också ett viktigt verktyg för att främja hållbarhet och miljöarbete[2] [3].

Viktigt att ha i åtanke är att det inte finns en klar definition av PLM utan att det är ett arbetsmetod och det kan finnas skillnader vid implementering mellan bolag.

2.3 JavaScript Object Notation

JavaScript Object Notation (JSON) är ett textbaserat filformat som används för utbyte av data. Två strukturer kännetecknar JSON: en samling av ”name-value pairs” och en lista av värden. Name-value pairs realiseras i många olika språk som objekt, dictionary eller hashtabell.

Det finns en samling av strukturer som är universella och alla moderna programmeringsspråk stödjer dem på ett eller annat sätt[4]. I JSON tar dessa strukturer formen av:

- Objekt: en oordnad samling av name-value pairs som omsluts av måsvingar.
- En vektor: bestående av en ordnad samling värden separerade av kommatecken omslutna av hakparenteser.

JSON har sju olika värdestyper: en textsträng omsluten av citattecken, ett numeriskt värde, objekt, array, true, false och null[5].

2.4 BOM-filer

BOM-filerna som har använts är alla JSON-objekt med en trädstruktur, där filstorleken har varierat mellan 340 och 2800 KB och antalet noder varierat mellan 800 och 7000. Varje fil har haft samma trädliknande struktur med maskinens pin i rotnoden och dess komponenter som dess barn. Dessa barn är i sin tur uppbyggda av olika komponenter.

På grund av sekretess utelämnas dessa filer från rapporten; men en exempelfil har skapats för att kunna ge förståelse för bom-filernas struktur. Det är dock viktigt att notera att detta exempel bara belyser strukturella likheter. I exempelfilen nedan har rotnoden två barn, som i sin tur har egna barn. I projektet har de största BOM-filerna innehållit närmare sjutusen noder, vilket resulterar i en JSON-fil på över 80 000 rader.

```
{
  "pin": "123456789",
  "name": "Dumper",
  "destination": "SE",
  "children": [
    {
      "partNumber": 1,
      "parent": "Dumper",
      "name": "Motor",
      "children": [
        {
          "partNumber": 11,
          "parent": "Motor",
          "partName": "Block",
          "children": [
            {
              "partNumber": 111,
              "parent": "Block",
              "partName": "Cylinder",
              "children": []
            }
          ]
        },
        {
          "partNumber": 12,
          "parent": "Motor",
          "partName": "Topplock",
          "children": []
        }
      ]
    },
    {
      "partNumber": 2,
      "parent": "Dumper",
      "partName": "Hytt",
      "children": [
        {
          "partNumber": 21,
          "parent": "Hytt",
          "partName": "Vindruta",
          "children": []
        }
      ]
    }
  ]
}
```

2.5 Comma Separated Values

Comma Separated Values (CSV) är en av de vanligaste import och export filerna för kalkylblad och databaser som kan öppnas med flera olika program så som Microsoft Excel eller Google Sheets. En CSV-fil är en ren text-fil[6] vilket gör att den även kan öppnas med olika textredigerare.

En CSV-fil består av en samling av poster där varje rad representerar en ny post. Då det saknas en klar definition av en CSV-fil, följer de flesta utvecklare formatet[7]:

- Varje post är tilldelad en rad.
- Varje rad består av lika många fält separerade av kommatecken.
- Första raden kan vara en rubrikrad för att namnge de olika fälten. Rubrikraden följer samma format som övriga rader.

En CSV-fil kan ses som en tabell eftersom den organiserar data i rader och kolumner. Varje rad i CSV-filen motsvarar en post eller en rad i tabellen, och varje fält som separeras av kommatecken motsvarar en kolumn i tabellen.

CSV-filer är ett bra och enkelt sätt att organisera data utan att använda ett databasprogram. I Python finns det bibliotek som stödjer hanteringen av CSV-filer vilket gör att man både kan läsa och skriva formatet[8].

2.6 Rekursiva funktioner

Att lösa ett problem rekursivt innebär att man delar upp problemet i mindre delproblem, där varje problem är på samma form som huvudproblemet. Syftet är att varje delproblem ska vara litet och betydligt enklare att lösa än huvudproblemet. Detta görs genom att kombinera lösningarna från delproblemen vilket i sin tur löser huvudproblemet. Kanske den mest utmanande delen är att kunna identifiera och definiera ett delproblem som löser huvudproblemet [9, s.427].

Ofta inleds den rekursiva funktionen med ett basfall som avgör hurvida rekursionen ska avslutas eller inte. Ett klassiskt exempel på rekursion är beräkningen av fakulteten för ett naturligt tal, där basfallet kontrollerar argumentet som skickas till funktionen.

```
procedure FACTORIAL(n)
    if n == 0 then
        return 1
    else
        n ← n * FACTORIAL(n − 1)
    end if
    return n
end procedure
```

Då $n == 0$ returnerar funktionen 1 då $0! = 1$, annars multipliceras n med funktionen ”FACTORIAL$(n - 1)$”, vilket är ett rekursivt anrop till funktionen själv.

Andra vanliga tillämpningsområden är att man söker igenom olika datastrukturer såsom träd, där man vill söka efter ett värde, skapa en kopia eller, som i detta arbete, hitta alla maskindelar i en BOM-fil. Där behandlas barnen till varje nod som ett eget träd, vilket används för att rekursivt anropa funktionen.

```
procedure TRAVERS(Root, parts)
    for Child in Root do
        parts ← Child.partNumber
        if Children in Child then
            TRAVERS(Child, parts)
        end if
    end for
    return parts
end procedure
```

För alla barn som tillhör rotnoden sparas maskindelen till en lista för att sedan kontrollera om barnet i sin tur har några egna barn. Om så är fallet, kallas ”TRAVERS(Child, parts)” rekursivt med barnet som rotnod.

Det är vanligt att en hjälpfunktion behövs vid implementering av rekursiva funktioner. Denna hjälpfunktion kan utföra nödvändiga kontroller eller skapa argument som den rekursiva funktionen använder.

I detta exempel av en hjälp funktion skapas en tom lista som används som argument i den rekursiva funktionen ”TRAVERS”.

```
procedure TRAVERS_BOM(BOM_FILE)
    parts ← []
    return TRAVERS( BOM_FILE, parts)
end procedure
```

2.7 Python

Python är ett programmeringspråk som är väldigt populärt och används inom flera olika områden [10] [11]. Dess enkelhet, popularitet, öppna källkod och oberoende av plattform har gjort att språket har många tredjeparts-bibliotek som användare bidrar till. Flera av dessa bibliotek lämpar sig till maskininlärning och data science, där Python har blivit ett populärt val [12].

2.8 Tkinter

Tkinter är ett Python bibliotek som möjliggör användning av programvaran Tk via Python-kod. Tk är ett bibliotek för programmeringspråket Tcl (implementerat i C) och tillhandahåller ett bibliotek av element (widgets) och verktyg för att programmera dessa till ett grafiskt användargränssnitt. Tkinter är Pythons standardgränsnitt för användning av Tk och möjliggör programmering av applikationer för persondator. Tkinter använder en objekt-orienterad struktur med så kallade ”widgets”, som kan vara knappar, textrutor, fönster, menyer m.m. Dessa ”widgets” är organiserade i en hierarki där första argumentet till en ”widget” är dess förälder i strukturen. Därtill finns konfiguration av layout och dynamisk uppdatering av innehåll baserat på användarens interaktion [13].

2.9 Scikit-learn

Under projektets gång har olika Python bibliotek använts, däribland biblioteket scikit-learn som tillhandahåller olika klasser och funktioner inom maskininlärning. Scikit-learn tillandahåller verktyg för att t.ex bearbeta, klassificera, analysera och klustra olika typer av data. Det är ett populärt verktyg inom dataanalys med öppen källkod och gratis programvara [14].

2.9.1 Train_test_split

För att kunna utföra en meningsfull utvärdering av olika klassificeringsmodeller har klassen `train_test_split` använts. Den andel av datan som ska vara träningsdata specificeras som en parameter och funktionen väljer sedan slumpmässigt ut vilka datapunkter som blir träningsdata. Givet detta förhållande delas datamängden upp i en träningsdel och en valideringsdel [15].

2.9.2 CountVectorizer

CountVectorizer är en klass som hanterar textdokument och konverterar dem till en numerisk representation. För varje dokument skapas en vektor, där varje position i vektorn motsvarar en specifik term från samlingen av alla textdokument. Dessa termer bildar ett ”dictionary” som används vid skapandet av vektorerna. För att konvertera en samling dokument till vektorform räknas förekomsten av varje term i dokumentet och summeras i motsvarande element i vektorn. Om inga begränsningar har gjorts på ”dictionary” så kommer vektorernas längd att motsvara ”dictionary’s” storlek. Denna numeriska representation av dokument kan sedan användas för att träna maskininlärningsmodeller för att utföra klassificering, klustering eller annan textanalys[16].

2.9.3 TruncatedSVD

TruncatedSVD används för att minska dimensionen på numerisk data som representeras som matriser. Detta görs genom att använda singular value decomposition

(SVD), som är en metod för att faktorisera matriser. Till skillnad från principal component analysis (PCA) så är SVD mer lämpad för att hantera ”glesa matriser”, det vill säga matriser med många nollvärden[17]. Vid dimensionsreduktion tar TruncatedSVD bort de minsta singulärvärdena i SVD-faktoriseringen och på så sätt minska dimensionen på datan med minimal förlust av information, vilket kan vara användbart för att effektivisera beräkningar och förbättra prestandan hos maskininlärningsmodeller såsom t-SNE som beskrivs i avsnitt 2.9.4.

Mer information om SVD kan hittas i boken ”Matrix computations”[18].

2.9.4 TSNE

T-distributed Stochastic Neighbor Embedding(t-SNE) är ett verktyg för minska dimensionalitet och visualisera multidimensionell data. Genom att omvandla likheter mellan olika datapunkter till gemensamma sannolikheter kan datan optimeras till en lägre-dimensionell representation av punkterna. Syftet är att datapunkter som är lika i högre dimensioner också ska vara det i lägre dimensioner. För att detta ska kunna uppnås beräknas sannolikheten för att varje datapunkt används som referenspunkt för att definiera dess grannar. Därefter beräknas sannolikheten för att andra datapunkter är nära dessa referenspunkter i högre dimensioner med hjälp av t-fördelningen. Detta gör att t-SNE kan hantera komplexa icke-linjära strukturer i datan. Sedan optimeras en representation av datan i en lägre dimension genom att minimera skillnaden mellan sannolikheten i den högre och lägre dimensionerna med en gradientbaserad optimeringsalgoritm. Genom att optimera denna lägre dimensionella representation, bevaras likheterna mellan datapunkter från den högre dimensionella rymden in i den lägre dimensionella rymden. Resultatet kan sedan användas för att analyseras och visualisera datan i 2 eller 3 dimensioner. Klustren speglar likheter av datan i den högre dimensionen. Detta gör det möjligt att lättare finna mönster och likheter emellan olika datapunkter[19].

För ytterligare information om t-SNE algoritmen, se den relevanta vetenskapliga artikeln[20].

2.10 NoSQL

NoSQL syftar till de databaser som inte är relationella (data i tabellform). Istället kan icke-relationella databaser ta formen av nyckelvärden i en hashtabell, dokumentdatabaser, kolumndatabaser eller grafdatabaser. Fördelen med dessa databaser är att de kan hantera stora mängder ostrukturerad data och effektivt skalas upp i molntjänster när stora mängder ny data kommer in [21].

2.11 Microsoft Azure

Azure är en moln-tjänst som tillhandahålls av Microsoft. Plattformen tillhandahåller en rad olika verktyg och tjänster som gör det möjligt att hantera applikationer och

data i molnet. Azure används för flera olika ändamål såsom utveckling, drift och distribuering av olika applikationer och tjänster[22].

2.11.1 Azure Cosmos DB

Azure Cosmos DB är en datbasplattform för relations- och NoSQL-databaser. Azure Cosmos DB möjliggör hantering av både strukturerad och ostrukturerad data och hantering av flera olika data typer och dokument. Databasen är utformad för att kunna erbjuda skalbarhet och global tillgänglighet genom att låta användaren skala upp eller ner och flytta data över flera regioner runtom i världen[23].

2.11.2 Azure Storage

Azure Storage är en molnlagringslösning som erbjuder säkerhet, skalbarhet och tillgänlighet för att lagra olika typer av data i molnet. Tjänsten erbjuder flera olika lagrings-alternativ såsom ”Blob-storage” som är lämplig för att lagra stora mängder ostrukturerad data såsom bilder, video och olika dokument[24].

2.11.3 Azure Datafactory

Azure Datafactory (ADF) är en transformationstjänst som gör det möjligt att skapa, övervaka och schemalägga dataflöde mellan olika källor i molnet. ADF kan användas för en mängd olika ändamål så som dataflytt och kopiering av stora mängder data mellan olika molnapplikationer. Tjänsten är utformad för att vara skalbar och globalt tillgänglig vilket gör det möjligt att distribuera data mellan flera olika regioner[25].

3
Genomförande

Flera moment utfördes under arbetets gång. Till en början var arbetet fokuserat på att implementera databasen, att söka efter specifika modeller eller komponenter och att ta fram en metod för att jämföra två maskiner med ett lämpligt numeriskt värde. Sedan utfördes experiment av olika maskininlärningsmetoder för att jämföra hela datamängden på en storskalig nivå. Slutligen konstruerades en applikation för att med ett grafiskt användargränssnitt kunna markera och tydliggöra skillnaderna mellan två maskiners innehåll. Dessa olika moment beskrivs mer utförligt i styckena nedan.

3.1 Överföring av BOM-filer

För att hanter alla BOM-filerna användes Microsofts molntjänst Azure. På grund av filernas storlek och antal, laddades filerna först till Azure-blob storage och inte direkt till Cosmos DB på grund av begränsningar i databasen. En komprimerad fil laddades upp i blob-storage för att sedan packas upp på plats. För att kopiera filerna till databasen användes Azure Data Factory. Hastigheten på dataöverföringen kan justeras med värden som Data Integration Unit(DIU)[26] och Request Units(RUs)[27]. Dessa värden hade behövt justeras för att kunna kopiera hela datamängden från Blob-storage till databasen, vilket inte utfördes, utan endast en delmängd av datan fördes över till databasen.

3.2 Azure API

Ett API skapades för att göra det möjligt att kommunicera med databasen genom Python SDK. En instans av klassen CosmosClient skapas, vars constructor kräver två parametrar:

- url: slutpunktsadress för NoSQL förfrågningar.
- credential: Cosmos-nyckeln för att identifiera användaren.

Inloggningsuppgifterna sparas i privata miljövariabler i utvecklingsmiljön på författarnas personliga datorer. Dessa variabler kan sedan användas i koden för att koppla upp sig till databasen. Slutligen skapas en instans av en container-klass som representerar databasen och kan sökas med queries.

3.3 Traversering av BOM-filer

Då alla BOM-objekt i datasamlingen är JSON-filer med en trädstruktur, användes en rekursiv funktion för att lista alla olika komponenterna i en maskin. Genom en stödfunktion skapades en tom lista som sedan användes som argument i den rekursiva funktionen.

Båda funktionerna är beskrivna i pseudokod i avsnitt 2.6 i teknisk bakgrund.

3.3.1 Komponenter med innehåll av bly

Genom en metod som systematiskt söker igenom en BOM-fil och sparar dess komponenter, ges en möjlighet att fokusera på komponenter som är av särskilt stort intresse. Antalet delar i varje maskin som innehåller bly identifierades genom jämförelse med en lista på alla blykomponenter, vilket ger en möjlighet att jämföra olika maskiner sinsemellan. Antalet blykomponenter per maskin delades med antalet totala komponenter för att ta hänsyn till maskinernas storlek.

3.4 Numeriskt värde för likhet mellan två multimängder

För att jämföra ett stort antal maskiner underlättar det att ha ett numeriskt värde för varje jämförelse mellan två maskiner, istället för en lista med namn på komponenter de har gemensamt. Inom statistik och data science är Jaccards index eller Jaccards likhet ett etablerat begrepp vid jämförelse av två mängder [28]. För två mängder A och B kan Jaccards index skrivas som

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}. \tag{3.1}$$

där täljaren är antalet gemensamma element och nämnaren är totalt antal element i båda mängderna. I detta sammanhang togs beslutet att dubbletter är av intresse vid en jämförelse. Därför användes istället Jaccards index med multimängder, där ett element kan förekomma mer än en gång. I Python användes listor istället för multimängder och listor med maskiners alla delar tas fram genom att traversera BOM-filen med en rekursiv funktion.

Jaccards index fungerar på liknande sätt för två multimängder med undantaget att värdet ligger mellan 0 och $\frac{1}{2}$ istället för mellan 0 och 1. Detta förtydligas om man skriver

$$J(A,B) = \frac{|A \cap B|}{|A| + |B|}, \tag{3.2}$$

där nämnaren är ett alternativt sätt att skriva kardinaliteten av unionen för de två multimängder, vilket är summan av antalet element i vardera mängd. Om A och B är identiska ges att
$J(A,B) = J(A,A) = \frac{|A \cap A|}{|A|+|A|} = \frac{|A|}{2|A|} = \frac{1}{2}$.

Ett beslut togs att multiplicera Jaccards likhet för multimängder med 2 för att få ett mer intuitivt värde mellan 0 och 1, där en större siffra innebär mer gemensamt. För att ta fram överlappet mellan två listor beräknades först antalet förekomster av varje unik komponent i vardera maskin. I Python görs detta med klassen Counter som relaterar varje element med antalet förekomster den har i listan. Antalet förekomster de två maskinerna har gemensamt för varje komponent summeras. Det innebär att om en komponent inte finns i någon av maskinerna så bidrar den inte till summan. Om en komponent förekommer x gånger i båda maskinerna adderas x till summan. Men om en komponent förekommer x gånger i den ena maskinen och y gånger i den andra adderas det mindre av dessa två tal till summan. Det numeriska värdet som beräknas med denna Python funktion kan matematiskt skrivas som

$$2\frac{|A \cap B|}{|A| + |B|}. \tag{3.3}$$

Funktionen visas i kodexempel A.1 i Bilaga A.

3.5 Extrahera data och beräkningar

Funktionen för att jämföra två listor från kodexempel A.1 kan användas på olika urval av datan. Funktioner för att söka efter olika modeller, hytter, motorer eller specifika maskinexemplar i databasen togs fram i Python. Motsvarande funktioner gjordes för hela datamängden på lokal dator. Först identifierades de 43 olika maskinmodeller som datan bestod av. Namnet på modellen finns under ”hostingPartName” bland de första barnen till individen (parts). Ett exempel på hur man kan ta fram alla modeller i databasen visas i kodexempel B.1 i Bilaga B. Koden är ett utdrag och utelämnar importer och annat förarbete som behövs för att ha tillgång till container (databasen). Mer om detta står under avsnitt 3.2 som beskriver hur uppkopplingen till Azure utförs. Kodexempel B.2 i Bilaga B är ett exempel på sökning efter innehållet från en variabel, vilket kommer till nytta vid programmering av applikationen och att låta användaren välja intresseområde.

Ett heatmap skapades i Python för att visualisera jämförelser mellan de olika maskinmodellerna. En slumpmässig individ valdes från varje modell och jämfördes med alla andra individer, vilket resulterade i en 43x43 matris4.1. För att jämföra specifika komponenter så som hytter (cabs) eller motorer så gjordes olika sökningar för att extrahera relevant data. För motorerna tillhandahölls en lista på deras nummer (partNumber) från handledaren på Diadrom Systems AB. För hytter fick en mer omfattande sökning utföras efter namn innehållande ordet ”cab”. Efter att upprepat uteslutit irrelevanta delar, diskuterades de identifierade hytterna med handledaren för att bekräfta om den framtagna datan är rimlig. Sökningen försvårades av att hytter (och motorer) inte alltid är i första nivån av BOM strukturen. Data för motorer och hytter från databasen sparades i CSV-filer för att snabba på vissa efterföljande beräkningar. Filerna innehåller alla maskinexemplars pin, modellnamn, namn och nummer på motor/hytt och dess innehåll. Genom Python biblioteket Pandas kan filerna laddas in till en ”Data Frame” och hanteras effektivare. Kod skrevs för att

låta en användare via terminalen välja att göra jämförelser baserat på pin, modellnamn, eller komponentnamn. För en Data Frame vid namn df kan man i Python t.ex. identifiera vilken rad (eller rader) en viss pin är med koden

```
index = df[df['pin'] == pin].index.values
```

där sökning efter alla rader med ett visst modellnamn eller komponentnamn kan göras på motsvarande sätt. Då omfånget valts så utförs beräkningar med funktionen för Jaccards index som beskrivs i avsnitt 3.4. Resultatet skrivs ut i terminalen och sparas i en egen csv fil. I detta fallet hade beräkningarna för alla maskiner i databasen gjorts på förhand och sparats i en matris.

En delmängd av all data överfördes till moln databasen och flera olika funktioner skapades i Python för att kunna koppla upp sig och göra olika sökningar bland datan. Detta möjliggjorde en funktion som låter användaren skriva in en PIN för en maskin och få fram en jämförelse mellan dess hytt och alla andra maskiners hytter i databasen. Jämförelsen är baserad på Jaccards index och beskrivs i avsnitt 3.4. Namn och värden sparas i en CSV-fil och sorteras från mest gemensamt till minst gemensamt.

3.6 CountVectorizer

För att få en numerisk representation och att kunna utföra olika typer av matrisoperationer på datan användes klassen CountVectorizer. Genom att traversera hela datasetet och låta varje maskin representeras av en textsträng av alla dess parts kunde sedan en numerisk representation av datan på matrisform skapas även kallat Document Term Matrix(DTM).

3.7 Träning och validering

För att kunna utvärdera en maskininlärningsmodell är det viktigt att dela upp datamängden i en träningsdel för att träna modellen och valideringsdel för utvärdering. Att modellen inte haft tillgång till valideringsdatan är nödvändigt för att kunna göra någon form av korrekt utvärdering av modellens prestanda. För att dela upp datan valdes ett förhållande mellan tränings och valideringsdatan på 80/20, vilket är en vanlig uppdelning bland flera andra alternativ [29]. Standardvärde för train_test_split funktionen som användes från scikit-learn är 75/25, vilket ändras med en parameter till funktionen [15].

3.8 DecisionTreeClassifier

Klassificering är en vanlig metod inom maskininlärning. Därför tränades ett beslutsträd på datan för att kunna förutspå maskinmodell baserat på en maskins komponenter. Datan vektoriserades med CountVectorizer och delades upp i träningsdata och valideringsdata med train_test_split. Sedan tränades beslutsträdet DecisionTreeClassifier från scikit-learn på datan. Det färdigtränade beslutsträdet

testades sedan på valideringsdatan för att ta fram hur bra den klassificerar maskinmodell ut efter dess komponenter. Beslutsträdets attribut ”feature importance” sorterades efter storlek och sparades.

3.9 t-SNE

För att ha möjlighet att plotta data med hjälp av t-SNE behövde datan först vektoriseras. Genom att låta varje maskin representeras av en sträng av alla dess komponenter kunde hela datamängden vektoriseras med hjälp av klassen CountVectorizer som beskrivs i avsnitt 2.9.2. För att reducera dimensionalliteten på datan användes SVD som beskrivs i avsnitt 2.9.3, vilket är rekommenderat eftersom datan är på gles matrisform. För att kunna identifiera de olika datapunkterna utefter modell skapades en separat lista med samtliga modeller, där varje modell tilldelades en färg för att lättare kunna utvärdera diagrammet och att kunna skilja de olika datapunkterna åt.

3.10 Applikation

Applikationen konstruerades för att jämföra två valbara maskinexemplar och deras innehåll i detalj. En startsida (startPage) konstruerades för att välja maskiner och en jämförelsesida (listPage) skapades för att lista innehållet och färgkoda skillnader mellan de två maskinerna.

3.10.1 Beräkningar för startsidan

För att välja en specifik individ från databasen så krävs dess pin nummer. En funktion skapades för att söka efter en maskin i databasen baserat på pin och sedan returnera dess data. Kod för detta visas i kodexempel C.1 i Bilaga C. Startsidan låter användaren välja två pins genom två olika ”Entry widgets” som uppdaterar dess sparade innehåll när användaren skriver in ny text. En knapp för att bekräfta valet skapades och om inmatningen inte får någon sökträff så behålls ett standardval eller tidigare träffar.

Vid ett bekräftat val som existerar så påbörjas beräkningar för att ta fram skillnaderna mellan de två BOM filerna. Differensen mellan två mängder A och B är de element som finns i A men inte i B (A\B). Genom att göra om de två BOM filerna till två mängder så kan A\B och B\A räknas ut i Python med den inbyggda funktionen ”difference” för mängder (set). En funktion skapades som givet två listor A och B returnerar en mängd A\B, som i fortsättningen refereras som diff_set.

Därefter skapas listor med alla namn och nummer på komponenterna i första nivån av BOM filerna. En funktion (create_first_level_list) skapades för att loopa igenom första nivån av datan och returnera det som en lista. Dessa jämförs med varandra för att identifiera vilka delar i första nivån som inte finns hos den andra. Index för delarna som endast finns i den enda maskinen sparades i två ny listor vid namn

main_diff1 och main_diff2. Ett kort exempel kan ses i kodexempel C.2 i Bilaga C, där indexet för delarna i maskin 1 som inte finns i första nivån av den andra maskinen sparas.

Sedan skapades en funktion för att traversera datan och identifiera vilka delar i första nivån som har barn som inte finns i den andra maskinen. På samma sätt så sparas dessa delars index i en ny lista. Funktionerna traverse_search och traverse_search_children kan ses i kodexempel C.3 och C.4 i Bilaga C. Eftersom datan kan komma från olika nivåer i maskinen så måste funktionen kolla om det är maskinens första nivå ('parts') eller om den givna datan är längre ner i strukturen ('children'). Argumentet search_keys jämförs mot datan och om en maskindels nummer finns bland delarna i search_keys så sparas indexet för delen i högsta nivån av strukturen. Genom att sätta search_keys lika med diff_set så identifieras de delar som inte finns i den andra maskinen. Denna information utnyttjas senare för att färgkoda maskinernas innehåll.

3.10.2 Widgets för jämförelsesidan

Applikationen består av två Frames, startPage och listPage (jämförelsesidan), som ligger på varandra. För att växla mellan sidorna används funktionen frame.tkraise(). Genom att ge en knapp en ”callback function” (via command argumentet) så anropas denna funktion när användaren klickar på knappen. Ett exempel på hur en knapp växlar tillbaka till startsidan visas i kodexempel C.5 i Bilaga C, där knappens rot är listPage, och startPage skickas med som argument till funktionen raise_frame genom en Python lambda funktion.

För att visa de två maskinernas komponenter i deras olika nivåer används två stycken ListBox widgets bredvid varandra. De tillåter användaren att skrolla igenom en given lista och välja ut ett namn i listan. Olika Label widgets används för att visa modellnamn, pin, nuvarande maskindel, nuvarande position i strukturen och nuvarande nivå för de respektive listorna. Vissa ”labels” uppdateras genom att anropa funktionen label.config(text=”nytt innehåll”) när en ny maskin eller delkomponent valts. Knappar för respektive lista skapades för att redigera färgmarkeringar och navigera igenom maskinens struktur. Layout för alla widgets ordnas med Tkinters ”grid” funktion. Sidan tilldelades 6 rader och 6 kolumner för att positionera de olika widgets:en i ett sorts rutnät. Storleken på de olika rutorna, hur många rutor en widget tar upp, positionering inuti rutan och deras förhållande när fönstrets storlek ändras är alla aspekter som konfigureras vid skapandet av varje widget.

3.10.3 Navigation och färgkodning för jämförelsesidan

Vid val av maskin eller delkomponent så fylls ListBox in med innehållet från dess första nivån. Jämförelserna beräknade från diff_set används för att färgkoda delarna. Om en maskindel inte finns i den andra maskinen och den har minst en delkomponent som inte heller finns i den andra maskinen tillges namnet i listan färgen lila.

Om endast en av dessa stämmer tilldelas färgen blå respektive röd. Men om ingen stämmer tilldelas delen en grön färg. En motsvarande funktion skapades för att fylla in listan men utan att inkludera några gröna namn. En knapp för respektive lista skapades för att växla mellan att inkludera eller exkludera gröna namn i listan. Varje gång listan uppdateras så rensas alla namn innan nytt innehåll fylls in. ListBox möjliggör att en funktion kallas varje gång en del i listan valts. Funktionen som skapades identifierar vilken individ som valts och dess index i listan för att kunna göra nytt innehåll och nya jämförelser. För att kunna navigera i BOM strukturen sparas nuvarande position i form av en lista av index. En funktion skapades för att kunna ta fram alla barn för en viss maskindel, givet dess position i strukturen. Sedan kan samma jämförelser som beskrevs för startsidan utföras med det nya innehållet. Dessa processer upprepas varje gång användaren går fram eller bak i strukturen. Flera olika variabler och funktioner skapades för att hålla ordning på datan och uppdatera sidan med korrekt innehåll.

4

Resultat

Här presenteras de resultat som har uppnåtts under arbetets gång.

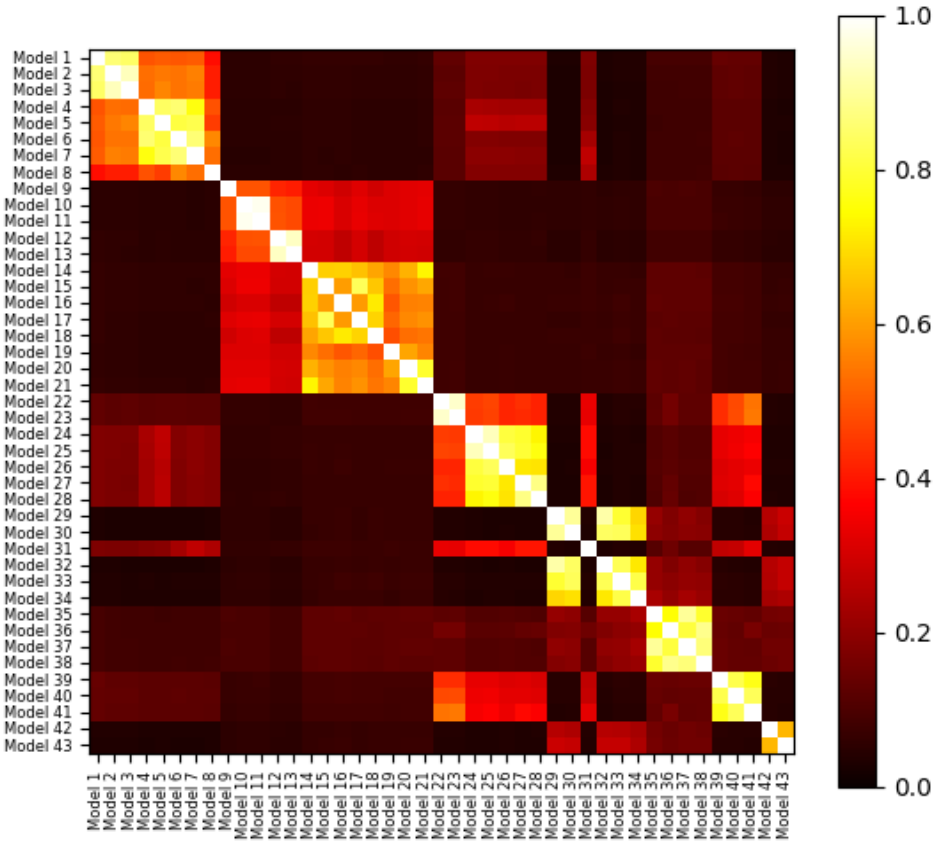
4.1 Extrahering av Komponentinformation från BOM

Efter traversering av BOM-filerna erhölls information om hur många komponenter varje maskin består av totalt, antal unika komponenter och antalet av varje maskindel i vardera maskin. Genom att systematiskt söka igenom varje fil kunde alla komponenter listas. Även namnen på alla maskinmodeller i datamängden kunde identifieras. En matris över hela datamängden skapades med information om antal av varje maskindel per maskinmodell. Traverseringen av datan har givit en kvantifierbar bild av maskinernas sammansättning, både individuellt och som helhet i datamängden. Detta har lagt grunden för vidare arbete med analys och jämförelse av datan.

4.2 Jämförelser mellan maskiners komponenter

En maskin av varje modell valdes godtyckligt och jämfördes sinsemellan med beräkningar baserade på Jaccards index som beskrivs i avsnitt 3.4. Detta visualiserades med en heatmap som kan ses i figur 4.1 där färgen representerar värdet från beräkningarna. Skalan är mellan noll och ett där ett innebär identiska maskiner och noll innebär att maskinerna inte hade en enda komponent gemensamt. Maskinerna är sorterade i bokstavsordning efter deras modellnamn. Detta skapar grupper eller kluster i heatmap:en eftersom maskiner med liknande innehåll ofta har liknande namn.

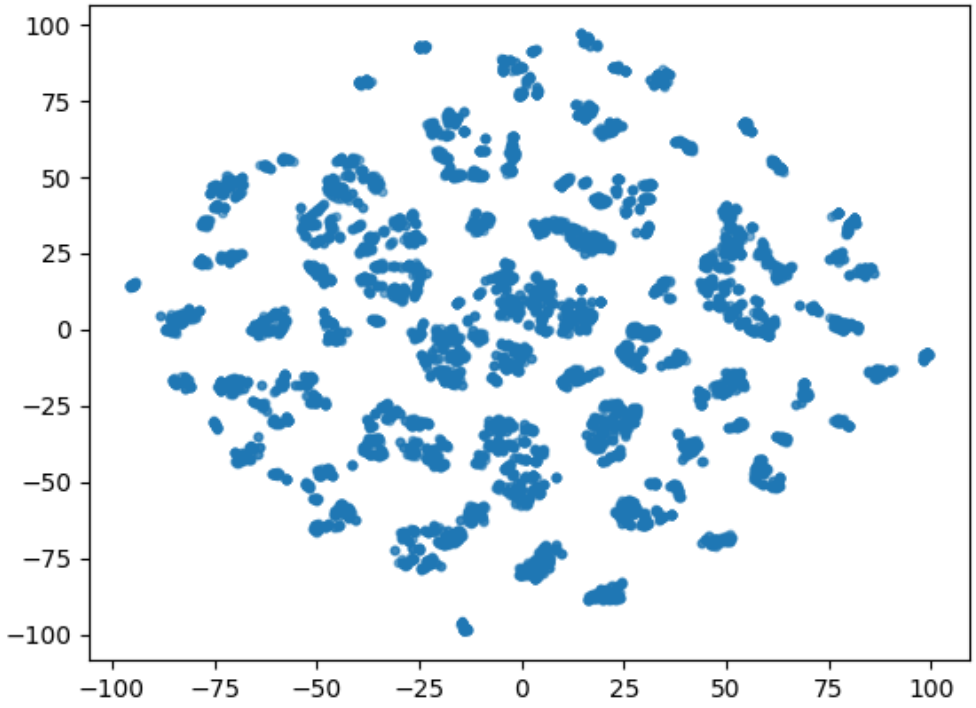
Denna visualisering av datan ger en möjlighet att identifiera liketer och skillnader mellan maskiner på komponentnivå, där ett värde nära 1 på heatmappen indikerar att maskinerna som jämförs har ett högre antal gemensamma komponenter medans ett värde närmare 0 indikerar på en lägre grad av likhet.



Figur 4.1: Heatmap med en godtycklig maskin från varje modell. 1 representerar 100% likhet och 0 representerar 0% likhet vid jämförelse av maskinernas komponenter

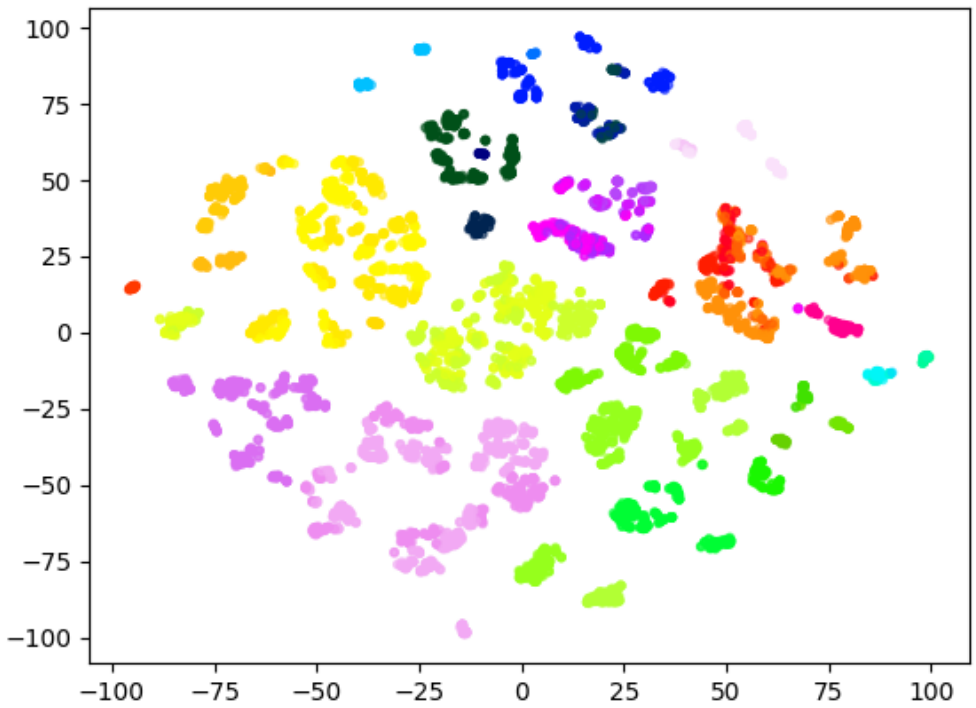
4.3 2-dimensionell representation av datamängden

En numerisk representation av av datamängden skapades med CountVectorizer. Dimensionaliteten reducerades till 100 med SVD (beskrivs i avsnitt 2.9.3) för att sedan ytterligare minska på dimensionaliteten ner till 2 med t-SNE (som beskrivs i avsnitt 2.9.4) vilket gav en möjlighet att visualisera datan med en graf som visas i figur 4.2. Baserat på varje maskins innehåll av komponenter så beräknas ett tvådimensionellt värde av t-SNE. Det är dessa värden för varje maskin i datamängden som markeras i figur 4.2.

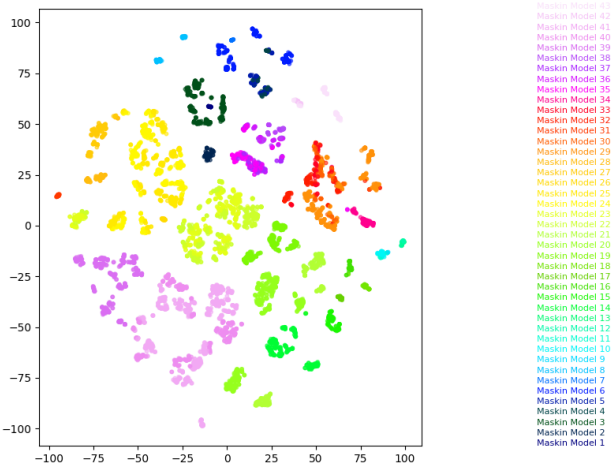


Figur 4.2: 2D representation av samtliga maskinexemplars innehåll utan färgkodning. Varje punkt representerar en maskin och är ett värde beräknat med t-SNE, baserat på maskinens komponenter.

Därefter färglades datapunkterna efter vilken modell varje maskin tillhör, vilket innebär att varje maskinmodell i plotten representerades av en unik färg. Denna kan ses i figur 4.3 nedan. Detta möjliggjorde identifiering av olika mönster och kluster inom plotten. Maskinmodellernas tillhörighet tydliggjordes genom en färgskala där samtliga maskinmodeller är representerade, vilket kan ses i figur 4.4. Detta gjorde att det gick att skilja de olika datapunkterna åt baserat på modell. Dock är modellerna anonymiserade i denna figur.



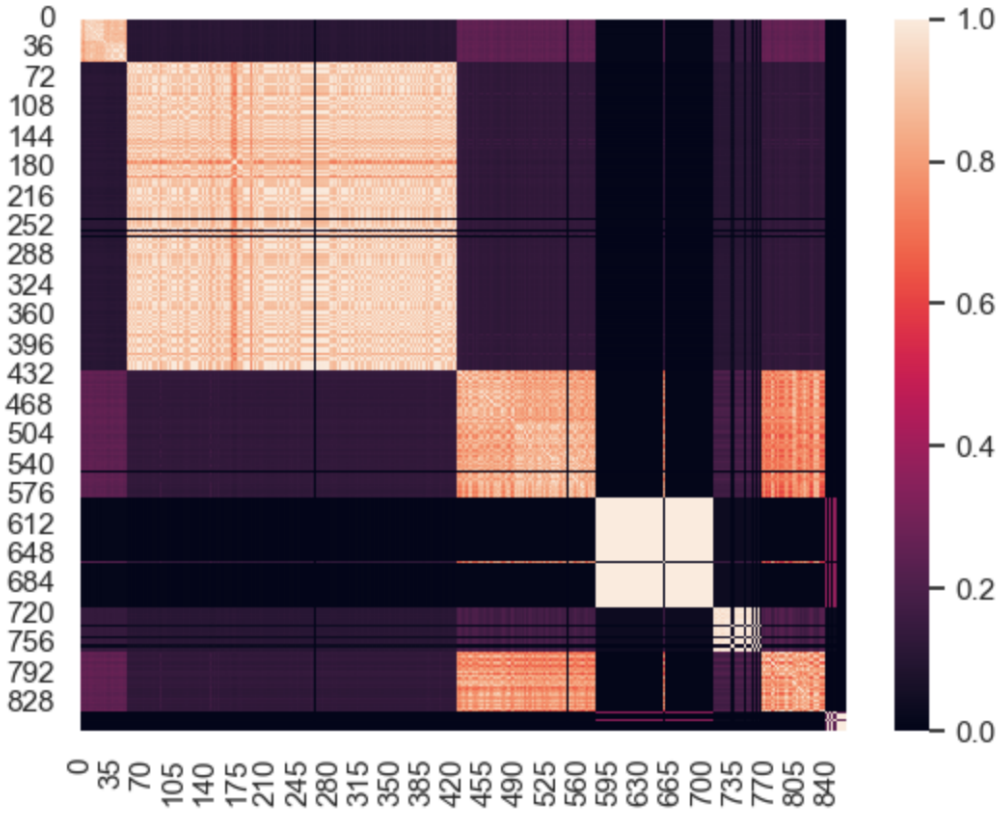
Figur 4.3: 2D representation av samtliga maskinexemplars innehåll med färgkodning baserat på modell. Varje punkt representerar en maskin och är ett värde beräknat med t-SNE, baserat på maskinens komponenter.



Figur 4.4: 2D representation av samtliga maskinexemplars innehåll med färgkodning baserat på modell. Anonymiserade modellnamn och deras motsvarande färg visas till höger i figuren.

4.4 Jämförelser mellan hytter från databasen

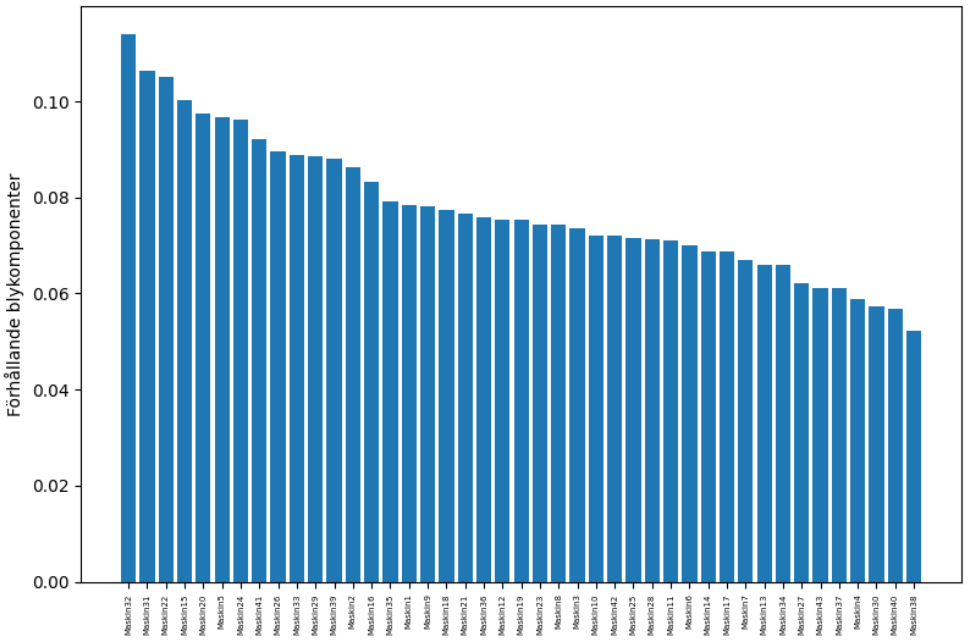
Alla hytter i databasen jämfördes med varandra för att skapa en matris i form av en heatmap där det numeriska värdet för gemensamhet reglerar färgen. En heatmap av hytternas jämförelser kan ses i figur 4.5 där hytterna är sorterade i bokstavsordning enligt modellnamn. För att anonymisera datan är hyttnamnen ersatta med siffror.



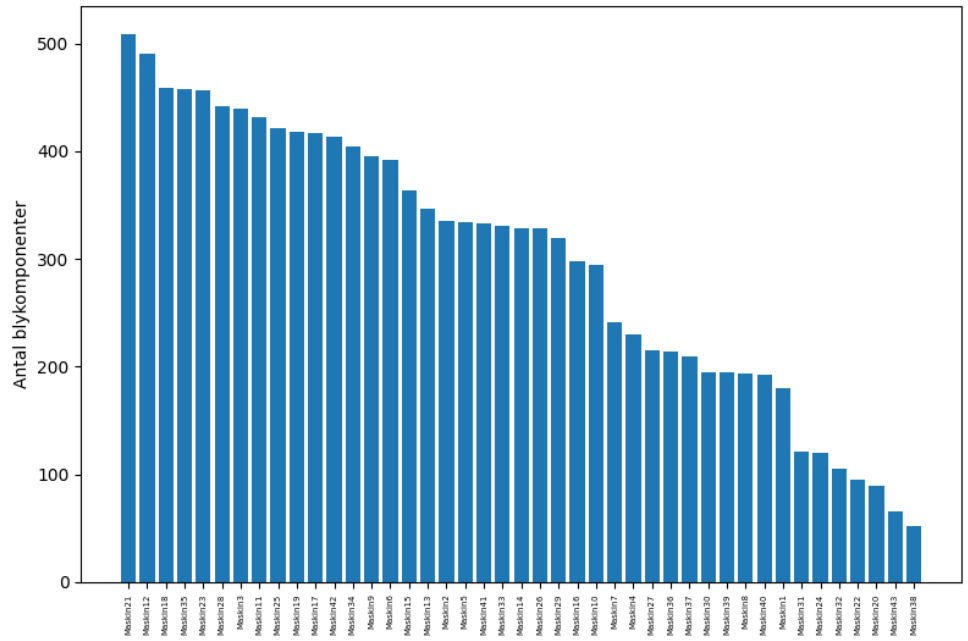
Figur 4.5: Heatmap av Jaccards index mellan alla hytter i databasen. Hytterna är sorterade i bokstavsordning enligt modellnamn

4.5 Blykomponenter

Efter att ha fått information om vilka komponenter som innehåller bly, kunde datan genomsökas. Det totala antalet komponenter och det totala antalet blykomponenter för ett urval av maskiner beräknades. Förhållandet mellan dessa kan ses i stapeldiagrammet i figur 4.6 nedan, där y-axelns värde är antalet blykomponenter delat på det totala antalet komponenter. Maskinerna i x-axeln är anonymiserade och är ett urval av maskiner med en individ från varje modell. Ett motsvarande stapeldiagram skapades med endast det totala antalet blykomponenter i y-axeln. Detta kan ses i figur 4.7.
Om en maskin har en högre andel blykomponenter finns det goda skäl att göra en



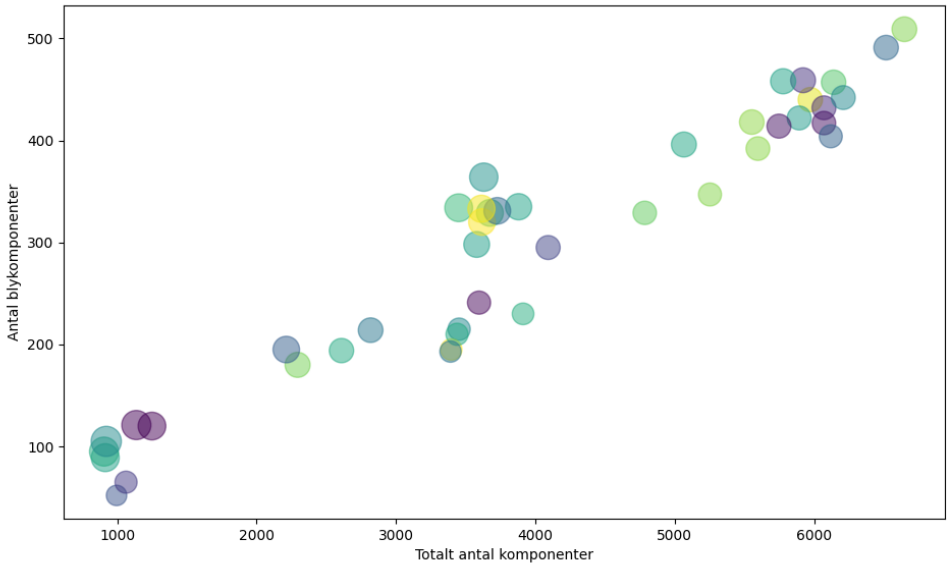
Figur 4.6: Förhållandet mellan antalet blykomponenter och det totala antalet komponenter för ett urval av maskiner. Y-axelns värde är antalet blykomponenter delat på totala antalet komponenter.



Figur 4.7: Totala antalet blykomponenter per maskin

vidare undersökning. På detta sätt kan maskinens miljöpåverkan eventuellt minskas.

För ytterligare visualisering av blykomponenter skapades ett spridningsdiagram som kan ses i figur 4.8, där x-axeln representerar det totala antalet komponenter och y-axeln antalet blykomponenter och varje punkt i diagrammet representerar en unik maskin.



Figur 4.8: Förhållandet mellan det totala antalet komponenter och blykomponenter.

Utifrån den aktuella datamängden framträder ett linjärt mönster, vilket tyder på att antalet blykomponenter är i direkt korrelation med det totala antalet komponenter. Dock kan man urskilja en avvikelse i grafen runt 3 500 komponenter där antalet blykomponenter tycks vara relativt högt, vilket kan vara av intresse för vidare analys.

4.6 Klassificering med beslutsträd

Alla delar i datamängden vektoriserades och tränades på ett beslutsträd som efteråt kan förutspå maskinmodell baserat på komponenter som indata. DecisionTreeClassifier från scikit-learn klarade av att förutspå maskinmodell med 99% noggrannhet. Beslutsträdet lyckades att hitta unika komponenter för var och en av de olika maskinmodellerna i träningsdatan. Det färdigtränade beslutsträdet gör sina prediktioner efter en unik komponent från varje maskinmodell. Exempel på dessa komponenter kan vara en programvara eller en maskinhytt som endast återfinns i en modell.

4.7 Applikation

En enkel applikation med grafiskt användargränssnitt skapades där användaren kan välja två maskinexemplar från databasen baserat på pin och sedan jämföra deras innehåll i detalj. Applikationen visar vilken nivå i strukturen som användaren befinner sig på, alla maskindelar på den nivån och namnet på maskindelen som visas för tillfället. Innehållet för den nuvarande nivån i vardera maskin visas i form av två listor av namn på delarna (partName och partNumber) som användaren kan skrolla igenom. Användaren kan välja att gå djupare i strukturen genom att klicka på en maskindels namn eller gå tillbaka en nivå med en bakåt knapp för vardera maskin. Maskindelarna är färglagda för att visa om de inte existerar i den andra maskinen och om delen har någon underkomponent som inte finns i den andra maskinen. Om båda scenariorna är sanna så får maskindelen en lila färg, om endast det första scenariot är sant får delen en blå färg, om endast det andra scenariot är sant får delen en röd färg och om ingen av de två alternativen är sanna får delen en grön färg. Delar med grön färg finns alltså i den andra maskinen och alla dess underkomponenter finns också i den andra maskinen. Om dessa delar är ointressanta för användaren kan man växla mellan att exkludera dem eller inkludera dem med en ”toggle green” knapp. Det finns även en knapp till respektive maskin för att återgå till första nivån av strukturen och det finns även en knapp för att återgå till startsidan där man kan skriva in nya pin för att jämföra andra maskiner.

5
Diskussion

I detta kapitel diskuteras resultaten av arbetet och hur de förhåller sig till projektets mål och syfte. Olika observationer görs angående resultatens relevans till syftet och eventuella förbättringar diskuteras. Kapitlet tar även upp hur delar av arbetet hade kunnat utvecklas och byggas vidare på.

5.1 Diskussion kring databasen

Valet av en NoSQL databas från Microsoft Azure togs i samråd med handledaren på Diadrom då det fanns möjligheter att skapa konton via företaget och smidigt kunna skapa en moln databas. NoSQL tycks passa den typen av data som hanteras, där strukturen hos de tusentals json dokumenten var okänd för författarna i förhand. Ifall en relationell databas hade kunnat utföra sökningar snabbare eller kunnat effektivisera jämförelser mellan specifika komponenter är utanför författarnas vetskap. Vid överföring av filer från Blob storage till databasen med hjälp av en data factory så påverkar värderna hos DIU[26] och RU[27] förflyttningens hastighet. Eftersom en ökning av dessa värden kostar pengar, eftersom det saknades kunskaper kring eventuella lösningar i Microsoft Azure och för att arbetet inte skulle dra ut över tiden så togs beslutet att endast överföra en liten delmängd av alla filer. Det fanns fördelar i att kunna påbörja sökningar och jämförelser snabbt och få en övergripande bild av datan. Databasen kunde kopplas till applikationen och demonstrera fördelar med den typen av användargränsnitt och insikter den kan ge. Nackdelen med att inte ha förflyttar alla filer till databasen är osäkerheten kring hurvida samma analyser hade kunnat utföras under en rimlig tid med tusentals filer i databasen. Analyser med maskininlärning på hela datamängden utfördes istället på en lokal dator, skilt från databasen.

5.2 Säkerhet kring Microsoft Azure

För sökningar i en databas i en applikation där användaren får mata in egna sökord är en så kallad SQL injektion en stor säkerhetsrisk. Vad som i vissa sammanhang kallas ”prepared statement”[30] utförs i Azure Cosmos databaser med parametriserade sökningar. Parametisering av en SQL query för t.ex funktionen get_pin() i kodexempel C.1 i Bilaga C ska förhindra skadliga SQL injektioner enligt Microsofts egen dokumentation [31]. För att inte läcka några inloggningsuppgifter till Azure Cosmos databasen så sparas dessa som variabler i utvecklingsmiljön under arbetets gång, istället för synlig text, ifall koden skulle publiceras eller delas med andra. Vid

skapandet av en fullskalig applikation hade ett system där användare kan logga in behövt implementeras, vilket är utanför arbetets omfattning. I nuläget kan applikationen endast köras från en utvecklingsmiljö med korrekta variabler för inloggning, vilket duger för arbetets målsättning, där syftet endast är att undersöka ifall de framtagna verktygen i programmet kan ge intressanta insikter till datan.

5.3 Diskussion kring Jaccards index

Python har enkla, inbyggda funktioner för att skapa mängder (set) av listor och sen räkna ut snittet eller skillnaden mellan dessa, vilket är något som utnyttjas till vissa beräkningar i applikationen. Nackdelen med mängder i detta fallet är att man inte tar hänsyn till skillnader i antalet av varje delkomponent, vilket ansågs vara relevant i detta fallet. Om en maskin har x antal av en viss maskindel och en annan maskin har 10x av samma del så är denna skillnad utav intresse. Man skulle kunna ta det totala överlappet mellan två maskiner som standard värde för gemensamhet. Men då tas inte storleken på varje maskin i åtanke. Om två små maskiner har nästintill exakt samma delar, så att överlappet är x stycken, så är det missvisande om en mycket större maskin med mycket fler maskindelar också har x stycken i överlapp med de mindre maskinerna. Därför valdes beräkningarna i avsnitt 3.4, för att ha det numeriska värdet för gemensamhet som en kvot mellan antalet överlapp och summan av det totala antalet delar hos de två maskinerna. Kvoten får multipliceras med två för att spannet ska vara mellan noll och ett. Det ska också noteras att värdet som beräknas i avsnitt 3.4 inte tar hänsyn till maskinens struktur eller om vissa maskindelar har mer betydelse än andra och därför borde vara en större faktor till beräkningarna. Även om några få experiment genomfördes där delar jämfördes per nivå i strukturen så lyckades inga meningsfulla mått tas fram som tar hänsyn till detta.

5.4 Ur ett Product Lifecycle Management perspektiv

Sammansättningen av varje maskin skapar en tydlig bild på komponentnivå. Det gör det möjligt att förbättra, hantera och optimera en produkts livscykel, som tillverkning, underhåll och slutligen återvinning av produkten. Detta perspektiv möjliggör att designa en mer effektiv tillverkningsprocess och optimera användningen av råmaterial. Om man besitter information om en maskins komponenter blir det också lättare att planera underhåll och service. Det i sin tur kan bidra till förlängd livslängd och minskning av driftstörningar. Mer detaljerade sökningar av komponenter kan även ge en insikt i vilka komponenter som har störst miljöpåverkan. Detta kan i sin tur ge möjligheter att söka efter alternativa material för att minska miljöpåverkan och att minska arbete och kostnader vid demontering och återvinning av maskinen.

5.5 Miljö och etik

I projektet har alla komponenter som innehåller bly identifierats och jämförelser mellan olika maskiner och modeller har utförts med blyinnehåll som perspektiv. Denna information kan användas för att identifiera de maskiner med störst potential att minska mängden bly. Detta skulle kunna utvecklas till att omfatta andra skadliga råvaror och ämnen. Analysen av farliga komponenter kan användas vid tillverkning av nya maskiner för att på ett aktivt sätt försöka minimera eller fasa ut dessa, genom att skapa nya komponenter som kan användas för att ersätta de med negativ miljöpåverkan eller hög toxicitet. Höga halter av bly i kroppen är giftigt och metoder för att hantera eller minska bly kan reducera farliga utsläpp [32]. Att proaktivt arbeta med dessa miljöfrågor resulterar inte bara i en förbättrad miljö, utan leder också till besparingar när det gäller avfallshantering och andra miljörelaterade kostnader. Dessutom kan ett företags anseende höjas genom att det aktivt arbetar med att minska användningen av farliga ämnen.

5.6 Analys av kluster

I heatmap av olika maskinjämförelser i figur 4.1 och den 2-dimensionella representation av datan i figur 4.2 går det att urskilja vissa mer eller mindre tydliga kluster. Att sortera i bokstavordning och färglägga i bokstavsordning skapade tydliga mönster i graferna, eftersom modeller med lika namn oftast också har liknande innehåll. Detta gjorde det lätt att identifiera vilka modeller som är mest lika varandra. Det är dock viktigt att notera att likheten endast är baserad på komponenter och ingen hänsyn tas till strukturella skillnader.

5.6.1 t-SNE plotten

Genom färgsättning av datapunkter efter maskinmodell ges det en tydlig bild hur modellerna bildar olika kluster. Det är också tydligt vart de olika modellerna hamnar i relation till varandra, där kluster som ligger närmare varandra är ett tecken på att de har en större andel gemensamma komponenter mot de som ligger längre ifrån varandra.

Färgsättning av de olika maskinmodellerna har gjorts utifrån en sorterad lista där likheter i namn också avspeglar sig tydligt i grafen. Maskiner med liknande namn delar oftast flera komponenter, och detta bidrar i sin tur till att förstärka effekten av flera av klustrerna i plotten beräknad av t-SNE algoritmen4.2.

För att få en djupare förståelse för klusterbildningen kan det vara av intresse att utföra en mer ingående analys av ursprungsdatan. Genom att undersöka de kluster som bildats och de modeller som tillhör varje kluster, kan man undersöka möjliga faktorer som bidrar till deras likheter. Sådana faktorer kan innefatta maskintyp, användningsområde och information relaterad till maskinens produktion och dess olika komponenter.

5.6.2 Heatmap av jämförda maskiner

Visualisering av datan genom en heatmap ger en möjlighet att identifiera liketer och skillnader mellan maskiner på komponent nivå, där ett högre värde på färgskalan indikerar ett högre antal gemensamma komponenter medans ett lägre värde indikerar på en lägre grad av likhet.

Även i heatmapen är maskinmodellerna sorterade efter namn, vilket förstärker klusterbildningen runt diagonalen. I figur 4.1 kan man t.ex. tydligt urskilja att Model1, Model2 och Model3 bildar ett tydligt kluster som avviker mot klustret som bildas av Model4, Model5, Model6 och Model7.

5.7 Insikter från beslutsträdet

Arbetsmetoden som användes på datan är ganska vanlig inom maskininlärning där olika vektorer av indata tilldelas en klass (label) och sedan tränas en lämplig maskininlärningsmodell på denna data i form av X och Y, invärde och utvärde. Detta var därför en logisk startpunkt för projektets undersökningar där målsättningen var att testa olika maskininlärningsmetoder och se om de kan utföra jämförelser på hela datamängden. Att klassificera datan är dock inte helt meningsfullt när målet med arbetet är inriktat mot att identifiera skillnader och likheter. Det finns dock insikter från klassificeringen som är värda att diskutera. Fördelen med beslutsträd och anledningen till att det valdes som maskininlärningsmodell är att deras resultat är lättare att tolka. Vetskap om varför beslutsträdet relaterar en grupp maskindelar till en viss modell kan möjligtvis ge insikt i modellens mest kännetecknande attribut. D.v.s att man undersöker vilka maskindelar som beslutsträdet la störst vikt vid i sitt beslut. Detta gjordes med attributet ”feature importance” till DecisionTreeClassifier i scikit-learns bibliotek. Beslutsträdet lyckas oftast identifiera maskindelar som är unika för respektive modell. Detta kan vara intressant i sig, men ett alternativ skulle vara att utesluta unika maskindelar och se vad beslutsträdet grundar sina beslut på i såna fall. Andra ändringar skulle kunna vara begränsningar i träddjupet och hantering av obalans bland klasserna (modeller har olika antal individer i indatan).

5.8 Förbättringar till applikationen

Vid slutförandet av rapporten innehåller applikationen en del kända buggar relaterade till växling av grönmarkerade delar och byte av maskiner. Inga tester har utförts för att identifiera problem eller kontrollera att funktioner gör det de ska. Koden följer inte några objekt-orienterade principer och är i allmänhet ostrukturerad. Allt detta skulle behöva förbättras om man vill bygga ut applikationen och dess funktionaliteter.

För presentationen av maskinernas innehåll på jämförelsesidan finns det flera förbättringsområden. Man skulle kunna sortera delarna i listan efter t.ex bokstavsordning eller baserat på storlek (antal underkomponenter per maskindel). Att låta

användaren välja hur listan ska sorteras och möjlighet att söka i listan är också användbara egenskaper. Den valda maskindelen skulle också kunna presenteras med mer information än dess namn och tillhörande modell. Förslagsvis kan totalt antal underkomponenter visas, antal blykomponenter, utsedd marknad eller annan intressant information visas när användaren klickar på en maskindel. Om blykomponenter hade inkluderats i applikationen hade dessa också kunnat färgmarkeras. I nuläget kan endast grönmarkerade maskindelar exkluderas eller inkluderas. Men att växla innehållet av andra färger kan också vara intressant. Att markera skillnader och likheter med färger är inte nödvändigtvis det tydligaste, så att markera delarna med symboler bredvid deras namn kan övervägas. Eventuellt att man markerar med både färg och symboler. Oavsett val behöver applikationen en förklaring till vad markeringarna betyder. Det ska också nämnas att så som applikationen är utformad just nu så gynnas jämförelser av maskiner med små skillnader. ”Toggle green” knappen kan exkluderar alla gemensamma delar, vilket är hjälpsamt när dessa är många och man vill fokusera på skillnaderna. Men om man istället har två maskiner med stora skillnader och vill fokusera på deras likheter, så hade en ”toggle blue” knapp varit mer praktiskt för att exkludera alla delar i listan som skiljer maskinerna åt.

Applikationen skulle kunna inkludera andra funktionaliteter som skapats till detta projekt. En ny sida skulle kunna jämföra ett maskinexemplar med alla andra baserat på funktionen som beskrivs i avsnitt 3.4 och presentera resultatet i en sorterad tabell. Program för att jämföra hytter och motorer på detta vis har redan skapats och skulle därför kunna integreras i applikationen. Detta skulle ge användaren möjlighet att först identifiera vilka maskinexemplar hos en modell som är mest intressanta att jämföra i detalj (t.ex om det finns relativt stora skillnader mellan två individer inom samma modell) för att sedan använda jämförelsesidan på de två valda modellerna. Jämförelserna skulle också kunna vara mellan specifika delar i maskinen, så som hytter eller motorer. En annan funktionalitet är att kunna ta fram maskindelar av stort intresse automatiskt utan att användaren behöver leta igenom BOM-strukturen manuellt. Applikationen skulle kunna identifiera motorns eller hyttens position i strukturen och ta användaren dit automatiskt i jämförelsesidan.

Slutligen behöver sättet maskinerna jämförs diskuteras. Istället för att identifiera vilka underkomponenter i den första nivån av en maskindel som inte finns någonstans i den andra maskinen, så hade man kunnat välja att endast jämföra delar i motsvarande nivå. Detta görs när två nya maskiner laddas in i jämförelsesidan, eftersom det kan vara intressant att se hur deras första nivåer förhåller sig till varandra. Svårigheten med detta är att strukturen inte alltid är samma mellan olika modeller eller t.o.m inom modeller. T.ex kan motorer vara placerade i olika nivåer av trädet. Ordningen delarna listas för varje nivå är inte heller konsekvent. Dock är fördelen med att jämföra motsvarande nivåer med varandra just att dessa strukturella skillnader kan identifieras. I nuläget kontrolleras om en del inte finns någonstans i den andra maskinen, vilket markeras med en blå färg. Detta hade eventuellt kunnat kombineras med att ta fram motsvarande nivå i den andra maskinen och jämföra endast dessa nivåer.

5.9 Utvärdering av planering och tillvägagångsätt

Det fanns inte så tydliga mål eller idéer i början av arbetet för vilka metoder som skulle användas för att jämföra filerna. Eftersom innehållet i filerna är sekretessbelagt så fanns det en osäkerhet kring vilken typ av data som det handlar om. Det kan vara därför som flera olika mål skapades, såsom att jämföra hytter eller att skapa en applikation, eftersom det fanns en osäkerhet kring hur meningsfullt resultatet av jämförelserna skulle bli. I efterhand framstår arbetet som lite för brett. Det kanske hade varit bättre att dyka djupare ner i metoder baserade på maskininlärning eller att endast arbeta med databasen och applikationen. Tidsplanen från planeringsrapporten följdes. Men p.g.a arbetets bredd så blev det svårare att komma fram till meningsfulla insikter inom de olika delområdena. När arbetet med databasen fastnade så fanns det en press att gå vidare till nästa steg i planeringen.

Litteraturstudien hade kunnat vara mer ingående. Men åt andra sidan så identifierades föga relevant material under den tiden som efterforskning gjordes. Arbetsmetoden kunde ha involverat mer reflektion av resultatet. Författarna upptäckte förbättringsområden bland jämförelserna, hur datan plottades och applikationen väldigt sent i arbetet när rapporten skulle skrivas och resultatet skulle reflekteras över. Det hade troligtvis varit bättre att diskutera förbättringsområden oftare med varandra och kanske stanna upp för att fördjupa sig i områden. Arbetet delades ofta upp mellan de två författarna för att jobba mer effektivt. Det har troligtvis haft en negativ påverkan på hur vi reflekterat över våra resultat.

6
Slutsats

I början av arbetet erhölls en stor datamängd av BOM filer för anläggningsmaskiner som delvis hanterades med en molndatabas, men mestadels på lokal dator för att testa maskininlärningsalgoritmer på hela datamängden. Genom att räkna antalet av varje maskindel per maskinexemplar skapades en matris med CountVectorizer, vars dimensionalitet kunde reduceras med SVD och t-SNE. När varje maskin plottades utifrån deras värden från t-SNE kunde tydliga kluster identifieras baserat på vilken modell de tillhör. Denna metod för att jämföra maskinerna baserat deras komponenter ger en övergripande bild av hela datamängden och vilka modeller som kan anses mer lika varandra.

En funktion baserad på Jaccards index togs fram för att ge ett numeriskt värde för likheten mellan två stycken maskiner baserat på deras komponenter. Genom att jämföra utvalda maskiner från alla olika modeller kunde kluster av maskiner identifieras. Maskiner inom samma modell jämfördes för att identifiera variationen inom respektive modell. Detta gör det möjligt att identifiera maskinexemplar som sticker ut. Systematiska sökningar gjordes i filerna för att ta fram specifika komponenter så som hytter eller identifiera antal komponenter som innehåller bly. Slutligen konstruerades en enkel applikation som låter användaren navigera BOM strukturen för två valbara maskiner och se vilka komponenter som specifikt skiljer de två åt. Sammanfattningsvis möjliggör arbetets resultat en arbetsgång där övergripande kluster kan identifieras i hela datamängden, avvikande maskiner inom modeller kan identifieras och slutligen undersökas i detalj med applikationen.

Litteraturförteckning

[1] A. Jenkins, "What Is a Bill of Materials (BOM)? Expert Guide & Tips." https://www.netsuite.com/portal/resource/articles/erp/bill-of-materials-bom.shtml. Hämtad: 2023-06-02.

[2] PTC, "What is plm?." https://www.ptc.com/en/technologies/plm-ppc. Hämtad: 2023-05-30.

[3] CIMdata, "Product lifecycle management (plm) definition." https://www.cimdata.com/en/resources/about-plm. Hämtad: 2023-05-30.

[4] JSON, "Introducing json." https://www.json.org/json-en.html. Hämtad: 2023-05-10.

[5] Oracle, "Introduction to json." https://javaee.github.io/tutorial/jsonp001.html. Hämtad: 2023-05-10.

[6] P. Beginners, "Introduction to csv files." https://python-adv-web-apps.readthedocs.io/en/latest/csv.html. Hämtad: 2023-05-31.

[7] Y. Shafranovich, "Common format and mime type for comma-separated values (csv) files." https://datatracker.ietf.org/doc/html/rfc4180#page-2. Hämtad: 2023-05-31.

[8] Python, "Csv file reading and writing." https://docs.python.org/3/library/csv.html#module-csv. Hämtad: 2023-05-31.

[9] S. S. Epp, *Discrete mathematics with applications*. Boston : PWS Pub. Co., 1995.

[10] "Python." https://www.python.org/. Hämtad: 2023-06-05.

[11] TIOBE, "Tiobe index for june 2023." https://www.tiobe.com/tiobe-index/. Hämtad: 2023-06-05.

[12] C. Voskoglou, "What is the best programming language for Machine Learning?." https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7. Hämtad: 2023-06-05.

[13] python.org, "tkinter - python interface to tcl/tk." https://docs.python.org/3/library/tkinter.html. Hämtad 2023-05-09.

[14] scikit learn, "scikit-learn - machine learning in python." https://scikit-learn.org/stable/. Hämtad: 2023-06-05.

[15] scikit learn, "sklearn.model_selection.train_test_split." https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html. Hämtad: 2023-05-15.

[16] scikit learn, "sklearn.feature_extraction.text.countvectorizer." https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html. Hämtad: 2023-06-05.

[17] scikit learn, "sklearn.decomposition.truncatedsvd." https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html. Hämtad: 2023-06-05.
[18] G. Golub and C. V. Loan, *Matrix computations 4th Edition*. The Johns Hopkins University Press, 2013.
[19] scikit learn, "sklearn.manifold.tsne." https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html. Hämtad: 2023-06-05.
[20] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research 9*, pp. 2579–2605, 2008.
[21] Azure, "Nosql-databas – vad är nosql?." https://azure.microsoft.com/sv-se/resources/cloud-computing-dictionary/what-is-nosql-database. Hämtad: 2023-05-30.
[22] Azure, "Vad är azure?." https://azure.microsoft.com/sv-se/resources/cloud-computing-dictionary/what-is-azure/. Hämtad: 2023-05-30.
[23] Azure, "Welcome to azure cosmos db." https://learn.microsoft.com/en-us/azure/cosmos-db/introduction. Hämtad: 2023-05-30.
[24] Azure, "Introduction to azure storage." https://learn.microsoft.com/en-us/azure/storage/common/storage-introduction. Hämtad: 2023-05-30.
[25] Azure, "What is azure data factory." https://learn.microsoft.com/en-us/azure/data-factory/introduction. Hämtad: 2023-05-30.
[26] Azure, "Copy activity performance and scalability guide." https://learn.microsoft.com/en-us/azure/data-factory/copy-activity-performance. Hämtad: 2023-05-09.
[27] Azure, "Request units in azure cosmos db." https://learn.microsoft.com/en-us/azure/cosmos-db/request-units. Hämtad: 2023-05-09.
[28] F. Karabiber, "Jaccard similarity." https://www.learndatasci.com/glossary/jaccard-similarity/. Hämtad: 2023-05-31.
[29] V. R. Joseph, "Optimal ratio for data splitting," *STATISTICAL ANALYSIS AND DATA MINING*, pp. 531–538, 2022.
[30] J. Nummenmaa and A. Ranta, *Databases in 137 Pages*. 2021. [Opublicerad manuskript].
[31] Microsoft, "Parameterized queries in azure cosmos db." https://learn.microsoft.com/en-us/azure/cosmos-db/nosql/query/parameterized-queries#examples. Hämtad: 2023-05-22.
[32] Naturvårdsverket, "Fakta om bly." https://www.naturvardsverket.se/amnesomraden/miljofororeningar/metaller/fakta-om-bly/. Hämtad: 2023-05-25.

A

Funktion i Python för Jaccards likhet mellan två listor

```python
def compare_lists(list1: list[str], list2: list[str]):
    """
    Compare machines by their parts. Given two lists of all parts for each machine.
    :return: a ratio of parts in common (value between 0 and 1)
    """
    if not list1 or not list2:
        raise Exception("List cannot be empty")

    counter1 = Counter(list1)
    counter2 = Counter(list2)

    #number of common parts (overlap)
    common_sum = sum(min(counter1[element], counter2[element]) for element in counter1)

    # Ratio of common parts
    return (2 * common_sum) / (len(list1) + len(list2))
```

Kodexempel A.1: Kod i Python för en funktion som tar två listor som argument och ger tillbaka ett värde för deras gemensamma innehåll.

B

Funktioner i Python för sökning i Azure databas

```
    models = set()

    for item in container.query_items(
            query='SELECT * FROM c',
            enable_cross_partition_query=True):
        models.add(item['parts'][0]['hostingPartName'])
```

Kodexempel B.1: Utdrag av kod i Python för att loopa igenom Azure databasen (container) och spara alla modellnamn i ett set

```
    name = "Some Model Name"
    pins = []

    for item in container.query_items(
            query='SELECT c.pin FROM c WHERE c.parts[0].
   hostingPartName=@hostName',
            parameters=[
                {"name": "@hostName", "value": name}
            ],
            enable_cross_partition_query=True):
        pins.append(item)
```

Kodexempel B.2: Utdrag av kod i Python för att söka i Azure databasen (container) och spara alla maskiners pin som tillhör en viss modell

C

Utdrag av funktioner i Python till applikationen

```python
def get_from_pin(pin):
    global container
    for item in container.query_items(
            query='SELECT * FROM c WHERE c.pin=@hostPin',
            parameters=[
                {"name": "@hostPin", "value": pin}
            ],
            enable_cross_partition_query=True):
        return item
```

Kodexempel C.1: Kod i Python för en funktion som tar en pin (sträng) som argument och returnerar den första datan som matchar i databasen (container).

```python
names1, numbers1 = create_first_level_list(data1)
names2, numbers2 = create_first_level_list(data2)

main_diff1 = []
for ind, part in enumerate(numbers1):
    if part not in numbers2:
        main_diff1.append(ind)
```

Kodexempel C.2: Utdrag av kod i Python för att skapa listor av maskinernas första nivå och sedan identifiera de index med innehåll som inte finns i den andra listan.

```python
def traverse_search(data, search_keys):
    indices = []
    if 'children' in data:
        for ind, part in enumerate(data['children']):
            if traverse_search_children(part, search_keys): # if part has at least one child in search_keys
                indices.append(ind)
    elif 'parts' in data:
        for ind, part in enumerate(data['parts']):
            if traverse_search_children(part, search_keys):
                indices.append(ind)
    else:
        raise Exception("Invalid Data. No 'parts' or 'children'")
```

```
13      return indices
```

Kodexempel C.3: En funktion i Python för att identifiera vilka delar i datans första nivå som har underkomponenter som finns i search-keys. Returnerar en lista av index för de identifierade delarna.

```
1  def traverse_search_children(part, search_keys):
2      for child in part['children']:
3          if child['partNumber'] in search_keys:
4              return True
5          elif 'children' in child:
6              if traverse_search_children(child, search_keys):
7                  return True
8      return False
```

Kodexempel C.4: En funktion i Python som returnerar True ifall maskindelen (part) innehåller någon underkomponent som finns i search-keys. Annars returneras False.

```
1  def raise_frame(frame):
2      frame.tkraise()
3
4  startButton = Button(listPage, text="back to start", command=lambda
       : raise_frame(startPage))
```

Kodexempel C.5: En Tkinter knapp med en 'callback function' till argumentet command. Att klicka på knappen anropar funktionen som växlar sida.

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige
www.chalmers.se

GÖTEBORGS
UNIVERSITET

CHALMERS