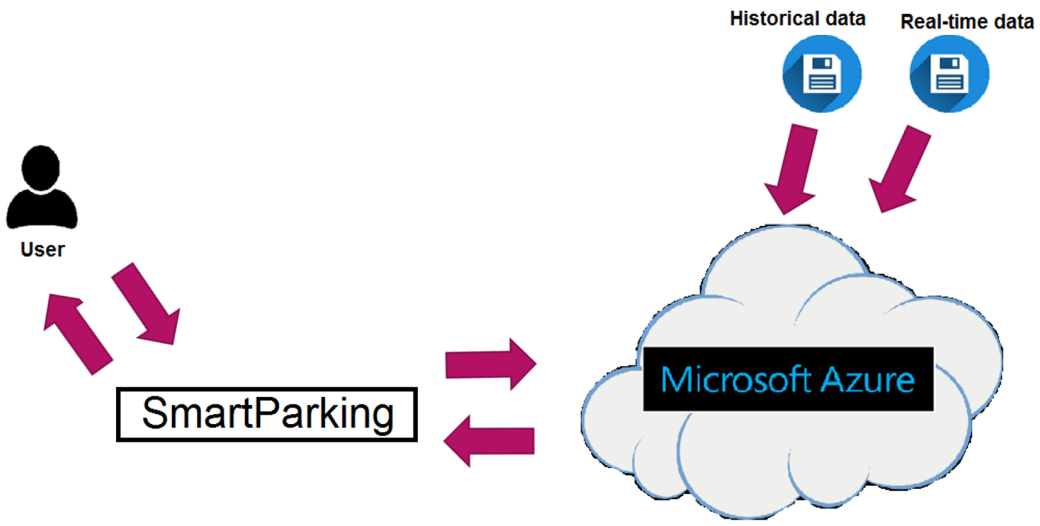
CHALMERS



SmartParking

Ett verktyg för att underlätta för pendlare

Examensarbete i Data- och Informationsteknik

Viktor Albihn
Joakim Willard

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2018

EXAMENSARBETE

SmartParking

Ett verktyg för att underlätta för pendlare

Viktor Albihn
Joakim Willard

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2018

SmartParking
Ett verktyg för att underlätta för pendlare
Viktor Albihn
Joakim Willard

Chalmers handledare: Sakib Sistek, Institutionen för data- och informationsteknik
Företags handledare: Gabriel Ibanez, Cybercom Göteborg
Examinator: Peter Lundin, Institutionen för data- och informationsteknik

Institutionen för Data- och Informationsteknik
Chalmers tekniska högskola / Göteborgs universitet
SE-412 96 Göteborg
Sverige
Telefon +46 31 772 1000

Omslag: Visualisering av de kompletta systemet, användare skickar förfrågan till SmartParking och får svar tillbaka.

Institutionen för Data- och Informationsteknik
Göteborg, Sverige 2018

SmartParking
Ett verktyg för att underlätta för pendlare
Viktor Albihn
Joakim Willard
Institutionen för Data- och Informationsteknik, Chalmers tekniska högskola

Sammanfattning

Göteborgs kommun strävar efter att minska trafiken i centrum. Västtrafik, som ansvarar för kollektivtrafiken i västra Sverige, letar efter lösningar för att minska trafiken. En lösning de tror på är att förse långdistanspendlaren med information i förtid, om det finns plats på pendelparkeringar som är i pendlarens område. För att se om detta var möjligt utfördes ett examensarbete, på Cybercom Group i Göteborg. Där man hade i uppgift att bedöma om det går att skapa ett proof of concept, och hur det i så fall uppnåddes. Resultatet från detta examensarbete blev ett enkelt system som består av en SQL-databas, en Machine Learning modell baserad på Decision Forest Regression algoritm, API:er samt ett enkelt användargränssnitt skrivit i Python. Designen på systemet är utformat på ett sådant sätt att det enkelt ska kunna användas av tredjepartsutvecklare. Man har lyckats att gå från klient via webb-API till Machine Learning modellen och tillbaka. Resultat kan ge Västtrafik en inblick i hur de kan vidareutveckla detta, till något som kan tas i bruk. Det kan annars visa hur man kan gå tillväga för att bygga upp ett liknande system. Om Västtrafik anser att detta är något att fortsätta med, är det möjliga resultatet en minskning av trafiken i centrala Göteborg. Projektet involverade inte att man skulle leverar en färdig produkt eller att den skulle vara fullt optimerad för alla pendelparkeringar i Göteborg.

Nyckelord: Azure, Machine Learning, SQL, Decision Forest Regression, Python.

Abstract

The municipal of Gothenburg aims to reduce the traffic in the city. Västtrafik, which is responsible for public transport in western Sweden, is looking for solutions to help the municipal of Gothenburg to achieve this. One solution that Västtrafik believes in, is to provide information to the long-distance commuters in advance, if there is free space in commuter parkinglot. To see if this was possible, a bachelor thesis was performed at Cybercom Group in Gothenburg. The purpose of this thesis was to assess whether a proof of concept could be created and if so how it was achieved. The result of this graduation work became a basic system consisting of an SQL database, a Machine Learning model based on Decision Forest Regression algorithm, APIs and a basic user interface written in Python. The system is designed in such a way that it can easily be used by third-party developers. A full system from client via web API to the Machine Learning model and back, has been developed. This result can give Västtrafik an insight into how to further develop this into a functioning system. It can otherwise show how to build a similar system. If Västtrafik considers this valueable, the possible outcome is a reduction of traffic in Gothenburg. The project did not intend to deliver a fully finished product or that it would be fully optimized for all commuter parkinglots in Gothenburg.

Keywords: Azure, Machine Learning, SQL, Decision Forest Regression, Python.

Förord

SmartParking är ett examensarbete som utförts vid institutionen för Data- och informationsteknik på Chalmers tekniska högskola i Göteborg av två studenter på högskoleingenjörsprogrammet i datateknik. Omfattningen av examensarbetet motsvarar en hel termins halvtidsstudier. Examensarbetet skedde i samarbete med Cybercom Group i Göteborg.

Vi vill framföra ett stort tack till vår handledare Magnus Kjellberg, från Cybercoms konsultteam i Göteborg. Vi vill även tacka Gabriel Ibanez, ledare för Cybercoms Innovation Zone i Göteborg. Sist men inte minst, ett stort och varmt tack till Sakib Sistek, vår handledare på Chalmers.

Viktor Albihn, Gothenburg, June 2018
Joakim Willard, Gothenburg, June 2018

Innehåll

Beteckningar

ML - Machine Learning
API - Application Programming Interface
MVP - Minimum Viable Product
GUI - Graphical User Interface
SQL - Structured Query Language
SSMS - SQL Server Manager Studio
IDE - Integrated Development Environment

1
Inledning

I detta kapitel beskriver man bakgrunden, syftet och målet för projektet samt avgränsningarna.

1.1 Bakgrund

Miljöfrågan är en av de största politiska frågorna i dagens samhälle, vad och hur man kan göra förändringar för att göra en positiv påverkan på miljön. Göteborgs kommun har bland annat lagt mycket fokus på att försöka minska mängden biltrafik i innerstaden. Västtrafik[1] som är företaget som ansvarar för kollektivtrafiken i västra Sverige letar efter olika sätt att hjälpa till att uppnå detta mål. En idé som Västtrafik har kommit på är att försöka göra långdistans pendlandet mer användarvänligt. Då Västtrafik i dagens läge bara har ett enkelt system som hämtar föregående dags antal lediga platser för den sökta tiden. Man vill istället ha ett mer pålitligt system som använder sig av ML, som skulle höja sannolikheten att svaret på antal lediga parkeringsplatser är korrekt. Uppgiften att skapa denna proof of concept gavs till IT-konsult företaget Cybercom[2]. För att uppnå detta tänkte Cybercom använda insamlad data från de "smarta" pendelparkeringarna. Med denna data och implementering av Machine Learning ska en förutsägelse om antalet lediga platser kunna ges. Detta kommer att utföras i Microsofts Azure Cloud plattform.

1.2 Syfte

Syftet med projektet är att ge Västtrafik ett proof of concept, som kan användas som grund, för hur ett sådant system, enligt ovan beskrivning kan utformas. Detta ska göras för att Västtrafik ska kunna bedöma om det är rimligt, att utveckla och integrera ett liknande system i deras nuvarande applikation gällande information om kollektivtrafik. Det nya systemet skulle kunna förbättra reseplaneraren för deras kunder.

Frågeställning 1: Går det att skapa ett sådant proof of concept?
Frågeställning 2: Hur har man gått tillväga för att utveckla detta?

1.3 Mål

Målet med projektet är att skapa ett proof of concept som ska kunna förutspå om det kommer finnas ledig parkering på pendelparkeringar. Detta proof of concept som ska skapas kommer bestå av ML-modeller, för vardera parkeringsplats, som ska försökas generaliseras till en enda modell. Det kommer också skapas en databas som ska innehålla den historiska datan som modellen ska tränas med. För att kunna utnyttja ML-modellen kommer det produceras ett program som hanterar kommunikationen mellan modellen och användar klienten. Svaret från ML-modellen kommer sedan presenteras i ett grafiskt användargränssnitt.

1.4 Avgränsningar

Vad som inte kommer ske i detta projekt:

- Projektet utvärderar inte Azures förmåga inom ML
- Projektet utvärderar inte olika ML-program
- Projektet utvärderar inte vilken algoritm som ger bästa resultat
- Projektet hanterar inte att ML-modellen ska fullt optimeras

Detta är vad som kommer att ske i detta projekt:

- Optimering till en coefficient of determination av 0,7-0,8
- Utveckling och skapande av ett API
- Utveckling och skapande av en ML-modell
- Utveckling och skapande av ett användargränssnitt
- Uppstrukturering av en databas

2
Teknisk bakgrund

Detta kapitel kommer att behandla de tekniska termer och verktygen som har använts under detta projekt.

2.1 Azure

Azure är ett molnbaserat arbetsplattform med en rad inbyggda funktioner. Dessa funktioner gör det möjligt att utveckla och underhålla applikationer.[3] Exempel på några funktioner är SQL-databas, hosting av webbapplikationer och Machine Learning Studio.[4]

2.2 Python

Python är ett högnivå objektorienterat programmeringsspråk. Dess enkla syntax tillsammans med dynamisk typning gör det väldigt lättlärt. Då det också innehåller stöd för scripting gör det språket mångsidigt. Python stödjer skapandet av moduler och paket, vilket främjar kodmodularitet. Dessutom innehåller Python ett brett utbud av standardbibliotek som ger stöd för många olika funktioner.[5]

2.3 Machine Learning

Machine Learning används till att förutspå händelser eller förbättra prestanda av en process, detta görs genom att använda historisk data. Machine learning detekterar mönster i den historiska datan, genom olika algoritmer. När ett mönster har hittats används detta för att kunna förutse en framtida händelse eller förbättra arbetsförmågan hos ett system.

Machine learning ger datorsystem möjligheten att lära sig och att förbättra sig över tid, genom att tränas med kontinuerligt uppdaterad data. Detta gör att förutsägelserna blir bättre för specifika uppgifter. För att lära algoritmen ett mönster finns det tre olika sätt att sköta detta på, supervised learning, unsupervised learning och enhanced learning.[6][7][8]

Coefficient of Determination(R-squared)

Coefficient of Determination även kallad R-squared är ett statistiskt verktyg att utvärdera hur bra ML-modellen är jämfört med basmodellen. Basmodellen är medelvärdet för all historisk data som ML-modellen har tränats med, det vill säga basmodellen kommer alltid att ge medelvärdet som svar på en förutsägelse.

Värdet på R-squared ligger mellan 0-1, där 0 är att ML-modellen är exakt samma som basmodellen och 1 är då ML-modellen har hitta en algoritm som kan förutse alla värden från datan exakt, det vill säga ML-modellen är perfekt utformad för uppgiften.[9]

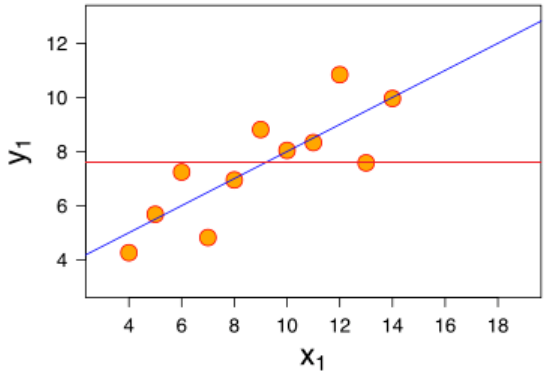


Figure 2.1: Röd linje är basmodellen, blå linje är ML-modellen

Feature

Feature är, ofta, en mätbar variabel som har en påverkan på förutsägelsen. Val av feature görs för att underlätta för datorsystem att hitta rätt mönster i den historiska datan.[10]

Supervised learning

Supervised learning innebär att systemet tränas med både den historiska datan, som den ska göra en förutsägelse på, samt svaret på förutsägelsen. Detta görs för att leda systemet till att hitta mönster mellan real-tidsdata och förutsägelsen. Den historiska datan behöver vara sorterad om datan är av typen tids-serie data, då sekvensen av händelser har relevans för mönstren som algoritmen ska hitta.[8]

Underfitting

Underfitting är när en modell gör felaktiga förutsägelser. Detta kan då bero på att den inte har lyckats hitta ett mönster eller hittat ett dåligt mönster i datan. Exempel på andledningar till detta kan vara dålig data eller för lite data.

Overfitting

Overfitting är när modellen lär sig datan för bra och kan därför inte hitta generella mönster. Detta gör så att modellen inte kan hitta mönstren på ny data, vilket medför att förutsägelserna blir dåliga.

Tids-serie data

Tids-serie data innebär att datan har en tidsaspekt, detta medför att individuella händelser i datan sker i en sekvens. Datamängder som representeras under en viss tidsperiod så som en månad, ett kvartal eller ett år. Tids-serie data består av mätningar som görs kontinuerligt.[11]

2.4 Databas

En samling data eller information som är sorterad på ett sådant sätt att man lätt kan söka efter och hämta en särskild data mängd som matchar sökningen, är en databas. Från början sparade man datan på ett hierarkiskt sätt så att man kunde söka och hitta informationen snabbt. Dock blev det väldigt komplext att manipulera den informationen som redan är sparad i trädstrukturen. Lösningen på detta problemet blev att man istället började använda relationer inom de olika tabellerna, som man skapade. Datan sparades i rader i dessa tabeller så att man inte tog bort den smidiga sök och hämta funktionen av enstaka data.[12]

2.5 Webb-API

Webb-API är ett API där HTMLs GET, POST, UPDATE och REMOVE används för att kommunicera mellan en klient och en server. Detta används för att skapa webbapplikationer, då en URL kan länkas till en specifik funktion i webb-API:et.[13]

2.6 JavaScript Object Notation

JavaScript Object Notation, JSON, är ett fil-format för att skicka data mellan webb-tjänster. JSON följer kodstandarder vilket gör det enkelt att läsa och använda. Ett JSON-meddelande byggs upp av ett JSON-objekt som innehåller JSON-objekt eller JSON-array. Det som finns i JSON-array eller JSON-objektet är olika värden av olika variabeltyper. [14]

2.7 Versionshantering

Versionhantering är en arbetsmetodik för att hålla reda på förändringar i ett projekt. Detta görs genom att man har en huvudstam som innehåller alla de senast fungerande funktionaliteter. När en ny funktionalitet ska läggas till, så från stammen skapar man en ny gren där den nya funktionaliteten skapas. När den anses vara

klar så återinför man den till stammen. Denna stam kan antingen finnas lokalt på ens egna dator eller på en internet tjänst. Detta gör det möjligt för flera utvecklare att utveckla olika funktionaliteter samtidigt, vid händelse att en ny funktionalitet skulle göra att systemet inte fungerar kan man återvända till senaste fungerande version av projektet.[15]

2.8 Agil utveckling

Agil utveckling är ett arbetssätt för att påskynda och förbättra programvaruutveckling. Detta görs genom att strukturera gruppens arbetssätt, med dagliga möten inom arbetsgruppen samt fortlöpande möten med produktägaren. Detta görs för att få kontinuerliga uppdateringar och feedback på projektet.[16]

3
Metod

Under hela projektet arbetade man agilt, då man satt på Cybercom Group Göteborg och var en del av arbetsgruppen Innovation Zone som hade dagliga möten. Där man går igenom vad som har gjorts, vad som ska göras och om man har några hinder i projektet. Utöver detta hölls regelbundna möten med Västtrafik som var produktägaren, för att hålla dem uppdaterade under projektets gång.

I det tidiga stadiet av projektet lades fokus på att lära sig Python, som användes för normalisering av rådatan från Västtrafik. Efter detta var gjort övergick projektet till att fokusera på ML-aspekten, där arbetsprocessen vara delvis experimenterande i ML Studio samt informationshämtning. För inhämtning av information så användes olika online källor samt ställde frågor till ML-mentor på Cybercom. Under informationshämtandet hittades ett liknande projekt, dock hade det primärt fokus på tränande av ML-modellen och ansågs inte som relevant för detta projekt.

Då ML-modellen uppnått önskat resultat, skiftade fokus på nytt till att kunna kommunicera med modellen. Utvecklingsprocessen utgick ifrån att först identifiera vilka olika klasser som behövdes, och sedan implementera dessa.

Projektet följde tidsplanering som finns i appendix.

3.1 Stash

I Stash, vilket är ett versionshanterings program, har vi en huvudgren *master branch* där det endast finns det som visas vid demon, en utvecklingsgren *develop branch* som innehåller allt som fungerar av det som är under utveckling. Innan demon gör man en sammanfogning av huvudgrenen och utvecklingsgrenen. Nya funktioner utvecklas i funktionsgrenar *feature branches*.

3.2 Minsta möjliga produkt

MVP för detta projekt är att skapa ett system som får en förfrågan från en användare, och sedan med hjälp av en data samling och Machine Learning gör en förutsägelse som presenteras i ett enkelt användar gränssnitt.

- Databas - Hantering av historisk data, konverteringstabeller av ord till nummer
- Olika modeller för olika parkeringar
- Användaren kommunicerar med modell i real-tid via webb-API
- Sättas upp som en webbapplikation
- Förutse 2 timmat i förväg
- API som kommunicerar med ML modell
- Presentera resultat från förutsägelsen i ett GUI
- Script som uppdaterar historisk data

3.3 Ytterligare funktionalitet

Ytterligare funktionalitet som har utvecklats men som inte behövdes för att uppnå produktägarens krav.

- Databasen lagrar även förutsägelser
- Förutse upp till 12h
- Effektivisera för användaren
- Script som får förutsägelser att ske varje 15 min
- Script som lagrar undan förutsägelser i databasen
- Naiv validering av förutsägelsen
- En generell modell

4
Genomförande

Detta kapitel kommer beskriva de olika delarna av systemet, samt arbetsgången av skapandet av dessa.

4.1 System Design

Detta avsnitt går igenom de olika komponenterna av systemet. Systemet kan delas in i en modell del och en kod del.

ML Modell

ML-modellen är tränade enligt supervised learning, då modellen får features som förutsägelsen ska baseras på samt svaret på förutsägelsen. Modellen får in data via en databas anslutningsmodul. Datan delas sedan upp på år, 2014 till 2016 denna data blir träningsdatan och år 2017 blir verifieringsdata för modellen. Träningsdatan används av en *regressions*-algoritm för att få en tränad modell. Sedan testas den tränade modellens förutsägelser mot verifieringsdatan. Efter det att modellen har blivit verifierad görs modellen om till en webbapplikation som kan anropas med dess tilldelade API-nyckel.

Program struktur

Detta underavsnitt går igenom klasserna som skapats, i samband med detta projekt, och utgör API:et, samt scriptet som använder API:et. UML av programmet visas i figur 5.1 i Resultat kapitlet.

AzureDBAccess

Klassen AzureDBAccess hanterar all kommunikation med SQL-databasen som finns uppsatt på Azure plattformen. Detta innebär upprättande och avslutande av anslutning till databasen. Hämtande av historisk data, utföra konverteringar mellan olika typer av parkerings ID:n.

Utility

Denna klass innehåller metoder för konvertering av UNIX-tidsstämplar och identifiera röda dagar.

VTAccess

Denna klass hanterar att hämta ny säkerhetstoken från Västtrafiks REST-API, samt hämtar historisk- och real-tidsdata om antalet lediga parkeringsplatser från deras API.

WeatherData

Denna klass hämtar väderdata från tredjeparts API:et DarkSky, samt filtrerar ut temperatur och en väder summering ur svaret.

ApplicationUpdateDB

Denna klass hämtar historisk data från databasen, via AzureDBaccess, och väder data från Darksky, via WeatherData, samt real-tidsdata från Västtrafik, via VTAccess. Detta byggs sedan ihop till en sträng som lagras i databasen.

ApplicationMakePrediction

Denna klass hämtar datan som behövs för att göra förutsägelsen från databasen. Bygger ihop JSON-meddelande med denna information och skickar det till ML-modellen. ML-modellen gör förutsägelsen och returnerar ett JSON-meddelande till klassen som sedan blir bearbetat och sedan förs in i databasen.

ApplicationGetPrediction

ApplicationGetPrediction är en webbapplikation som tar emot parkeringsområde, datum och tid från en klient och hämtar förutsägelsen från databasen och returnerar den till klienten.

SmartParkingGUI

Denna klass skapar användargränssnittet, där användaren matar in parkeringsområde, datum och tid. Detta skickas till ApplicationGetPrediction som sedan returnerar ett JSON-medelande. Ur detta medelande hämtas den relevanta informationen, som sedan presenteras för användaren.

Programflöde

Klassen ApplicationUpdateDB körs var 15:e minut via Windows schemaläggare. Den initierar AzureDBAccess, VtAccess och WeatherData, bygger insert-strängarna för de olika parkeringsplatserna med hjälp av den historiska datan som finns lagrad i databasen och som hämtas via AzureDBAccess. Dessutom behövs även real-tidsdata från Västtrafik i dessa insert-strängarna. Insert-strängarna skickas sedan till AzureDBAccess som utför inserts:en till databasen.

ApplicationMakePrediction körs var 15:e minut via Windows schemaläggare. Den initierar AzureDBAccess och WeatherData, bygger ihop strängen som ML-modellen

behöver för att kunna göra förutsägelsen. Detta genom att hämta historisk data via AzureDBAccess och hämtar väderdata genom WeatherData. Denna sträng skickas sedan till ML-modellen som gör förutsägelsen och returnerar den. Returvärdet som kommer tillbaka formateras till en insert-sträng som sedan blir införd i databasen.

Scriptet ApplicationGetPrediction tar emot parametrarna parkeringsplats och eftersökt tid från användaren. Tiden som användaren matar in omvandlas till UNIX-tidsstämpel, genom en statisk metod som finns i utility klassen. Den eftersökta förutsägelsen hämtas från databasen och konverteras till ett JSON-objekt som returneras till användaren.

Användarklienten hämtar sedan ut informationen från JSON-objektet och presenterar det för användaren.

4.2 Arbetsgång

Detta avsnitt går igenom arbetsgången under projektet.

Datanormalisering

Det första och mest tidskrävande steget var datanormaliseringen av den historiska datan från Västtrafik. Datan innehåller information, alla händelser som skett på smarta parkeringar, om alla Västtrafiks pendelparkeringar. En händelse är när en bil antingen kör in eller ut ur en parkeringsplats. Först analyserades datan och sorterar ut den data som anses vara relevant, för de målen som skulle uppnås. Efter att datan var sorterad för de parkingsplatser som skulle analyseras, sorterades korrupt data bort och om det saknades data fylldes dessa glapp i med föregående värde. Sortering och filtrering gjordes med enkla Python script.

Modell skapandet

Efter att datan blivit normaliserad, gjordes en tutorial i Azure ML Studios för att snabbt få en överblick över de olika funktioner som programmet har. Efter tutorialn var gjord, bestämdes det att skapandet av modellen skulle ske iterativt. Detta innebar att göra små och få förändringar från en iteration av modellen till den nästa. Datan som modellen tränas med innehåller både informationen som förutsägelsen ska utföras på samt värdet som modellen bör förutse. På grund av detta tränas modellen enligt supervised learning. Med detta skulle första iterationen byggas.

Iteration ett

Detta krävde att ett val gjordes angående vilken typ av algoritm som skulle användas, beroende på vilken typ av svar som efterfrågas. Valet låg mellan ett numeriskt värde, antalet lediga parkeringsplatser, eller ett binärt svar, om det finns lediga platser eller ej. Västtrafik ville vid denna tidpunkt ha ett numeriskt värde som svar, detta gjorde att typen regressions-algoritm valdes. Det behövde också

bestämmas vilka features som skulle kunna ha en inverkan på modellens resultat. På ett möte med ML-mentorn från Cybercom bestämdes det, att först börja med väder och om det är röd dag eller ej. När detta var bestämt prövades först linjär regressions-algoritm. Datan som modellen tränades med var delvis uppdelade efter parkeringsplats ID, men också efter en enkel tumregel. Denna tumregel är att använda 2/3 till 3/4 av datan som träningsdata och resterande delen som verifieringsdata. Denna iteration gav ett lågt resultat. Vid konsultation med ML-mentorn bestämdes det att linjär regressions-algoritm inte var bra nog, och att Decision Forest Regression valdes istället.

Iteration två

Med den nyvalda algoritm och i övrigt samma förutsättningar som tidigare, fick denna modellen väldigt högt resultat. När resultat visades för ML-mentorn, påpekades det att resultatet verkade orimligt bra. Det orimligt bra resultatet indikerade overfitting. Dock påpekade mentorn att resultatet berodde på att datan som användes är av typen tids-serie data. Innebörden av att det är tids-serie data, var att modellen fick svaret på förutsägelsen samtidigt som den skulle utföra förutsägelsen. Detta gjorde att den enkelt kunde förutsäga resultatet. Vilket medförde att uppdelningen av data inte kan göras slumpmässigt, utan måste istället fördelas på tidsbasis. Datan fördelades så år 2014-2016 blev träningsdata och 2017 blev verifieringsdatan.

Iteration tre

Nästa iteration av modellen som nu hade datan uppdelad på tid, men samma algoritm som tidigare iteration fick lågt resultat. En diskussion fördes det om mängden data som modellen tränades med var tillräckligt stor. På grund av diskussionen ändrades uppdelningen av datan, till att inte längre fördelas på individuell parkeringsplats. Utan endast att dela den på tid. Denna förändring påverkade dock inte resultatet något nämnvärt.

Iteration fyra

Då tidigare resultat hade berott på tids-serie data, valde man att läsa på ytterligare om tids-serier. Det var vid detta tillfälle som man insåg att modellen behövde ytterligare features, för att kunna hitta mönster. Med denna insikt valdes det att läggas till en feature, som var att ta föregående dygns värde för samma tid. Denna iteration av modellen gav avsevärt högre resultat. Med detta resultat valdes det inför nästa iteration att lägga till fler features, som utgjordes av föregående tid.

Iteration fem och sex

I iteration fem lades en vecka och två veckor tillbaka till, då blev resultatet blev ännu bättre än tidigare iteration. I samband med iteration sex diskuterades om 12 timmar tidigare skulle läggas till för att motverka övergången från söndag till måndag samt fredag till lördag. Då man visste att mönstret ändrades markant mellan vardagar och helger. Efter alla iterationer uppnåddes det eftersökta målet av

en coefficient of determination, som låg inom spannet 0,7-0,8. När detta var uppnått övergick projektet till att utveckla programkoden som skulle kunna kommunicera med modellen.

Databas

Efter att datan normaliserats så användes först CSV-filer som datakälla för modellen. Detta var inte ett effektivt sätt att jobba på, då vid varje förändring av filerna så behövdes de laddas upp till ML studio vilket gjorde att det blev flera versioner av filen i programmet. På grund av detta gjorde man valet att sätta upp en databas, för att lättare kunna manipulera datan och uppdatera den i real-tid. Man valde först att sätta upp en lokal databas för att inte öka kostnaderna för mycket för Cybercom. Eftersom Azure tar betalt för mängden data som ska lagras och man visste inte vid denna tidpunkt hur stor databas som behövdes. Detta gjordes för att ML studio har stöd för att lätt ansluta till en Azure databas. Efter att Azure databasen var uppsatt kunde den lokal databasen exportera över till Azure databasen. För att jobba smidigare mot Azure databasen så användes SSMS för att skapa och hantera de olika tabeller, views och queries. SSMS hade en inbyggd funktion som kunde läsa av CSV-filer och fylla en tabell med datan från filen.

Skapande av API och webb-API

Först bestämdes vilka olika kommunikationspunkter systemet skulle behöva ha, de som bestämdes var Azure databasen, Västtrafiks API och väder API:et Darksky. Efter detta var bestämt påbörjades skapandet av klassen för att kunna kommunicera med databasen. Denna klass består av en del som hämtar data för att kunna bygga förfrågan till modellen samt en del som för in data i databasen.

Då man avsåg att hålla databasen uppdaterad blev detta den mest prioriterade funktionaliteten. När denna klass var klar skiftade fokus till VtAccess som skulle kommunicera med Västtrafiks API. För att hämta ny säkerhetstoken och antalet lediga platser på en parkeringsplats. Vid detta skeddet var det bara klassen som skulle hämta data, både historisk- och väderprognosdata, från Darkskys API kvar.

Då en av features i ML-modellen var vilken typ av dag det är, behövdes också funktionalliteten att identifiera veckodagar, röda dagar samt perioderna då det är skollov. För att lösa detta skapades en Utility klass, som innehåller en metod som testar de satta röda dagarna. Från de satta röda dagarna finns det relationer till de röda dagar som kan infall under längre tidsspann, till exempel påsk som inte infaller på något satt datum eller vecka.

Efter att alla klasser hade skapats som hanterar att hämta data, så diskuterades det vilka klasser/script som skulle använda sig av dessa. Det valdes då att bygga klassen ApplicationUpdateDB som ansvara för uppdatera databasen med den information som ML-modellen behöver. Denna klass hämtar informationen från de olika källor och skickar detta för att föras in i databasen. Efter att funktionen för att

databasens innehåll hålls uppdaterad så var det dags att bygga script ApplicationMakePrediction som skulle begära förutsägelser från ML-modellen. Då detta script ska kunna anropas från en mobilapplikation eller webbläsare så behövde scriptet vara ett webb-API. Detta gjorde att Pythons standard bibliotek Flask valdes för att hantera webbapplikations funktionaliteter.

När detta script var gjort kunde man köra det lokalt på en dator där man kan skicka in en förfrågan och få ett svar från modellen. Vid ett senare tillfälle föreslogs det från mentorn på Cybercom att det borde skapas en klass som körs var femtonde minut. Funktionen begär en förutsägelse på tillgängliga parkeringsplatser två timmar framåt, som sedan kan lagras i en databas. Förslaget gavs för att göra systemet snabbare för användaren, då tidigare version utförde förutsägelsen i real-tid vilket tar tid. Det nya sättet gör att systemet bara hämtar förutsägelsen från databasen vilket är betydligt snabbare. Detta ledde till att ApplicationMakePrediction byggdes om från att vara webbapplikation till en vanlig klass. Samt att ApplicationGetPrediction, som blev den nya webbapplikationen, skapades vars funktionalitet är att hämta förutsägelser från databasen.

Användargränssnitt

För att presentera svaret som returneras från scriptet på ett bra och smidigt sätt så skapades ett enkelt användargränssnitt i Pythons standard GUI bibliotek, Tkinter. Det som behövs för att hämta en förutsägelse från databasen är parkeringsplatsen, datum och tid som användaren efterfrågar. Det bestämdes att användaren bara ska behöva mata in parkeringsområdet, inte den specifika parkeringsplatsen, samt datum och tid. När användaren begär information för ett parkeringsområde så returneras ett svar. Detta svar innehåller de individuella förutsägelserna för parkeringsplatserna inom detta parkeringsområdet. De parkeringsområden som har undersökts innehåller allt från en parkeringsplats till fyra stycken.

I användargränssnittet presenteras parkeringsplatserna under varandra, först namnet och under namnet skrivs datum och tid ut. Sedan kommer en boolean om det finns lediga platser eller ej. Boolean används för att i detta skedet bestämde Västtrafik att de ville ha en boolean som svar, då de trodde att detta underlättar för deras kunder att förstå svaret. Detta följs av en indikation av hur pålitlig förutsägelsen är, genom färgkodning där grön är hög sannolikhet att svaret stämmer, gult är att det är viss sannolikhet att svaret stämmer och rött är att det låg sannolikhet att svaret stämmer. Denna utvärdering av booleanen görs genom en enkel verifiering. Där förutsägelsen jämnförs med medelvärdet av värdena för en och två veckor tillbaka för samma tid.

4.3 Testning

Under projektets gång, när ny funktionalitet skulle läggas till kördes programmet för att testa att den nya koden fungerade korrekt och ändrades om den inte gjorde

det. Detta gjordes kontinuerligt under funktionalitetens utveckling samt innan den nya koden återinfördes i stam koden i Stash.

5
Resultat

Ett system som kan förutse antalet lediga platser på de utvalda pendelparkeringarna har skapats, presenteras som boolean på begäran av produktägaren Västtrafik. Förutsägelserna lagras i en databas, som sedan kan hämtas och visas i ett användargränssnitt. Det skapades en generell ML-modell som kan appliceras på alla utvalda parkeringsplatser.

5.1 SmartParking som resultat

Systemet som har skapats uppfyller specifikationerna för MVP som står i 3.2. Dessutom har ytterligare funktionaliteter som finns specificerade i 3.3 även utvecklats.

Övriga funktionaliteter utöver MVP

- Utföra förutsägelser i förtid istället för i real-tid och spara dom i databasen
- Ständig uppdatering av datan varje kvart i databasen
- En databas med lagrad historisk data
- Kunna utför förutsägelse upp till 12 timmar i förväg
- En enkel validering av förutsägelsen
- En generell modell skapades för parkeringsplatser som täcks av projektet parkeringsplatser

Beskrivning av systemet

Systemet består av två program som exekveras varje kvart. Det första av de två programmen ansvarar för att bygga ihop och lagra, i en cloud baserad databas, informationen som det andra programmet behöver för att begära en förutsägelse från modellen. Detta görs genom att programmet hämtar nuvarande antalet lediga platser på en parkeringsplats, sedan hämtas historisk data från databasen. Utöver denna information läggs också till vilken typ av dag det är, samt nuvarande väder situation. Allt detta förs sedan in i databasen för att kunna användas av nästa del av systemet.

Det andra programmet som utgör systemet, hanterar att fylla databasen med förutsägelser. För att göra detta så hämtar programmet informationen som föregående program har fört in i databasen. Med denna information skapas ett JSON-meddelande som skickas in till modellen. Modellen utför en förutsägelse baserat på de värden

som finns i JSON-meddelandet och returnerar ett JSON-meddelande innehållandes förutsägelsen. Förutsägelsen lagras sedan i databasen för att kunna hämtas av en klient.

Användaren matar in parkeringsområde, datum och tid i klienten. När denna informationen har blivit inmatad så trycker man på "Send query".

Värdena som matats in skickas som parametrar till webb-API:et. Baserat på värdena hämtas den data som utgör förutsägelsen från databasen. Datan formateras som ett JSON-meddelande och skickas tillbaka till klienten. Klienten presenterar den relevanta information från JSON-meddelandet för användaren.

Figur 5.1 nedan visar UML över systemet.

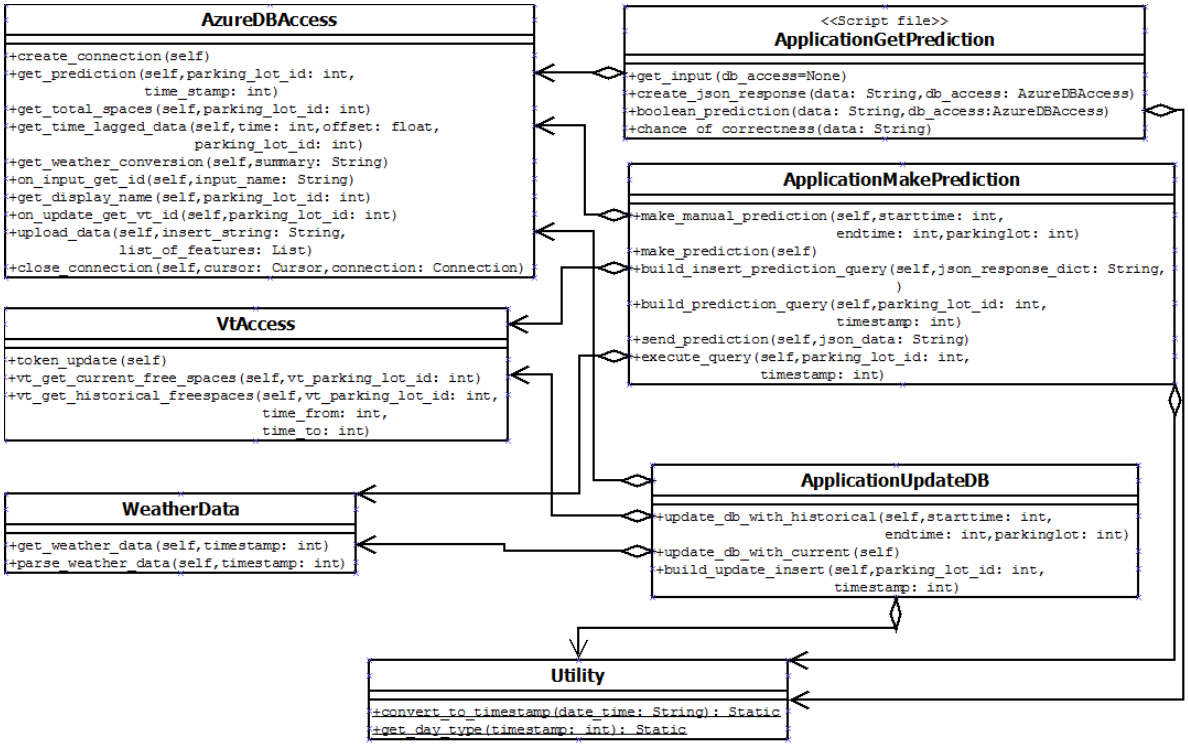


Figure 5.1: UML över systemet

5.2 Resultatet från SmartParking systemet

Här kommer det att beskrivas tillsammans med bilder hur resultatet ser ut från de olika aspekterna i systemet.

Resultat från användargränssnitt

Detta avsnitt kommer att visa hur användargränssnittet är uppstrukturerat samt hur svar från modellen presenteras.

Användargränssnittet är programmerat i Pythons standard GUI bibliotek, Tkinter. Användaren matar in parkeringsområde och tidpunkt, sedan trycker användaren på knappen "Send query". Då skickas efterfrågan till backenden. Användargränssnittet tar emot responsen från backenden och hanterar och presenterar datan för användaren.

Följand två bilder kommer visa hur resultat presenteras i användargränssnitt, figur 5.2 visar svar från ett parkeringsområde som består av endast en parkeringsplats.

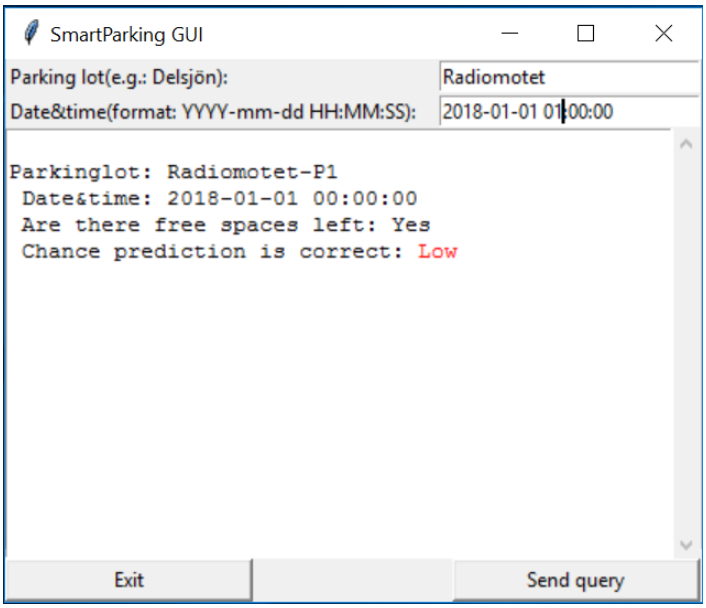


Figure 5.2: Resultat från användargränssnitt, parkeringsområde med en parkeringsplats

Figur 5.3 visar hur svaret ser ut från ett parkeringsområde bestående av flera parkeringsplatser.

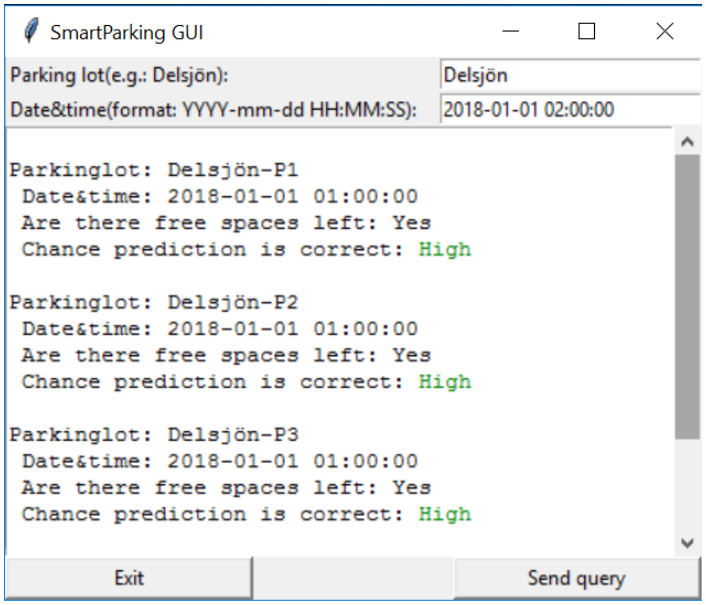


Figure 5.3: Resultat från användargränssnitt, parkeringsområde med flera parkeringsplatser

Resultat från ML-modell

Figur 5.4 och 5.5 illustrerar en jämförelse för två olika parkeringsplatser i Delsjöområdet, över en 14-dagars-period. Jämförelsen är mellan det verkliga värdet, som är den övre grafen, och förutsedda värdet, som är den nedre grafen.

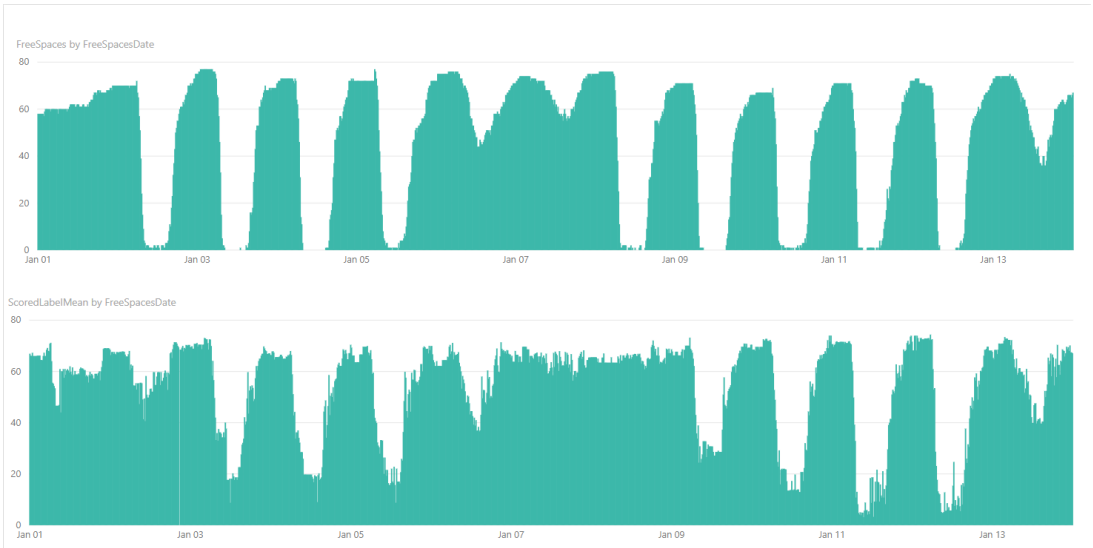


Figure 5.4: Jämförelse mellan faktisktvärde (övre graf) och förutsettvärde (undre graf) för parkering 17 under samma tidsperiod.

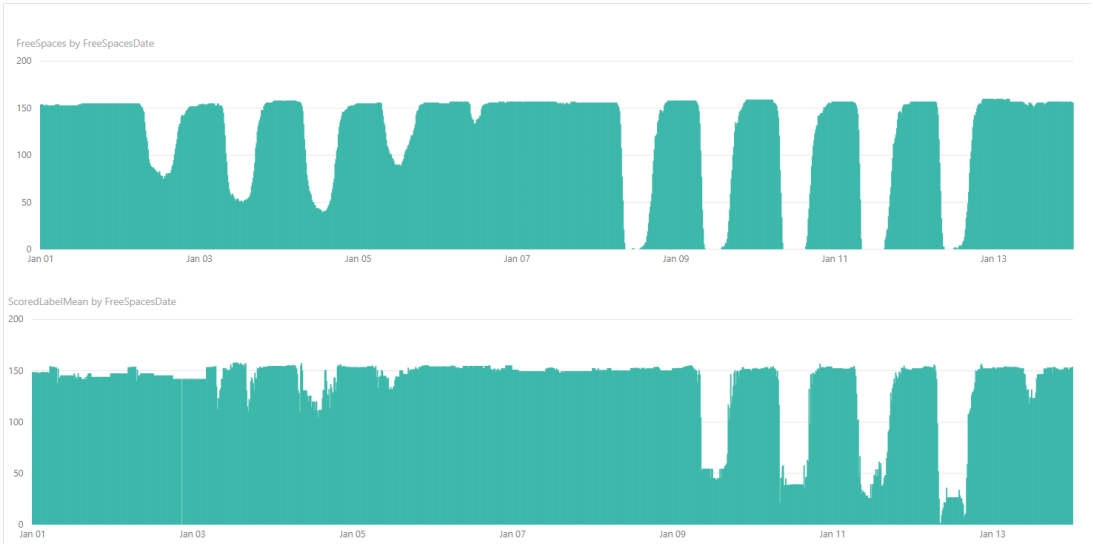


Figure 5.5: Jämförelse mellan faktisktvärde (övre graf) och förutsettvärde (undre graf) för parkering 172 under samma tidsperiod.

De andra förutsägelserresultaten finns i appendix A, sektion A.3.

Resultat lagrat i databasen

I denna del kommer man visa resultat både från hur den historiska datan lagras och hur förutsägelserna lagras.

Historisk data

Den historiska datan som är lagrad i SQL-databasen.

	ParkingLotID	FreeSpacesDate	FreeSpaces	Temperature	Summary	Holidays	PrevDay	PrevWeek	Prev12Hour	Prev2Weeks
1	17	2018-01-01 00:00:00.0000000	58	5	16	2	67	70	61	62
2	18	2018-01-01 00:00:00.0000000	51	5	16	2	50	50	50	48
3	112	2018-01-01 00:00:00.0000000	126	5	16	2	12	2	130	133
4	171	2018-01-01 00:00:00.0000000	72	5	16	2	72	72	72	71
5	172	2018-01-01 00:00:00.0000000	154	5	16	2	155	160	155	130
6	17	2018-01-01 00:15:00.0000000	58	5	16	2	69	70	60	62
7	18	2018-01-01 00:15:00.0000000	51	5	16	2	51	50	50	48
8	112	2018-01-01 00:15:00.0000000	126	5	16	2	11	2	129	133
9	171	2018-01-01 00:15:00.0000000	72	5	16	2	72	72	72	71
10	172	2018-01-01 00:15:00.0000000	154	5	16	2	155	160	155	130
11	17	2018-01-01 00:30:00.0000000	58	5	16	2	70	70	58	62
12	18	2018-01-01 00:30:00.0000000	51	5	16	2	51	50	50	48
13	112	2018-01-01 00:30:00.0000000	126	5	16	2	12	2	130	133
14	171	2018-01-01 00:30:00.0000000	72	5	16	2	72	72	72	71
15	172	2018-01-01 00:30:00.0000000	154	5	16	2	155	160	154	130
16	17	2018-01-01 00:45:00.0000000	58	5	16	2	70	70	55	62
17	18	2018-01-01 00:45:00.0000000	51	5	16	2	51	50	51	48
18	112	2018-01-01 00:45:00.0000000	125	5	16	2	12	2	130	133
19	171	2018-01-01 00:45:00.0000000	72	5	16	2	72	72	72	71
20	172	2018-01-01 00:45:00.0000000	154	5	16	2	155	160	155	130
21	17	2018-01-01 01:00:00.0000000	58	5	16	2	70	70	56	62
22	18	2018-01-01 01:00:00.0000000	51	5	16	2	51	50	51	48
23	112	2018-01-01 01:00:00.0000000	126	5	16	2	12	2	130	132
24	171	2018-01-01 01:00:00.0000000	72	5	16	2	72	72	72	71
25	172	2018-01-01 01:00:00.0000000	153	5	16	2	155	160	155	130

Figure 5.6: Lagrad historisk data.

Förutsägelser

Förutsägelserna lagras i samma stil som den historiska datan med undantaget att förutsägelserna har två extra kolumner. Dessa två kolumner är svaret som ML-modellen generar. Då detta är förutsägelser så är kolumnen över FreeSpaces, från figur 5.7, null. På grund av att FreeSpaces är real-tids värdet för parkeringen och existerar inte vid en förutsägelse.

	ParkingLotID	FreeSpacesDate	FreeSpaces	Temperature	Summary	Holidays	PrevDay	PrevWeek	Prev12Hour	Prev2Weeks	ScoredLabelMean	ScoredLableStandardDeviation
1	17	2018-01-01 00:00:00.0000000	NULL	5	16	2	67	70	61	62	66,8958333333333	8,13895703396285
2	18	2018-01-01 00:00:00.0000000	NULL	5	16	2	50	50	50	48	44,5833333333333	6,79205009717759
3	112	2018-01-01 00:00:00.0000000	NULL	5	16	2	12	2	130	133	32,375	32,5687040912916
4	171	2018-01-01 00:00:00.0000000	NULL	5	16	2	72	72	72	71	71,75	0,661437868077805
5	172	2018-01-01 00:00:00.0000000	NULL	5	16	2	155	160	155	130	148,625	14,0093594728025
6	17	2018-01-01 00:15:00.0000000	NULL	5	16	2	69	70	60	62	66,1458333333333	8,0792246905686
7	18	2018-01-01 00:15:00.0000000	NULL	5	16	2	51	50	50	48	48,1979166666667	2,69127239494983
8	112	2018-01-01 00:15:00.0000000	NULL	5	16	2	11	2	129	133	32,375	32,5687040912916
9	171	2018-01-01 00:15:00.0000000	NULL	5	16	2	72	72	72	71	71,75	0,661437868077805
10	172	2018-01-01 00:15:00.0000000	NULL	5	16	2	155	160	155	130	148,625	14,0093594728025
11	17	2018-01-01 00:30:00.0000000	NULL	5	16	2	70	70	58	62	66	7,24041512425408
12	18	2018-01-01 00:30:00.0000000	NULL	5	16	2	51	50	50	48	48,1979166666667	2,69127239494983
13	112	2018-01-01 00:30:00.0000000	NULL	5	16	2	12	2	130	133	32,375	32,5687040912916
14	171	2018-01-01 00:30:00.0000000	NULL	5	16	2	72	72	72	71	71,75	0,661437868077805
15	172	2018-01-01 00:30:00.0000000	NULL	5	16	2	155	160	154	130	147,875	13,7345241212865
16	17	2018-01-01 00:45:00.0000000	NULL	5	16	2	70	70	55	62	67,25	3,16008088053569
17	18	2018-01-01 00:45:00.0000000	NULL	5	16	2	51	50	51	48	47,7916666666667	2,42634557335075
18	112	2018-01-01 00:45:00.0000000	NULL	5	16	2	12	2	130	133	32,375	32,5687040912916
19	171	2018-01-01 00:45:00.0000000	NULL	5	16	2	72	72	72	71	71,75	0,661437868077805
20	172	2018-01-01 00:45:00.0000000	NULL	5	16	2	155	160	155	130	148,625	14,0093594728025
21	17	2018-01-01 01:00:00.0000000	NULL	5	16	2	70	70	56	62	66	7,24041512425408
22	18	2018-01-01 01:00:00.0000000	NULL	5	16	2	51	50	51	48	47,7916666666667	2,42634557335075
23	112	2018-01-01 01:00:00.0000000	NULL	5	16	2	12	2	130	132	32,375	32,5687040912916
24	171	2018-01-01 01:00:00.0000000	NULL	5	16	2	72	72	72	71	71,75	0,661437868077805
25	172	2018-01-01 01:00:00.0000000	NULL	5	16	2	155	160	155	130	148,625	14,0093594728025

Figure 5.7: Lagrad förutsedd data.

6
Analys och Diskussion

I detta kapitel kommer projektet att analyseras samt diskuteras.

6.1 Analys och diskussion av resultat

Vi lyckades att få ett fungerande end-to-end system, systemet består av en SQL-databas, en ML-modell, flertal API:er samt ett användargränssnitt.

Databas

En databas blev skapad för att lättare manipulera den historiska datan samt att ha möjligheten att kunna uppdatera den historiska datan. Datan blev också enklare tillgänglig för gruppmedlemmarna.

Modellen kan behöva tränas om, i framtiden, på grund av förändring av mänskligt beteende. Man kommer att behöva lagra ny data, baserad på det nya beteendet, för att kunna träna om modellen.

Databasen fick även funktionen att lagra förutsägelserna, då detta minskade svarstiden på anropet från klienten. För att svarstiden från ML-modellen var betydligt längre än svarstiden från databasen.

ML-modell

När antalet användare av systemet ökar kommer modellen förmodligen behöva tränas om, detta beror på att människors beteende kan komma att förändras. På grund av folks användande av tjänsten kan detta komma att förändra deras beteende som modellen ursprungligen var tränad på. Om detta händer kommer modellen behöva tränas om med den nya datan, som är baserad på det förändrade beteendet, som har blivit lagrade i databasen.

ML-modellen var förvånansvärt lätt att skapa, då ML Studio var användarvänligt. Efter en lättare genomgångskurs kunde man började experimentera och testa med den normaliserade datan från Västtrafik. Den svåra uppgiften med ML-modellen var hur man skulle hantera tidsfaktorn och få den att bli tränad i rätt sekvens. Det vill säga att man tränar modellen med data i kronologiskordning. Förståelsen kring time-lagged data var besvärlig till en början. Detta då man inte förstod helt hur

ML-processen fungerade. När man hade fått förståelse för hur time-lagged fungerade i ML-modellen. Fick man bättre resultat, samt att modellen lärde sig ett mönster istället för att memorera värden.

Under ML-modellens skapande, ifrågasattes om datamängden för en specifik parkeringsplats var tillräcklig för att träna en specifik modell. I efterhand har diskussioner lett till insikten att mängden data per parkeringsområden förmodligen var tillräcklig, för att träna områdesspecifika modeller.

Som man kan se i figur 5.4 och 5.5 i kapitel 5.2, sektion resultat från ML-modell så ger modellen ett relativt bra resultat för den första parkeringsplatsen(fig 5.4). Medan resultat för den andra parkeringsplatsen(fig 5.5) är markant sämre. Detta kan vara en indikation att en separat modell för den andra parkeringsplatsen hade eventuellt kunnat ge ett bättre resultat.

Backend

Systemet gjordes som ett webb-API, då olika plattformar ska kunna kommunicera med systemet. Klasstrukturen baserades utifrån att varje klass bara skulle vara ansvarig för en funktionalitet. Metoderna skulle eventuellt kunnat göras mer dynamiska för att reducera antalet metoder i klasserna.

Mycket tid lades på funderingar runt att göra modell API:et dynamiskt. Detta för att eventuellt kunna ha plug and play funktionalitet för ML-modeller. Man kunde inte komma på en lösning för plug and play funktionaliteten, då olika modeller kommer ha olika features och detta i sin tur gör att koden skiljer sig.

Då programmen ska köras var 15:e minut fanns det olika sätt att lösa detta på. Varianten att köra via Windows schemaläggare har visat sig skapa vissa problem, då programmet inte alltid körs med bestämt beteende. En annan lösning hade varit att sätta programmet i en oändlig while-loop som sover i 15 minuter mellan varje exekvering, dock hade detta i längden förmodligen inte varit hållbart då programmet alltid hade behövt vara igång. Samtidigt är oändlig while-loop betraktad som dålig praxis inom programmering. Då lösningen med while-loopen ansågs var det sämre alternativet, valdes Windows schemaläggare som lösning.

Användargränssnitt

Användargränssnittet valdes att göras i Python, då övrig kod är skriven i Python. Detta är ett väldigt enkelt användargränssnitt som tar in parkeringsområde samt datum och tid som parametrar från användaren. Användargränssnittet visar sedan resultatet i form av en boolean, som säger att det finns plats eller ej, för det sökta parkeringarsområdet för den sökta tiden.

Resultatet visas som en boolean trots att Decision Forest Regression valdes, då Västtrafik efterfrågade ett booleanskt svar vid senare tillfälle. Det visas även en naiv

validering över hur troligt det booleanska svaret stämmer, detta genom färgkodnin på orden High(grön), Medium(gul), Low(röd).

6.2 Andra funderingar och tankar

I detta delkapitel kommer det diskuteras om ett antal frågor som uppstod under projektets slut.

Python bästa språket för detta projektet?

Python valdes på grund av att man ville lära sig ett nytt språk, gruppmedlemmarna anser att det är ett språk som kommer att användas mer i framtiden och kommer vara bra att ha kunskap inom. Detta påverkade inte projektet något nämnvärt då Python var relativt lätt att lära sig. Python är ett användarvänligt språk och det finns många bibliotek som ger stöd för olika funktionaliteter. Pythons syntax är väldigt lätt att ta till sig.

Systemet har en del som är ett webb-API, och detta var väldigt enkelt att sätta upp i Python utan större problem. Däremot var Pythons standard GUI bibliotek, Tkinter, relativt svårt att lära sig. Detta gör att man borde kanske valt något annat bibliotek, alternativt annat programmeringsspråk som man har mer erfarenhet inom. I efterhand bedömdes det att det förmodligen hade varit enklare att ha byggt användargränssnittet i Java istället för Python. Då tidigare erfarenheter fanns inom Java Swing.

På grund av Pythons stöd för scripting tillsammans med bra bibliotek för ändamålet, anses detta varit ett bra val med undantag för användargränssnittet.

Varför inte behålla CSV?

Valet att lägga datan i databas istället för att fortsätta att använda CSV-filer var för att göra datan mer tillgänglig för gruppmedlemmarna, samt enklare att jobba med. Detta då man var tvungen att skicka de uppdaterade CSV-filerna, så att alla i gruppen hade rätt version att jobba med, för att lägga till fler features. När gruppmedlemmarna gjorde två olika saker, var man tvungen att göra en sammanslagning mellan de två olika CSV-filerna för att få den slutgiltiga CSV-filen.

Kommunikation mellan användargränssnitt och modell på ett annat sätt?

Det finns ett flertal olika alternativ till Flask, dock var Flask otroligt enkelt att komma in i vilket ledde till att behovet att utvärdera eventuella alternativ aldrig uppstod. Då Flask är ett standard bibliotek i Python, fanns det tillgängligt utan tredjepartsbibliotek. Andra alternativ till Flask är till exempel Django och Spring boot.

Andra alternativ liknande Azure tjänsten?

Azure blev projektet något tvingade till att använda för att Cybercom har köpt in tjänsten från Microsoft. Samt att Västtrafik ville att det skulle göras cloud-baserat och i Windows miljö. Då man inte hade val angående Azure valdes det att inte undersöka alternativa cloud-plattformar. Dock vid sökning av cloud-frågor förekom det referenser till AWS.

Alternativt versionshantingsverktyg?

Stash är ett av versionsverktyg som används på Cybercom. Stash är Git baserat versionhanteringssystem, dock så hade GitHub eller BitBucket varit bättre för detta projekt, då Azure erbjöd specialiserat stöd för dessa tjänster i jämnförelse med Stash.

6.3 Etik

Den etiska frågan i det nuvarande systemet är relativt liten då det inte registrerar specifika bilar, utan bara att en bil åker in eller ut ur parkeringsområdet. Dock ska man vara medveten om att detta fortfarande är lätt grad av övervakning. Det borde undvikas, på grund av säkerhetsskäl, att utöka systemet med kamror eller vikt känsliga sensorer. Detta skulle kunna användas till att kartalägga specifika bilars förflyttningar och därmed förarnas beteende och vanor. Om detta då inte hanteras på rätt sätt och det sker en säkerhetsläka så skulle obehöriga kunna utnyttja denna information.

6.4 Miljö

En mer utvecklade version av detta system, och användandet av det skulle kunna ha den eftersökta påverkan att minska mängden trafik i centrala Göteborg. Detta då långdistanspendlare kommer kunna få veta i förväg om det kommer finns parkering åt dem, så de kan välja att ta kollektivtrafik in till staden istället för att köra. Gruppmedlemmarna tror på att en väder prognos tillsammans med en förutsägelse gällande parkingsplats kommer kunna påverka en individ till att ställa sin bil vid en pendelparkering och åka kollektivt in till centrum eller arbetsplats i centrala Göteborg.

7
Förslag till fortsatt arbete

Här kommer en uppsättning av utvecklingsalternativ.

- Flytta schemaläggar hantering in i programmet istället för att använda motsvarande OS funktionalitet
- Förbättra ML-modellen

Bibliography

[1] Västtrafik, "Västtrafik AB - det här är vi", 2018. [Online], Tillgänglig: www.vasttrafik.se/om-vasttrafik/vasttrafik-ab/, Hämtad: 2018-05-30.

[2] Cybercom, "About the Group", 2018. [Online], Tillgänglig: www.cybercom.com/About-Cybercom/About-the-Group/, Hämtad: 2018-05-30.

[3] Microsoft Corporation, "Vad är Azure", 2018. [Online], Tillgänglig: www.azure.microsoft.com/sv-se/overview/what-is-azure/, Hämtad: 2018-05-22.

[4] Microsoft Azure, "Dokumentation om Azure Machine Learning Studio", 2018. [Online], Tillgänglig: www.docs.microsoft.com/sv-se/azure/machine-learning/studio/, Hämtad: 2018-02-06.

[5] Python, "What is Python? Executive Summary", 2018. [Online], Tillgänglig: www.python.org/doc/essays/blurb/, Hämtad: 2018-03-22.

[6] Chris Stephen, "Machine Learning: What It Is, and What It Isn't", 2018. [Online], Tillgänglig: www.threatvector.cylance.com/en$_u s/home/machine - learning - what - it - is - and - what - it - isnt.html$, Hämtad: 2018-05-22.

[7] Y. Kodratoff, *Introduction to Machine Learning*, San Mateo, USA:Morgan Kaufmann Publishers, Inc., 2014. [Online], Tillgänglig: www.books.google.se/books?id=AQyjBQAAQBAJprintsec=frontcoverhl=svv=onepageqf=false, Hämtad: 2018-04-09.

[8] L. Danielsson, "Tre typer av maskininlärning – så väljer du rätt", 2018. [Online], Tillgänglig: www.techworld.idg.se/2.2524/1.696999/maskininlarning, Hämtad: 2018-04-12.

[9] Rachna Devasthali, "Coefficient of Determination (R-squared) Explained", 2018. [Online], Tillgänglig: www.towardsdatascience.com/coefficient-of-determination-r-squared-explained-db32700d924e, Hämtad: 2018-06-20.

[10] Jason Brownlee, "An Introduction to Feature Selection", 2014. [Online], Tillgänglig: www.machinelearningmastery.com/an-introduction-to-feature-selection/, Hämtad: 2018-06-20.

[11] BusinessDictionary, "What is time series data? definition and meaning", 2018. [Online], Tillgänglig: www.businessdictionary.com/definition/time-series-data.html, Hämtad: 2018-04-10.

[12] Luleå tekniskaUniversitet, ”Vad är en databas”, 2008. [Online], Tillgänglig: www.ltu.se/centres/Centrum-for-langsiktigt-digitalt-bevarande-LDB/Vad-ar-en-databas-1.43398, Hämtad: 2018-04-16.
[13] TutorialsTeacher.com, ”What is Web API?”, 2018. [Online], Tillgänglig: www.tutorialsteacher.com/webapi/what-is-web-api, Hämtad: 2018-06-13.
[14] JSON, ”Introducing JSON”, 2018. [Online], Tillgänglig: www.json.org/, Hämtad: 2018-04-25.
[15] Uppsala Universitet, ”Versionshantering”, 2018. [Online], Tillgänglig: www.it.uu.se/edu/course/homepage/oopjava/vt16/html/f08-version.html, Hämtad: 2018-05-22.
[16] Martin Comstedt, ”Arbeta agilt – vad är agile?”, 2017. [Online], Tillgänglig: www.onbird.se/arbeta-agilt-vad-innebar-det/, Hämtad: 2018-06-12.

A

Appendix 1

A.1 Tidsplanering

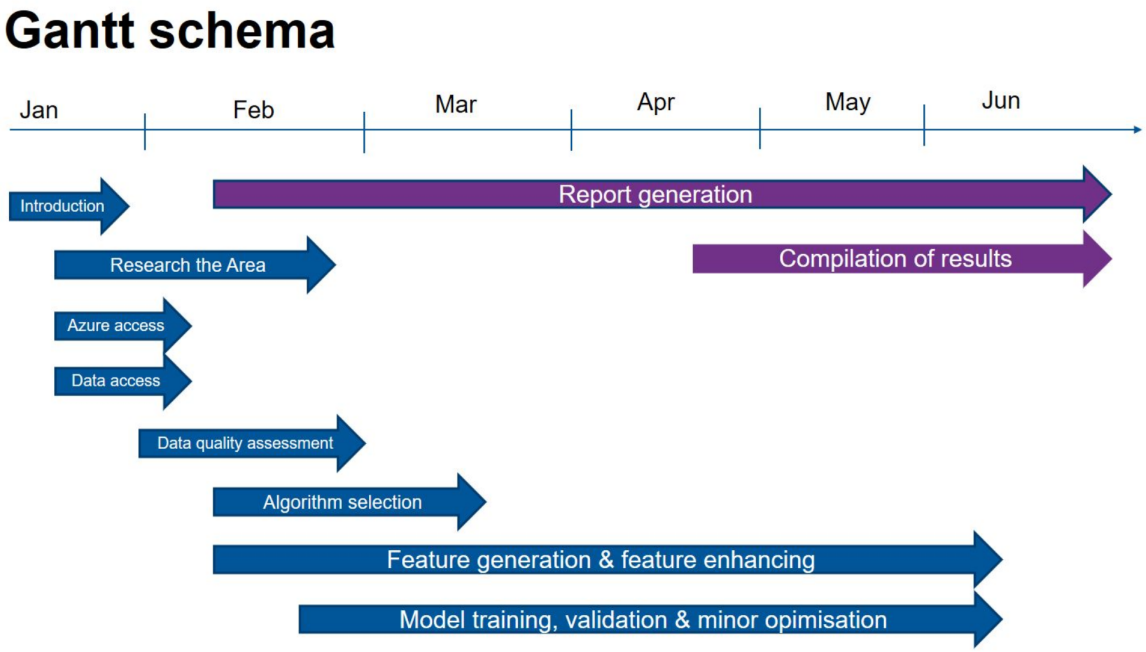


Figure A.1: Tidsplanering

A.2 UML

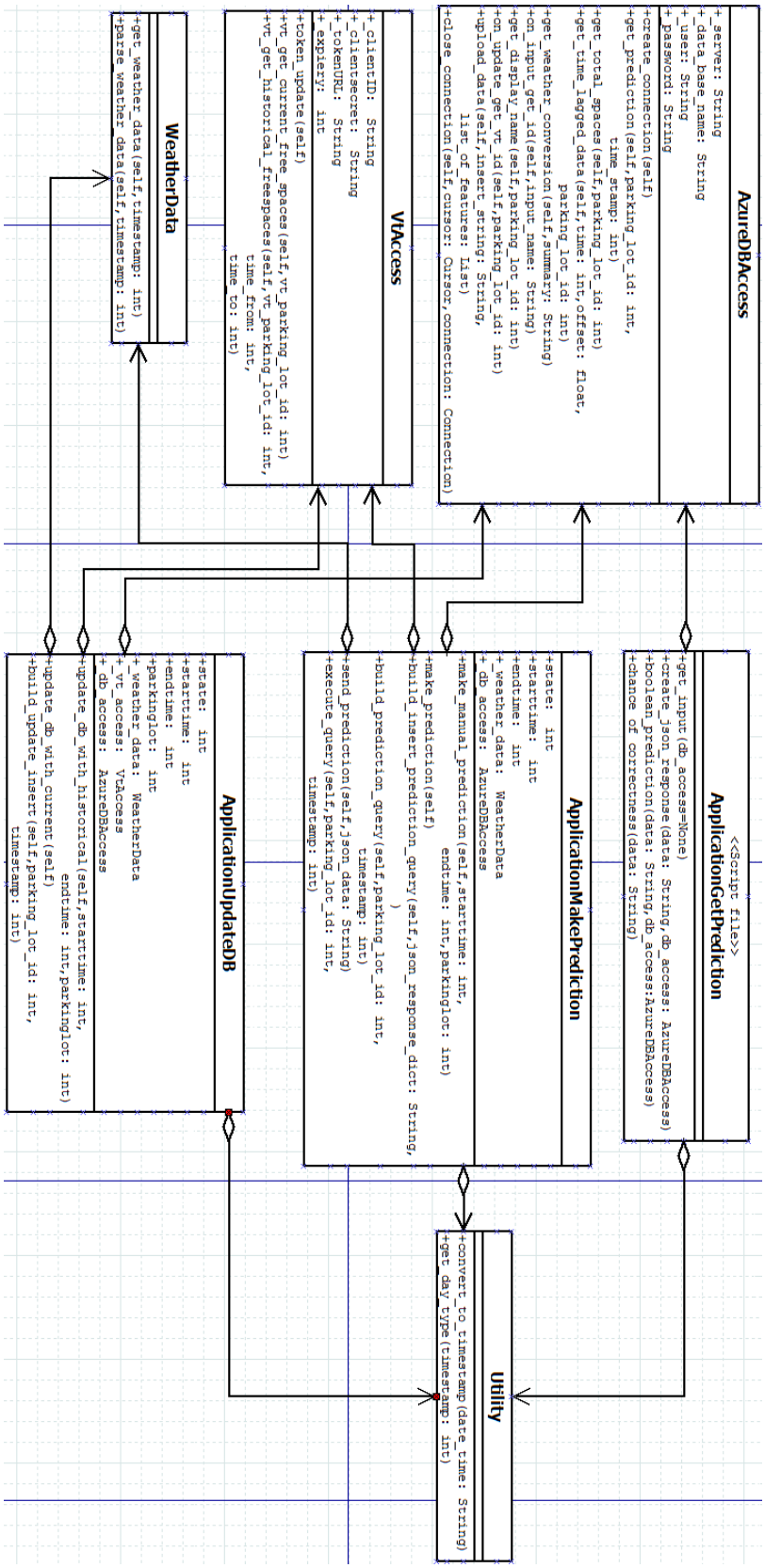


Figure A.2: UML över systemet

A.3 Förutsägelseresultat

Parkerings ID 18

Figure A.3: Jämförelse mellan faktisktvärde (övre graf) och förutsettvärde (undre graf) för parkering 18, under samma tidsperiod.

Parkerings ID 171

Figure A.4: Jämförelse mellan faktisktvärde (övre graf) och förutsettvärde (undre graf) för parkering 171, under samma tidsperiod.

Parkerings ID 112

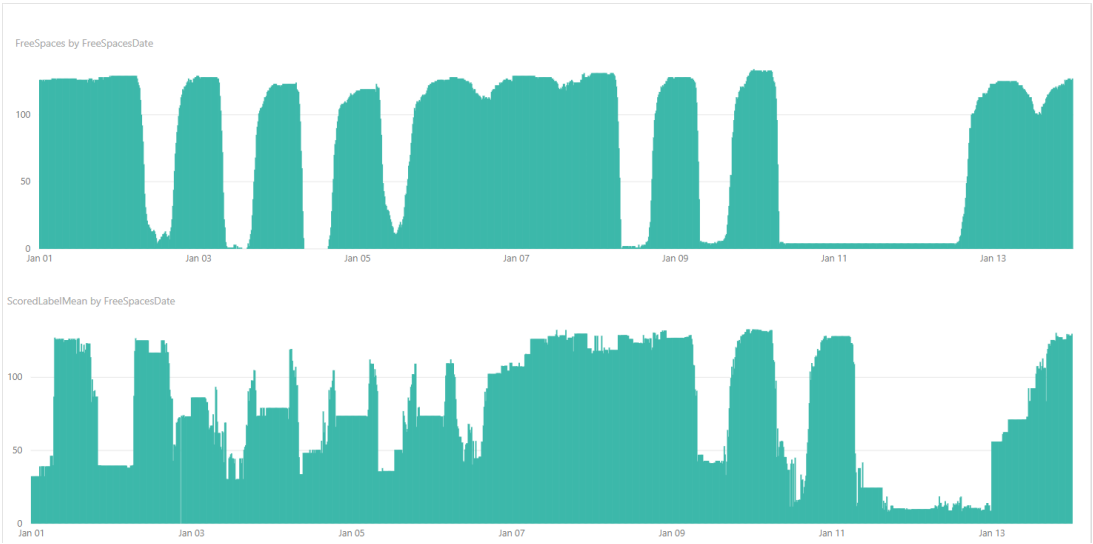


Figure A.5: Jämförelse mellan faktisktvärde (övre graf) och förutsettvärde (undre graf) för parkering 112, under samma tidsperiod.

A.4 Modell faser

Man börjar med att dra ut de moduler man ska använda sig av och kopplar samman dessa enligt figut A.6. Sedan kör man träningsprocessen, för att sedan sätta modellen i verifieringsstadiet. Efter verifieringsstadiet, figur A.7, aktiverar man ML-modellen och får då en API-nyckel som man använder för att kunna kommunicera med modellen.

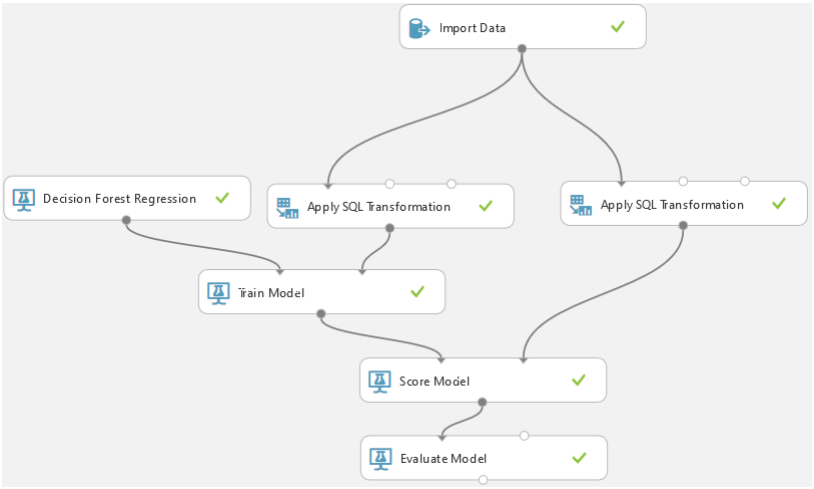


Figure A.6: ML-modellen i träningsstadie.

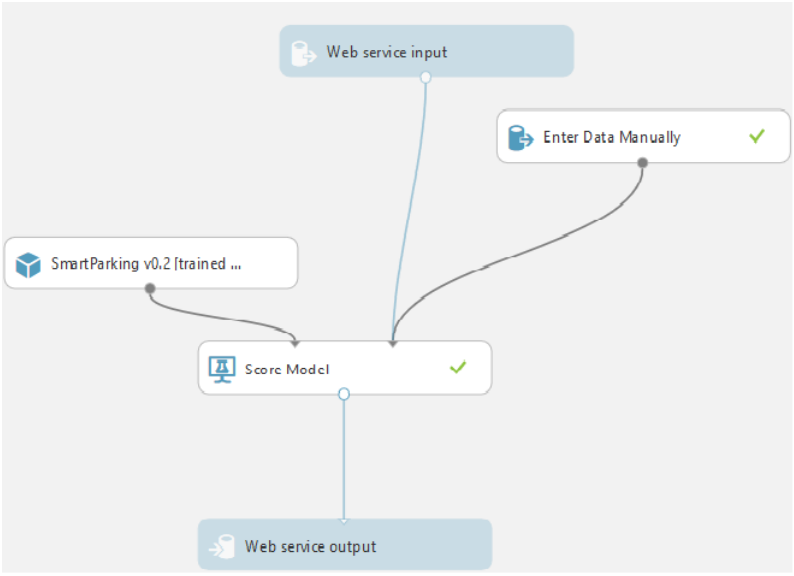


Figure A.7: ML-modell i verifieringsstadie.

A.5 Liknande projekt

D. H. Stolfi, E. Alba, X. Yao, *Smart Cities: Predicting Car Park Occupancy Rates in Smart Cities*, Cham, Schweiz: Springer International Publishing AG, 2017. [Online], Tillgänglig:
www.link.springer.com/chapter/10.1007/978-3-319-59513-9$_1$1, Hämtad: 2018-04-17.