Master of Science in Software Engineering
May 2018

Comparison and Prediction of Temporal Hotspot Maps

Andreas Arnesson
Kenneth Lewenhagen

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author(s):
Andreas Arnesson
E-mail: aar@bth.se
Kenneth Lewenhagen
E-mail: klw@bth.se

University advisor(s):
PhD Martin Boldt
Department of Computer Science and Engineering
PhD Anton Borg
Department of Computer Science and Engineering

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Context. To aid law enforcement agencies when coordinating and planning their efforts to prevent crime, there is a need to investigate methods used in such areas. With the help of crime analysis methods, law enforcement are more efficient and pro-active in their work. One analysis method is temporal hotspot maps. The temporal hotspot map is often represented as a matrix with a certain resolution such as hours and days, if the aim is to show occurrences of hour in correlation to weekday. This thesis includes a software prototype that allows for the comparison, visualization and prediction of temporal data.
Objectives. This thesis explores if multiprocessing can be utilized to improve execution time for the following two temporal analysis methods, Aoristic and Getis-Ord*. Furthermore, to what extent two temporal hotspot maps can be compared and visualized is researched. Additionally it was investigated if a naive method could be used to predict temporal hotspot maps accurately. Lastly this thesis explores how different software packaging methods compare to certain aspects defined in this thesis.
Methods. An experiment was performed, to answer if multiprocessing could improve execution time of Getis-Ord* or Aoristic. To explore how hotspot maps can be compared, a case study was carried out. Another experiment was used to answer if a naive forecasting method can be used to predict temporal hotspot maps. Lastly a theoretical analysis was executed to extract how different packaging methods work in relation to defined aspects.
Results. For both Getis-Ord* and Aoristic, the sequential implementations achieved the shortest execution time. The Jaccard measure calculated the similarity most accurately. The naive forecasting method created, proved not adequate and a more advanced method is preferred. Forecasting Swedish burglaries with three previous months produced a mean of only 12.1% overlap between hotspots. The Python package method accumulated the highest score of the investigated packaging methods.
Conclusions. The results showed that multiprocessing, in the language Python, is not beneficial to use for Aoristic and Getis-Ord* due to the high level of overhead. Further, the naive forecasting method did not prove practically useful in predicting temporal hotspot maps.

Keywords: hotspot, prediction, software packaging, similarity, multiprocessing

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Law enforcement agencies deal with a vast amount of crime and to a great extent they work with restricted resources, therefore is proactive policing of more interest instead of reactive policing. This means that law enforcement are interest in techniques to help them be proactive [10, 27]. As an example, The Swedish National Council for Crime Prevention (BRÅ) reports that in the year 2016 there was over 22000 reported burglaries in Sweden with a clearance rate of 4% [1]. As a tool to be more proactive in crime prevention, there exist techniques that aid the law enforcement when it comes to predicting crime. Two examples are *Hotspot analysis*, that focus on where crimes will occur and *spatiotemporal analysis* which focuses on where and when a crime will occur [10]. Bratton & Malinowsky address a need for a user friendly way of interpreting and visualizing the results from analysis tools, so police in the field can act on the information more directly, without having to rely on a crime data analyst [19].

Hotspot analysis as well as spatiotemporal analysis requires accurate data from crime scenes, which some times is hard to provide [28]. If the victim is present or if there are any witnesses when the crime occurs, the accurate time can be provided. Examples of such crimes are sexual assault and robbery. The victim can in such cases often provide the police with an almost exact time when the crime occurred. Other crimes, where it is harder to state an exact time, can be residential burglary or bike theft. One reason why it is hard to get an accurate time is that a crime, such as burglary, mostly occurs when the resident is not at home. The burglary then occurred sometime between when they left their home and the time they got back.

Hotspots that are based on events connected to one or more geographical locations and are represented in a hotspot map is called spatial hotspots and are often used in combination with a geographical information system (GIS) [27]. Another type of hotspot, that is based on events connected to a time stamp, i.e. when the crime was committed, is called a temporal hotspot.

[1] https://www.bra.se/bra-in-english/home/crime-and-statistics/residential-burglary.html

When analyzing a temporal hotspot, the result can be visualized in different resolutions, for example a diagram that displays the amount of crimes per hour, month, day or week and so on. The resolutions can in turn provide more advanced diagrams, when compared to each other, for example amount of crimes per hour compared against hour of the days of the week. The result can as such, be visualized as a matrix, where the y-axis could be weekdays and the x-axis could be hourly from midnight to midnight. Spatial hotspots and temporal hotspots are often used in combination with each other, to some extent, but the spatial hotspot alone has been far more researched and developed [7, 27]. Even though temporal data is equally valuable for predicting crime and can increase the accuracy of predictions compared to only using spatial data [13, 34].

Most crime hotspots display where or where and when certain events, such as burglaries or bicycle thefts, have occurred. Those hotspots are often represented by a map, based on a matrix, with different sized indicators of where the event has occurred. The matrix is filled with cells, where each cell represents a timespan or distance. For each event in the given area, a weight is constructed on each cells z-axis in the matrix, i.e it gradually stands out and eventually creating a cluster, which defines the hotspots.

There is a chance that the calculated values on the z-axis, the hotspots, appeared by coincidence. To measure the deviation and certify a degree of significance, one can test the result statistically. One such method that is appropriate for hotspots is a *Local Indicator of Spatial Association statistical test*, also known as a LISA statistical test [2, 24, 28].

Often a specific time of a crime cannot be provided instead a timespan is given, within which the crime could have occurred [6]. To aid the users of crime hotspots, one needs to create a temporal hotspot that uses both the best available practice to handle unknown time of the event or offense as well as the most appropriate statistic method. If the accuracy of the unknown temporal data can be improved, the hotspots can prove more useful when analyzing the result or predicting the future. With the ability to forecast where specific crimes may happen, the users can allocate their resources in a more beneficial and productive way.

This thesis researches temporal hotspots to help with amount of research on temporal hotspots compared to spatial, to investigate possibility for law enforcement to use naive methods to predict at what time a lot of crime occurs and to find ways to compare hotspot maps. Furthermore the authors found a research gap for each research question, more information about the research gaps can be found in section 1.1, Related Work. To answer the research questions in this thesis, a prototype in the programming language Python will be implemented. The different solutions to be evaluated are limited to what is possible with Python.

Solutions utilizing machine learning is outside the scope of this thesis because of time and knowledge limitations. An important aspect of crime prevention and analysis is spatial data, however this thesis will only research temporal data by reasons of amount of existing research on the respective areas. To what degree parallelism can be used to attain shorter execution time for the method for handling events with a range and the method for finding hotspots will be researched in this thesis as well. This is done in an attempt to improve the user experience by decreasing the execution time for creating hotspot maps which will minimize waiting times for the user. Another area to be researched is how two temporal hotspot maps can be compared. The authors have an ambition to release the prototype to the public. To do that the code for the prototype needs to be packaged, therefore the complexity of using existing packaging methods will be researched.

The prototype will have the following features; Creating a temporal hotspot map, comparing and visualizing difference between two hotspot maps and predict how a hotspot map will look. To demonstrate the diversity and capability of the prototype, a log file from a web server will also be used and evaluated. The reason behind the evaluation is to show that there are more usage areas, besides crime analysis, that can benefit from the prototype and that the use of temporal hotspots can prove itself useful in other areas.

1.1 Related Work

There exist a shortage of research on temporal hotspots as spatial hotspots have had a wider spread of influence in academics and within crime analytics and crime prevention [6, 7, 27]. This can be a consequence of the problems that emerge when data does not have known time of occurrence but instead consists of a timespan [7, 27]. This however does not mean that temporal information is less important in a crime- prevention and prediction aspect [6, 7].

Ratcliffe and McCullagh defines *aoristic crimes* as crimes without a known time , where instead there is a start time and an end time. Usually the point in time when a victim's possession was last seen and when it was noticed to be missing [20]. Ratcliffe and McCullagh also constructed the *Aoristic method* for analyzing aoristic crimes for the purpose of estimating when the crimes took place between the start and end time. The Aoristic method divides a crime event into the number of time units the start and end time of the crime spans. The value one is then divided by that number of time units, this is called an *aoristic value*. The Aoristic value is assigned to each time unit that the event spanned. A longer explanation can be found in section 2.2. Other notable methods are *Start, End, Midpoint* and *R*andom. Start use the start time of the crime. End use the end time of the crime. Midpoint use the time in the middle of the start and end time.

Random use a random time between start and end [7].

The accuracy of various aoristic temporal analysis methods, Start, End, Mid-point, Known-time, Random and Aoristic, has been tested on bike thefts [7]. 303 bike thefts were used as test data to estimate at what hour of the day the bike thefts took place. The Aoristic method, invented by Ratcliffe and McCullagh, was best at predicting the actual time of the thefts.

The question if the Aoristic method should be used where the duration of the crime is over 24 hours has been raised [26]. Furthermore the usefulness of the Aoristic method when the duration grows and the aoristic value approaches 0 has been questioned. Moreover the problematics of using small sample sizes, which increases the likelihood of misleading peak creation, was discussed [7].

The Aoristic method has also been compared to the Start, Stop, Average, Random and Aoristic extended [6]. Different temporal resolutions were used to discover if the unit or resolution used affected the result of the most accurate analysis method. The resolutions used were *hour of day*, *day of week*, *month in year* and *day of year*. Data of burglaries in Sweden were used for the experiments. Aoristic and Aoristic extended methods performed better than the rest. The Aoristic extended performed slightly better than Aoristic for the resolution *day of year* although the added precision did not necessarily out weigh the added complexity of the method. For the other resolutions the Aoristic method were most accurate. Furthermore it was research how different sample sizes affect the outcome of the temporal analysis methods. The different samples sizes used, including small, did not affect the Aoristic method negatively, which was suspected in previous research [7]. Additionally 29.6% of the data used for the experiments had a duration over 24h and therefore demonstrates that the Aoristic method can be used with data spanning over 24h [7, 26].

Ratcliffe compared various temporal analysis methods with different types of crimes. It was concluded that the Aoristic method can be used to "*smooth incongruities in the data set*" [28]. The Start and End methods should not be used for crimes of the types *break and enter*, *vehicle crime* and *malicious damage*, however the Aoristic and Mid-point methods are applicable to use and the Aoristic even more so.

It has been research how seasons and holidays affect the temporal distribution of completed burglaries in Denmark [31]. Additionally it was explored how temporal distribution differ between various types of properties. To analyze the temporal distribution of burglaries the *weighted estimate* method was used, which works the same way as the Aoristic method. Furthermore it was presented why the use of Start, Stop Average and Mid-point creates a false view of reality and

should not be used. The temporal distribution was analyzed with the resolutions month- and week of year, season, day of week and hour of day. According to the paper, burglaries occurs less frequent during and summer and spring than winter and fall. The peak during the winter is largely attributed to Christmas. Temporal analysis has also been researched in combination with spatial analysis [14, 22, 26, 27, 35]. Additionally spatial analysis has been researched in regards to crime prevention [5, 17].

To find out if the dataset consists of hotspots, an *local indicator of spatial association*, also referred to as LISA [2], is appropriate to calculate if the clusters have a statistical significance or if the clusters have appeared randomly. Anselin provided the name LISA as an appellation on several methods, used to show the significance of clustering values [2]. There are several statistic methods to use, such as *Local Moran's I, Getis-Ord Gi* and Local Geary's C*, with different qualities [2, 29]. Local Moran's I uses the covariance in the dataset, Local Geary's C measures the differences between the features in the dataset and Local Getis-Ord Gi* compares the local average to the overall average [29]. The method Local Getis-Ord Gi* is an appropriate method for this study, due to the fact that it is widely used in statistical analysis applications as well as in crime analysis [29].

Corcoran et al. builds an artificial neural network (ANN) to compare with linear regression and random walk to forecast crime trends in geographical crime hotspots, i.e. how many crimes happen in a specific area. The authors enhanced the ANN with a novel approach called Gamma test. The ANN method generally performed superior than the other two [16].

Crime forecasting has been done with ARIMA in China. With a time series containing data of crimes for 50 weeks a prediction with ARIMA was performed for the number of crimes in week 51. ARIMA results were compared to the predictions made with Simple Exponential Smoothing (SES) and Holt Exponential Smoothing (HES). It was concluded that ARIMA have better forecasting accuracy than the other two methods. To measure the accuracy of the forecasts, Root Mean Squared Error (RMSE) and Mean Absolute Percent Error (MAPE) were used [25].

Gorr et al. evaluates the accuracy of one month ahead temporal forecasting, in small areas, using ten models [36]. The naive models often used by the police are fitted against univariate time series models. Number of crimes per month in each area is predicted. The small areas used for data are precincts in Pittsburgh, PA. To achieve an accuracy maximum of 20% absolute forecast error an average crime count of at least order 30 or more is needed for a precinct. Another result was that the naive models used by the police performed worse than all of the time series models. HES used with city-wide data resulted in best accuracy. MAPE

was used to measure the accuracy of the forecasts. Most of the aforementioned forecasting techniques are advanced and can require extra expertise among the police to use. Gorr et al. also discusses the seasonal phenomenon of crime. Burglary rates are higher during the winter and the end of the year.

Using the naive lag 12 method to forecast is not an accurate method for predicting spatial crimes or how many crimes will happen in a month. In lag 12, the same month from last year is used to represent the same month this year. Instead research suggest that more recent data needs to be used to forecast future crime [29, 36]. Another naive model, Random walk, which is popular among the police is to use the month before. E.g. to forecast May assume it will be the same as April. Random walk has been proven to produce poor results predicting how many crimes will happen next month [36].

The advantage with the naive methods are that they are easy to use and understand. This means that the level of expertise among the police to use the naive methods can be low. However they do not produce very accurate results when it comes to number of crimes. No related work have been found where naive models have been used together with temporal hotspots.

Currently little research have been conducted in the particular area of comparing the overlap of hotspots. The measures for comparing the overlap of datasets is far more discussed [8, 9, 15, 37]. Choi et al. provide an extensive list of similarity and distance measures and groups them by the similarity of the measures themselves [8]. A lot of the measures are very similar in functionality because they are based on the same method, and their original purpose was to calculate the similarity in different fields such as biology, ethnology, taxonomy and chemistry etc [8, 15]. Hubalek lists forty-three similarity measures and evaluates them according to their correlation and functionality [15].

To summarize the related work, research on spatial hotspots has dominated temporal hotspots. There are a few methods for working with events with a timespan and the Aoristic method is the popular choice among researchers, because it provides the best result. Research on predicting spatial hotspots have overshadowed temporal hotspots. Both naive and advanced technologies have been used to predict spatial hotspots, where the advanced methods have shown better result than the naive. When it come to research on predicting temporal hotspots little have been done. Research exists about advanced prediction methods. Research on hotspot map overlap is scarce while overlap on datasets are not and dataset overlap methods are applicable to the structure of hotspots maps. Furthermore, to the authors knowledge, no research has been found for the following areas: comparing two temporal hotspots and comparison of different software packaging methods. Lastly if parallelism can be used to accomplish shorter execution time

for the Aoristic method or Getis-Ord*, is another area which have not yet been researched.

1.2 Aims and objectives

The greater aim of this thesis is to expand the research on temporal hotspot maps by evaluating different methods for temporal hotspot maps. The evaluated methods will be built into a prototype which will be used to evaluate the methods.

Several smaller aims have been derived to achieve the greater aim. The first is to be able to create temporal hotspot maps based on events with either a timespan or known time. The second is to take advantage of parallelism to increase performance. Thirdly, comparing temporal hotspot maps. Fourthly, predicting temporal hotspot maps by analyzing previous hotspots and the last aim is to find a packaging method for the prototype which is not complex for both developers and end users.

In order to complete the aims presented above the following objectives will be carried out:

1. Select a technique to discover hotspots.
2. Select the best method for handling events with a timespan.
3. Explore how to utilize parallelism with the selected technique to discover hotspots and the method for handling timespan data in the prototype.
4. Examine how to compare the overlap in two hotspots.
5. Investigate how to predict a hotspot.
6. Research different methods Python code can be packaged.

1.3 Research questions

The following research questions will be answered in this thesis:

RQ 1 To what degree can multiprocessing be utilized to increase performance of the methods Getis-Ord* and Aoristic compared to a serial implementation?

With RQ1, Getis-Ord* and Aoristic will be implemented both utilizing multiprocessing and not utilizing it. Performance testing will be done on each implementation to see what performance difference there is. The motivation

behind RQ1 to discover if multiprocessing is appropriate to increase performance of the algorithms, objective 3. To do this, first a method for creating hotspot, Local Getis-Ord Gi*, and a method for handling time span events, Aoristic, need to be selected, objective 1 and 2. From here Local Getis-Ord Gi* will be written as Getis-Ord*. Performance describes execution time of the algorithms.

RQ 2 To what extent can different hotspots be compared and provide significant similarity, difference and overlap?

The incentive for RQ2 is to understand how two hotspot maps differ and can be compared, objective 4. The comparison will consist of the similarity as well as the difference in the calculated hotspots. The motivation behind RQ2 is that the user, for example law enforcement, are provided the functionality to compare hotspot maps and find out to what extent the hotspot maps are similar. The user can then create two hotspot maps and see to what extent the crimes occurs at the same time and day. By comparing two hotspot maps, the user can also see if there is a trend in burglaries at some time of day and take precautions and act accordingly. The motivation behind visualizing the overlap is that the user easy can see which time and day crimes occurred at the same time on a chosen confidence level. The result of RQ2 can also be used to give input on RQ3, and compare the predicted hotspot map with the actual outcome.

RQ 3 How accurately can historic data be used to predict future temporal hotspots using naive methods?

RQ3 is motivated in objective 5, to investigate how a hotspot map can be predicted by analyzing matrices that contain temporal data for previous time units to one that is to be predicted.

RQ 4 How do established software packaging methods compare in regards to usage complexity for developers and end users?

Objective 6 will be accomplished in RQ4, to examine different packaging methods so the authors know what is needed to package the prototype.

Chapter 2

Background

This chapter will introduce and explain methods and concept the reader needs to understand when reading this thesis.

2.1 Parallelism

In traditional programming, the execution of processes are made *sequentially*, i.e. one process at a time [3]. Consider, for example, a list where each item should be passed to a function through a loop. The list is then treated as a queue and only when one function call is complete, the next element in the list can be sent to the function, as in figure 2.1.

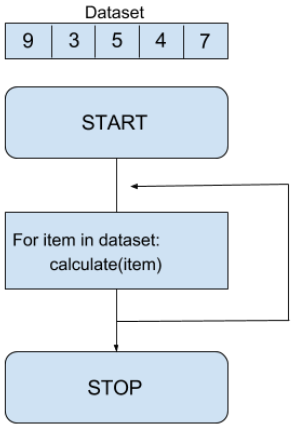


Figure 2.1: Illustration of sequential.

Parallelism could be beneficial when there is a set of processes, that has to be carried out simultaneously. The execution time can be reduced because the processes are divided among the computer's resources, for example one process for each processor. This way a set of processes can be carried out at the same time, as seen in figure 2.2, if the code structure allows it to run separately. At some point the processing power to create new processes surpasses the execution

time of the actual process, and creating an overhead, which slows down the parallelism [3].

The two main types in parallel computing is *multiprocessing* and *threading*. Multiprocessing makes use of a computer's multiple cores, i.e. processors. The processors can be used as resources when dividing the workload, which makes it possible to run multiple processes simultaneously. When using threading, the workload is divided so each process is running on its own thread [3]. A downside with threading is that Python is restricted with GIL (Global Interpreter Lock). GIL protects Python objects from being accessed and preventing the use of multiple threads when executing bytecodes simultaneously [1].

Due to the restrictions with GIL, this thesis will only explore the possibilities with parallelism through multiprocessing.

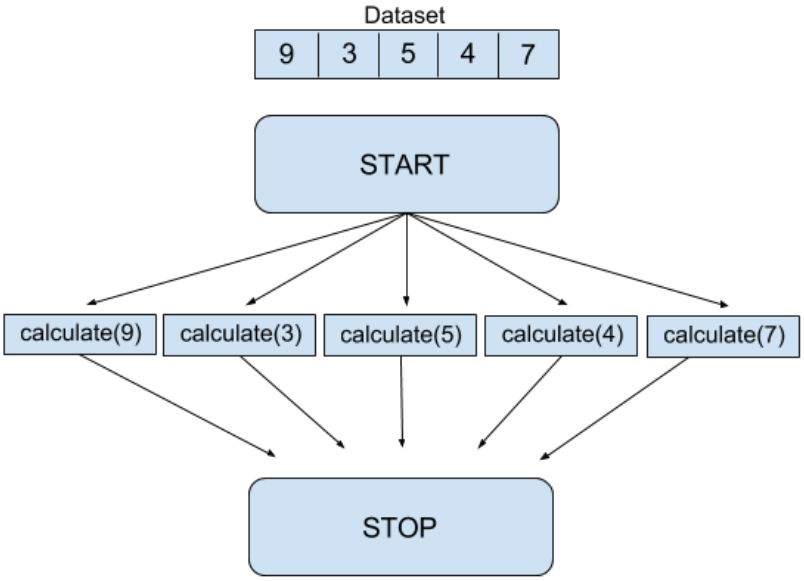


Figure 2.2: Illustration of parallelism.

2.2 Temporal analysis method

The Aoristic method was selected to be used in the prototype. All found related work concluded that the Aoristic method performs best [6, 7, 28, 31].

The Aoristic method "*calculates the probability that an event occurred within given temporal parameters, and sums the probabilities for all events that might*

[1] https://wiki.python.org/moin/GlobalInterpreterLock

have occurred" [28].

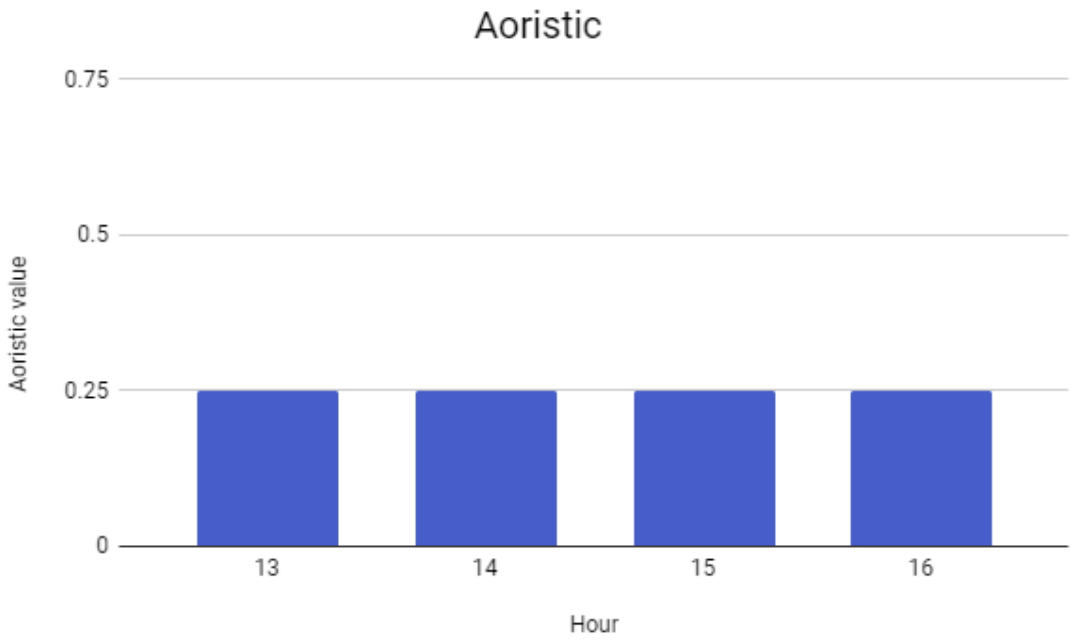


Figure 2.3: Spread of the Aoristic value based on a crime which spans four hours, kl. 13:25–16:05

The Aoristic method needs a temporal unit, e.g. hours or days, and events with a duration, in our case the events are crimes. The method calculates an *aoristic value* based on the number of units the duration spans. Each crime (event) is associated with an initial value of 1.0, the value is then evenly divided by the number of units the crime spans, n, resulting in an aoristic value distributed to each unit. After all crimes have been processed the probability of when crimes have occurred is indicated by summarizing the values for all units.

For example if the unit is *hours* and a crime start at the time *13:25* and end at *16:05* the crime have a duration of four hours. 1.0 is then divided by four, n, and each unit, hour, the duration spans, 13, 14, 15 and 16 is increased by the aoristic value, 0.25, i.e. $\frac{1.0}{n}$. This can be seen in figure 2.3. The equation does not take into account if the crime partially or fully spans a unit. A complete unit has the same value as a partial unit. In other words processing a crime with the start time of *13:00* and an end time of *16:00* produces the same output as the previous example.

By representing the time of the crimes with two temporal units (each with a different resolution), consequently creating a matrix, a more comprehensive understanding can be had of the temporal distribution of when the crimes take

place. Such combinations, of temporal resolutions, can look as follows:

- weekday by hour-of-day (7x24)
- weekday by month (7x12)
- weekday by week-in-the year (7x52)
- month by hour-of-day (12x24)

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.5	0	0	0	0	0	0	0	0	0
0	0	0.5	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Table 2.1: Matrix, with resolution month by hour-of-day, with an aoristic crime which start 01-03-2014 14:45 and end 01-03-2014 15:25.

As an example, with the resolution *month by hour-of-day* and using the Aoristic method on a crime with the start date *01-03-2014 14:45* and the end date *01-03-2014 15:25*. The timespan of the crime is expressed in the smallest unit of the two in the resolution, hour-of-day, in this example two hours, 14:45-15:25. The value two is then used in the Aoristic equation, $\frac{1.0}{2} = 0.5$ and 0.5 is the aoristic value for this crime. In the matrix where each hour and month, the crime spans, intersect, the value is increased with the aoristic value. In the example, matrix[03][14] and matrix[03][15] gains 0.5. The matrix can be seen in table 2.1

2.3 LISA statistical method

LISA stands for Local Indicators of Spatial Association. LISA is an appellation of statistical methods. The purpose of LISA is to provide an indication on the significance of the clustering values, whereas they have appeared by chance or not [2]. The method Getis-Ord Gi* is an appropriate method for this study, due to that it is widely used in statistical analysis applications as well as in crime analysis [29].

The Getis-Ord* is given as:

$$G_i^* = \frac{\sum_{j=1}^{n} w_{i,j}\, x_j - \bar{X} \sum_{j=1}^{n} w_{i,j}}{S\sqrt{\frac{n\sum_{j=1}^{n} w_{i,j}^2 - (\sum_{j=1}^{n} w_{i,j})^2}{n-1}}}$$

Here is x_j the value for feature j. $w_{i,j}$ is the spatial weight between feature i and j, n is equal to the total number of features and:

$$\bar{X} = \frac{\sum_{j=1}^{n} x_j}{n}$$

$$S = \sqrt{\frac{\sum_{j=1}^{n} x_j^2}{n} - (\bar{X})^2}$$

Getis-Ord* statistic returns a *z-score* that holds a value for each feature representing the standard deviation. If the value is high and is surrounded by high values in their neighborhood, it is considered a hotspot. The same rule applies for low numbers. If a features' value is low and is surrounded by other low values, it is called a coldspot. The pseudo code for the implementation of Getis-Ord can be found in chapter 3, algorithm 2.

The result of Getis-Ord* is represented in a matrix, where each feature has a z-score. The matrix is displayed as a hotspot map.

To find out to what degree or certainty a hotspot is significant, in relation to the overall data, one can use *confidence intervals*. A confidence interval is a range of values that the z-score lies within, based on a *p-value*, probability value. The most common confidence level to test with is 95%, but also 90%, 99% and 99.99% are used [29]. The p-value is the probability that the analysis result has been created randomly, in this case called the *null hypothesis*. The null hypothesis reflect a negative effect of the analysis, in this case that found hotspot has been created randomly. The p-values are given as 1 minus the significance level. For example, a 95% confidence level has a p-value of 0.05[2].

[2]http://pro.arcgis.com/en/pro-app/tool-reference/spatial-statistics/what-is-a-z-score-what-is-a-p-value.htm

The statistical significance is provided as a table with the confidence intervals showing the ranges where a z-score has to fall within for respective confidence level as shown in table 2.2. The levels tested are 90%, 95% and 99% confidence.

Confidence level	p-value	z-score range
90%	0.10	-1.254 $<$ z-score $<$ +1.254
95%	0.05	-1.432 $<$ z-score $<$ +1.432
99%	0.01	-2.123 $<$ z-score $<$ +2.123

Table 2.2: Table representing confidence levels

The calculation of Getis-Ord* utilizes a method for finding each cell's adjacent cells, also called a neighborhood. The selected method is *Queen's case*, and is motivated by the fact that it includes all eight adjacent cells and provides a higher connection to the given cell than other methods such as Rooks case [24]. The pseudo code for the implementation of Queens case can be found in chapter 3, algorithm 1 In table 2.3 the neighborhood for the cell with the value 9 is calculated with a distance of 1:

1	1	5	2	4
2	4	8	6	7
3	3	9	2	11
1	4	3	2	4
4	2	7	3	5

Table 2.3: Table representing Queen's Case

To handle the cases where the neighbors are out of bounds, the method will continue in the adjacent cells. If the matrix consists of a resolution of *days* and *hours* and the cell to be calculated is 23:00 on a Sunday, it is of interest to get all of the connected cells i.e. 00:00 on Monday, where Monday is the first column and Sunday is the last. Table 2.4 is representing a hotspot map, where the y-axis is hours and the x-axis is weekdays. The cell with the value 11 is calculated:

1	1	5	2	4
2	4	8	6	7
3	3	9	2	11
1	4	3	2	4
4	2	7	3	5

Table 2.4: Table representing out of bounds

2.4 Comparison

Based on previous research, a similarity coefficient that measures overlap, works well with one-dimensional binary input. The compared matrices will therefore be flattened, so a hotspot map is represented by a one-dimensional list. Each hotspot or coldspot from the original dataset will be represented by a 1 and the absence of a hotspot or coldspot will be represented by a 0. The hotspot maps will also be compared three times, both hotspots and coldspots to get a total overlap, only hotspots and only coldspots.

2.4.1 Similarity coefficients

One major and important difference between the similarity coefficient measures is whether they take negative matches in consideration [8, 37]. According to Wong and Kim, this is an important decision to make when choosing similarity coefficient because for some data, a negative match will be calculated as similarity [37]. For this thesis comparison functionality, a negative match will occur if both indexes in the compared lists is marked with a 0, indicating an absence of a hotspot. Negative matches will not be included in comparisons in this thesis because if a negative match would count as a match, the result will always be 100% on matrices with boolean values. Some of the most commonly used similarity coefficients that does not take negative matches into consideration are *Jaccard Index*, *Sørensen–Dice*, *Kulczynski* and *Ochai* [8,37]. These four measures generate a resulting score of 0-1 and generally work well [15]. To get a percentage, one can multiply the score by 100. To get the dissimilarity, one withdraws the score from 1.

From this point on, the following definitions are used, note that the negative match is not included. True-True, True-False and False-True are explained in figure 2.4.

- a = True-True, TT, both values are 1
- b = True-False, TF, first value is 1, second value is 0
- c = False-True, FT, first value is 0, second value is 1

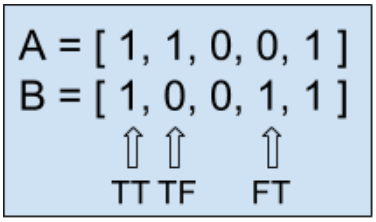


Figure 2.4: Explanation of TT, TF, and FT

Jaccard Index

Jaccard Index as defined in Equation 2.1, divides all a matches by the sum of all a, b and c matches, giving a score ranging 0.0-1.0. There is no weight on either variable, which produces a result of the exact matches [8, 9, 15, 37].

$$Jaccard = \frac{a}{a+b+c} \tag{2.1}$$

Jaccard index is implemented as shown in appendix D, algorithm 12.

Sørensen-Dice Index

Sørensen-Dice Index as defined in Equation 2.2, is very similar to Jaccard. The difference is that Sørensen-Dice calculates the matches on a with more weight hence the amount of a is multiplied by two and divided by the same result added with all b and all c [8, 9, 15, 37].

$$Sorensen - Dice = \frac{2a}{2a+b+c} \tag{2.2}$$

Sørensen-Dice index is implemented as shown in appendix D, algorithm 13.

Kulczynski Index

Kulczynski Index as defined in Equation 2.3, handles the weight differently than Sørensen-Dice Index. Kulczynski calculates the arithmetic mean probability that if b and c exists in one object, they exists in the other object to.

$$Kulczynski = \frac{1}{2}\left(\frac{a}{a+b} + \frac{a}{a+c}\right) \tag{2.3}$$

Kulczynski index is implemented as shown in appendix D, algorithm 14.

Ochai Index

Ochai Index as defined in Equation 2.4, is similar to Kulczynski index, but works with the geometric mean probability. To be considered similar by Ochai Index, the compared objects must share a high amount of variables [8, 9, 15, 37].

$$Ochai = \frac{a}{\sqrt{(a+b)(a+c)}} \tag{2.4}$$

Ochai index is implemented as shown in appendix D, algorithm 15.

Delta calculation

In this thesis, the purpose of the similarity measure is to calculate the overlap of hotspots and coldspots. The authors believe that the users of the prototype may benefit of more types of measures, such as the percentual delta [3] change of the amount of occurrences between the compared hotspot maps. This functionality will also be implemented as a measurement of comparison. The implementation of the delta functionality can be found in algorithm 7 in section 3.4.

2.5 Naive forecast method

To forecast temporal hotspots, this thesis introduces a naive method which sums together matrices that contain temporal data, like the one in table 2.1. The new produced matrix is then transformed to a hotspot map and resulting map contains the predicted hotspots. The method needs a number of matrices with the same resolution as input and the number of matrices used is called *time window*, the data they are comprised of is referred to as test data. As an example, to predict a hotspot map for burglaries in the month of April, burglary data for as many previous months as the time window is the input. In this example a time window of three will be used, this means that data for the months of January, February and March is needed. Each months data is put in a matrix and since the data has a range the Aoristic method is used to put the data in matrices. This results in three matrices, one for each month. The three matrices are added together resulting in a new matrix with the same resolution. The matrix is sent to Getis-Ord* and that results in a hotspot map which represents April.

Pseudo code can be found in algorithm 8 in section 3.5.

[3] https://sciencing.com/calculate-delta-percentage-8475192.html

Chapter 3

Implementation

This chapter displays Pseudo code for Getis-Ord*, the Aoristic method, similarity measurements, delta calculations and the Naive forecast method. The algorithms are presented in the order they appear in the chapter with results, chapter 5. Implementations that contain many lines of code are shown in appendixes.

3.1 Getis-Ord* pseudo code

This section is divided into two parts. The first part is the pseudo code for the implementation of finding each values corresponding neighborhood using Queens case. The second part is the pseudo code for the implementation of Getis-Ord.

The Queens case function is called on in a loop over the matrix containing all values. The parameters y and x is representing each cell in the matrix. The neighborhood is then calculated by taking the y and x position for each iteration and returning a new matrix containing the surrounding values i.e. the neighborhood.

Algorithm 1 Queen's case

```
{input is the matrix, y and x position, distance}
iterations = distance + distance + 1
result = empty matrix of size distance x distance
data = matrix from input
startY = (y - distance) % number of rows
startX = (x - distance) % number of columns
counterY = 0
while counterY < iterations do
  counterX = 0
  startX = (x - distance) % number of rows
  while counterX < iterations do
    result[counterY].append(data[startY][startX])
    counterX += 1
    startX += 1
    startX = startX % number of rows
  end while
  startY = (startY + 1) % number of rows
  counterY += 1
end while
{the variable 'result' is a matrix and holds the neighborhood}
```

To calculate the z-scores of each feature Getis-Ord* is implemented. When looping through the matrix holding all values, a call to the function that returns the neighborhood is made. That neighborhood is then used to calculate the necessary variables that Getis-Ord needs to finally find the corresponding z-score of all elements in the matrix.

The pseudo code is presented in algorithm 2.

Algorithm 2 Getis-Ord*

```
for x, y in matrix do
  neighbors = neighborhood for matrix[y][x]
  neighbors_sum = total sum of neighborhood
  square_weight = total square weight, based on all features in neighborhood
  {feature**2}
  j_count = number of features in neighborhood
  numerator = neighbors_sum - (mean * j_count)
  S = square root of (matrix square sum / number of elements in matrix) -
  mean**2
  denominator = S * square root of (number of elements in matrix * j_count)
  - square_weight**2) / number of elements in matrix
  z-score = numerator / denominator
  {The variable z-score holds the z-score for matrix[y][x]}
end for
```

Furthermore, the Python package *numpy* [1] is used for managing the matrices. It holds a great range of qualified built-in functions for calculating various statistical methods.

3.1.1 Getis-Ord* multiprocess pseudo code

The multiprocessing for the Getis-Ord algorithm is implemented with pool and the map method, because it has to utilize the possibility to pass arguments to the function and keep the result ordered. The input data is a list with tuples, containing the x and y positions from the matrix. The map method automatically creates chunks from the iterable list that are passed to the process pool[2]. The pseudo code is presented in algorithm 3.

Algorithm 3 Getis-Ord* parallelism

```
temp = list()
for y in range(number_of_rows) do
  for x in range(length_of_row) do
    temp.append((y, x))
  end for
end for
for all Pool() as p do
  matrix = p.map(get_neighbours((y, x)))
end for
{matrix holds the result}
```

[1] http://www.numpy.org/

[2] https://docs.python.org/3.6/library/multiprocessing.html#multiprocessing.pool.Pool.map

3.2 Aoristic implementation

In this section the different implementations of the Aoristic method, used for RQ1, will be displayed in pseudo code. The pseudo code is hard coded to use matrices with the hour-of-day resolution. If another resolution is to be used, one needs to change the size of the matrix that is created on line 2 in algorithm 4. There are three algorithms for the Aoristic method and they are presented in the following order: Sequential, Multiprocessing with shared memory and Multiprocessing. Multiprocessing with shared memory is presented after Sequential because they are very similar and therefore only how Multiprocessing with shared memory differ from Sequential is presented here. The entire Multiprocessing with share memory algorithm can be found in algorithm 9 in Appendix B.

3.2.1 Sequential pseudo code

Algorithm 4 contain pseudo code for the Aoristic method, it creates a matrix with the size 7x24, resolution weekday by hour-of-day, then loops over all events, calculates each event's aoristic value and adds the value to the appropriate cells.

Algorithm 4 Aoristic method

```
1: {input is events as Datetime objects in a list}
2: matrix[7][24] = 0 {Init matrix of resolution Days-of-week x Hours-in-day}
3: event_counter = 0
4: while event_counter < events.length do
5:     event = events[event_counter]
6:     nr_hours_span = ceiling((event.total_seconds()/3600))
7:     aoristic_value = (1 / nr_hours_span)
8:     duration_counter = 0
9:     while duration_counter < nr_hours_span do
10:        x = event.get_weekday()
11:        y = event.get_hour()
12:        new_value = matrix[x][y] + aoristic_value
13:        matrix[x][y] = new_value
14:        duration_counter ++
15:        event.add_hour(1)
16:    end while
17:    event_counter ++
18: end while {the variable 'matrix' now contains the temporal probability of
    when the events took place and is ready to be analyzed with Getis-Ord*}
```

3.2.2 Multiprocessing shared memory pseudo code

The Aoristic calculations looks the same here as for Sequential implementation. The difference in this implementation is that all the events have been divided between a number of processes. Each process adds the calculated aoristic values to the same matrix. As mentioned before, below only the code that differ, from the sequential algorithm 4, is displayed.

The following symbols, table 3.1, are used to understand the altered code below:

Symbol	Meaning
...	Unchanged code
x:	Line unchanged
x*	Line changed
→	New line

Table 3.1: Table explaining symbols

Algorithm 5 Aoristic method, multiprocessing with shared memory

```
...
3: event_counter = 0
→ shared_matrix = create_shared_matrix(matrix){number of events for each
process to process}
→ chunk = events.length / number of processes
→ for each process do:
4*      while event_counter < chunk
5:    event = events[event_counter]
            ...
11:         y = event.get_hour()
12*         new_value = shared_matrix[x][y] + aoristic_value
13*         shared_matrix[x][y] = new_value
14*         duration_counter ++
15:         event.add_hour(1)
           ...
18:     end while
→ end for
{the variable 'shared_matrix' now contains the temporal probability of when
the events took place and is ready to be analyzed with Getis-Ord*}
```

3.2.3 Multiprocessing pseudo code

This implementation is very long and can therefore be found in algorithm 10 in Appendix B. Here the algorithm only will be described breifly and how it differs from the other two implementations. The implementation is split into two parts. The first part is a lot like the entire algorithm 5, the key difference is that since there is no shared memory the processes cannot add each event's aoristic values to the same matrix. Instead a new matrix is created for each event that is processed. These matrices are then used in part 2 where all matrices are combined to one matrix which is the result of the method. In other words part 1 of the algorithm is comparable to the code in Algorithm 5 and 5, but because each process does not share memory part 2 is added to compensate. Part 1 will be referenced to as "Parallel calculation" in later sections.

3.3 Implementation of measures

The implementation of the *setup_ data()* function is used by all similarity implementations to get the respective values for TT, TF and FT. The input data is two one-dimensional lists with numeric data. By implementing this custom functionality, one can use the similarity indexes on coldspots as well. As an example, matches with -1 and -1 results in an addition to TT where an all binary list would not consider it a match, if only ones and zeros are to be calculated. The setup for all similarity measures is implemented as shown in algorithm 6.

Algorithm 6 Data setup for similarity measures

```
true_true = 0
true_false = 0
false_true = 0
# left and right are flattened numpy arrays
if len(left) == len(right) then
  for index, val in enumerate(left) do
    if left[index] in (1, -1) and right[index] == 0 then
      true_false += 1
    else if left[index] == 1 and right[index] == 1 then
      true_true += 1
    else if left[index] == -1 and right[index] == -1 then
      true_true += 1
    else if left[index] == 0 and right[index] in (1, -1) then
      false_true += 1
    else if left[index] == -1 and right[index] == 1 then
      true_false += 1
    else if left[index] == 1 and right[index] == -1 then
      true_false += 1
    end if
  end for
end if
return (true_true, true_false, false_true)
# The variables true_true, true_false and false_true holds the values for a, b
and c
```

3.4 Delta implementation

The following algorithm shows the implemented method for calculating the delta.

Algorithm 7 Implementation of delta calculation

```
delta =  "amount": 0, "occ": 0
old_nr_of_hotspots = np.count_nonzero(np.isnan(old))
new_nr_of_hotspots = np.count_nonzero(np.isnan(new))
old_sum = np.nansum(old)
new_sum = np.nansum(new)
delta["amount"] = round(((new_nr_of_hotspots - old_nr_of_hotspots) /
old_nr_of_hotspots) * 100, 2)
delta["occ"] = round(((new_sum - old_sum) / old_sum) * 100, 2)
```

3.5 Naive forecast method

The Naive forecast method first creates the matrix that will hold the result, the predicted matrix, that will be used to create a hotspot map with Getis-Ord*. Then it loops through the test data and add each cell's value to the same cell in the forecasted matrix.

Algorithm 8 Naive forecast method implementation

```
{input is a number of matrices with event data, called test data}
forecasted_matrix = matrix(number_of_rows, number_of_columns)
for matrix in test_data do
  for y in range(number_of_rows) do
    for x in range(number_of_columns) do
      forecasted_matrix += matrix[y][x]
    end for
  end for
end for
hotspot_map = create_hotspot_map_Getis(forecasted_matrix)
```

Chapter 4

Method

Various methods have been considered when planning this study. Considering the requirements and constraints for the study the authors have selected to rely on the experimental method for RQ1 and RQ3. For RQ2 the authors chose a case study and for RQ4, a theoretical analysis. For RQ1 and RQ3, experiments were selected instead of for instance case studies and simulation. Case studies are discarded for these RQ's because the topics are researchable as they are and are focused on conducting experiments. Simulation is discarded for these RQ's because the study is not conducted as a social progress or in the field of social behaviorism. For RQ2, the reason for conducting a case study is that the comparison uses two sets of data, where several cases will be used and later evaluated. Additionally, for RQ4, a theoretical analysis was selected over systematic literature review and literature review due to the lack of research papers on packaging Python code. The method then aims at finding relevant information in official documentation. Below is a description of the research methods used for answering each of the research questions (RQs). The case study and experiment 1 and 2 will be performed with the temporal resolution weekday by hour-of-day (7x24). This selection was made as it is primary resolution for law enforcement, to schedule work and analyze burglaries. The other resolutions are left for future work.

4.1 Data

The different datasets used for this thesis are presented here, divided into one section for each method that needs a dataset. The datasets are given a name so they can be referenced in later parts of the thesis and examples of the data are given.

4.1.1 Data for RQ1

This thesis will use two different sets of data. The first set consists of time recordings of all Swedish residential burglaries that took place in 2014. This dataset is referenced as **dataset 1**. The data is provided by law enforcement agencies and

include timestamps from when the crime may have been committed. The motivation for using this dataset is that it contains events of low temporal resolution, represented as a time interval.

To better research multiprocessing for the Aoristic method, dataset 1 was extended to contain 10x as much data. Sampling with replacement[1] was used to extend dataset 1 and keep all the values, in the creation process, independent from each other. The new dataset is ten times bigger than dataset 1 and is references as **dataset $1_{\textbf{extended}}$**.

Data representation for RQ1

The data in dataset 1 is represented by timestamps. Each crime has four attributes of interest as shown in table 4.1. date_{start} is the first day a crime may have occurred. date_{end} is the last day a crime may have occurred. This applies to time_{start} and time_{end} as well. They represent the first and last timestamp a crime may have occurred.

	Type	Format	Comment
date_{start}	Date	yyyy-mm-dd	The earliest date for a crime
date_{end}	Date	yyyy-mm-dd	The latest date for a crime
time_{start}	Time	hh:mm:ss	The earliest time for a crime
time_{end}	Time	hh:mm:ss	The latest date for a crime

Table 4.1: Representation of aoristic data.

4.1.2 Data for RQ2

Table 4.2 shows the input data for the evaluation of RQ2. The input data is one-dimensional lists with binary values, simulating hotspots and coldspots. The value 1 equals presence of a hotspot, -1 equals presence of a coldspot and 0 equals absence of a hotspot and coldspot. Input A and input B is representing the hotspot maps to be compared. The four tests are representing identical maps, dissimilar maps, partially identical maps and hotspot maps with both hotspots and coldspots. The reason these tests are chosen is to test four possible hotspot maps. With the authentic data, it is hard to produce all kinds of potential outcomes.

[1] https://www.ma.utexas.edu/users/parker/sampling/repl.htm

	input A	input B
No similarity	[1,0,1,0,0,1,0,0,0,0]	[0,0,0,0,0,0,1,0,0,1]
Identical	[1,0,1,0,0,1,0,0,0,0]	[1,0,1,0,0,1,0,0,0,0]
Partial	[0,0,1,0,0,1,0,0,1,0]	[1,0,1,0,0,1,1,0,1,0]
Coldspots/hotspots	[-1,-1,1,0,1,1,0,1,0,1]	[0,-1,1,0,0,1,0,0,1,1]

Table 4.2: Table showing the input data for similarity tests

4.1.3 Data for RQ3

In experiment 2, dataset 1 with the addition of the burglaries that took place in the months October, November and December in 2013, will be used. This data is reference to as **dataset 2**. The three months are added so prediction for an entire year can be tested.

The second dataset used consists of an access log from a web server for the duration 10/2017-12/2018, provided by staff at BTH. This dataset is referenced to as **dataset 3**. The motivation for using this dataset is that it contains fixed timestamps, i.e. precise temporal precision, to showcase that the prototype can be used for other types of data than crime data.

Data representation for Dataset 3

The data in dataset 3 is represented by a single timestamp. One example of a timestamp from a log is: *12/Aug/2017:13:05:34 +0200*. It is a log file from a web server where all requests made are saved in a log by the web server itself. The format of the logged requests follows *CLF* (Common Log Format)[2]. Each log has a precise temporal precision and the date and time is being parsed from each line in the log.

The log server differs from the burglary data in the way that a server is accessed every hour of the day from all over the world, both by people and other servers, this leads to a lot of data at all hours of the day. Burglaries are not as frequent and are local, they are done in Sweden, not from across the world and most burglaries span multiple hours. Burglary data is more likely to contain clusters as burglaries does not happen as often and maybe not all the time. While the server data is more likely to be more evenly spread out over the time of day and making it harder to find hotspots.

[2]https://httpd.apache.org/docs/2.4/logs.html

4.2 Experiment 1

Experiment 1 will be used to find an answer to RQ1. Getis-Ord* and Aoristic will be implemented with both multiprocessing and serial execution. The Aoristic method has two different multiprocessing implementations, one with shared memory and one without. The implementations are described in section 3.2. During the implementation of shared memory, for Aoristic, it was discovered that it produced faulty results. Each execution, with the same data, resulted in different values. Execution with the same data should always have the same result. Furthermore Python does not support multiprocessing with shared data on the Windows operating system and is therefore not a relevant solution. For these reasons multiprocessing with shared memory was removed from the study. In conclusion, experiment 1 for the Aoristic method will only use the sequential and multiprocessing without shared memory. The multiprocessing implementation without shared memory will from now on be referenced as the multiprocessing implementation.

Performance testing will be done on each implementation to see whether differences exists. Performance will be evaluated by measuring execution time. The experiment has two levels, Getis-Ord* and Aoristic, the independent variables are the sequential and the multiprocessing implementations and the dependent variable is execution time. Each implementation of the methods will be executed 10 times and an average will be calculated. The Aoristic method will be run in two iterations, to see how the amount of data affect each implementation, based on the data described in section 4.1. Once with dataset 1 and once with dataset 1_{extended}. Amount of data have a very small impact on the performance of Getis-Ord*. The number of calculations and iterations the code does in Getis-Ord* depends on the size of the matrix, the resolution, the bigger the matrix the more code needs to be iterated. Therefore Getis-Ord* will only be tested with dataset 1 as the data inside the matrix only affects arithmetic operations which in turn have a very small impact on execution time. Getis-Ord* will always be performed on a matrix of 7x24 resolution because the Aoristic method provides a matrix in the preferable resolution. The result of experiment 1 will show if the methods benefit from multiprocessing. The most eligible implementation will be applied to the prototype. For the Aoristic method execution time is counted from start of the calculation until one final matrix is produced.

The outcome of experiment 1 will be displayed as mean and standard deviation from the execution time of each implementation. The result from t-test and *Cohen's d* will also be presented.

4.2.1 Evaluation

Experiment 1 will be evaluated with t-test, also known as Student's t-test. T-test is a parametric hypothesis test used to determine if two datasets are significantly different from each other and to what degree [30]. Independent samples t-test was chosen over paired samples t-test because each data set is independent but identically distributed. An alpha value of 0.05 will be used for the tests.

T-test will be used to derive if there is a significant difference between the execution time of each implementation. The test will be used for both Getis-Ord* and Aoristic. If a statistical difference between the implementations is determined a Cohen's d test will be performed to identify the effect size of the difference.

Cohen's d is a standardized measure of effect size. Based on two means and standard deviations Cohen's d calculates a *d index*. The calculated d index can be compared to a table, derived by Cohen, to determine the magnitude of the effect size [30]. The three steps are small 0.2 - 0.5, medium 0.5 - 0.8, big > 0.8.

Effect size	*d*
Small	0.2 - 0.50
Medium	0.50 - 0.80
Big	>0.80

Table 4.3: Cohen's three categories for effect size.

4.2.2 Tools and measurements

Execution time is to be measured in the highest resolution possible by the Python time module[3]. The time will then be rounded to milliseconds with three decimals for Getis-Ord* and seconds with three decimals for the Aoristic method.

The confidence interval is calculated with the help from the Python package *SciPy* [4] and the method *interval* in the module *stats.norm*[5]. The variance is calculated with the *numpy.var* function[6].

[3]https://docs.python.org/3/library/time.html
[4]https://www.scipy.org/
[5]https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html
[6]https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.var.html

4.3 Case study

RQ2 will be answered with a program implementation case study. The case study will be conducted by implementing the four similarity measures, described in section 2.4.1 and evaluating them by using different lists of test data, described in table 4.2. The lists for the evaluation will have the size of 10 for ease of evaluation and the result will be multiplied by 100 to get a percentage value and rounded to one decimal for readability. The result will be presented in a table, showing the calculated score from each test. The reason for using fabricated lists is that it is hard to cover all test cases and evaluate them with authentic data. As a complement to the tests, the most accurate measure will be implemented into the prototype and tested with authentic data. That data will be for the cities Gothenburg and Karlskrona from 2014 and can be found in tables E.1 and E.2 in Appendix E.

The measure that provide the most accurate result will be implemented into the prototype. Two calculated hotspot maps will be selected and for each map, the coordinates that make up a hotspot will be marked with the value 1, coldspots will be marked with -1 and absence of hotspots and coldspots with the value 0, meaning the matrices used for the comparison will consist of numeric values. The two matrices will be the input data to the selected measure and the prototype will display both the similarity score as well as the dissimilarity score. The prototype will provide the most suitable measure for hotspots, coldspots as well as both included.

4.3.1 Evaluation

The result from the case study will be evaluated by comparing each similarity measure's result to an expected score. The lists for the test will be in the size of 10 for ease of evaluation. The result will be multiplied by 100 to get a percentage value and rounded to one decimal for readability. The result will be presented in a table showing the calculated score from each test.

4.3.2 Presentation of similarity

The hotspot and coldspot comparison functionality will also be implemented into the prototype. The overlap will be calculated and visualized as a new heatmap, only displaying the overlap. The compared maps will be traversed and for each coordinate where both maps consist of a hotspot or coldspot, the coordinate will be marked with a calculation of an increase or decrease of the z-score in percent. As a complement to overlap the delta will be calculated for the two compared hotspot maps and presented in a percentage increase or decrease in the amount of crimes detected and the amount of hotspots detected.

4.4 Experiment 2

RQ3 will also be answered using an experiment. The Naive forecasting method, explained in section 2.5, will be tested with three different time windows, 1, 2 and 3. The numbers represent how many previous months hotspot maps will be used to predict the next months hotspot map. The time windows are limited to three months because research suggest that recent data should to be used when forecasting crime, for example data that is one year should not be used [29, 36]. The experiment will be done with three use cases, the first is the full dataset of dataset 2, i.e. data for Sweden. The second use case is a subset of dataset 2 where only data from the city of Malmö in Sweden and the last use case is done with dataset 3, the server log. Each dataset will be divided into twelve chronological subsets for each time window.

Here is how each dataset will be split into 12 subsets explained using dataset 2. For time window 1 will 2013/12 (training data) used to forecast 2014/01 (test data), 2014/01 to predict 2014/02 and so on until all months in 2014 have been predicted. For time window 2 will two months be used to predict the next, i.e. 2013/11-12 to predict 2014/01, 2013/12-2014/01 to predict 2014/02 and so on. For time window 3 will three months be used as training data to predict the test data. Overall there are 36 subsets to evaluate the method and each subset consists of training data and test data. Time window 3 and its subsets are illustrated in figure 4.1.

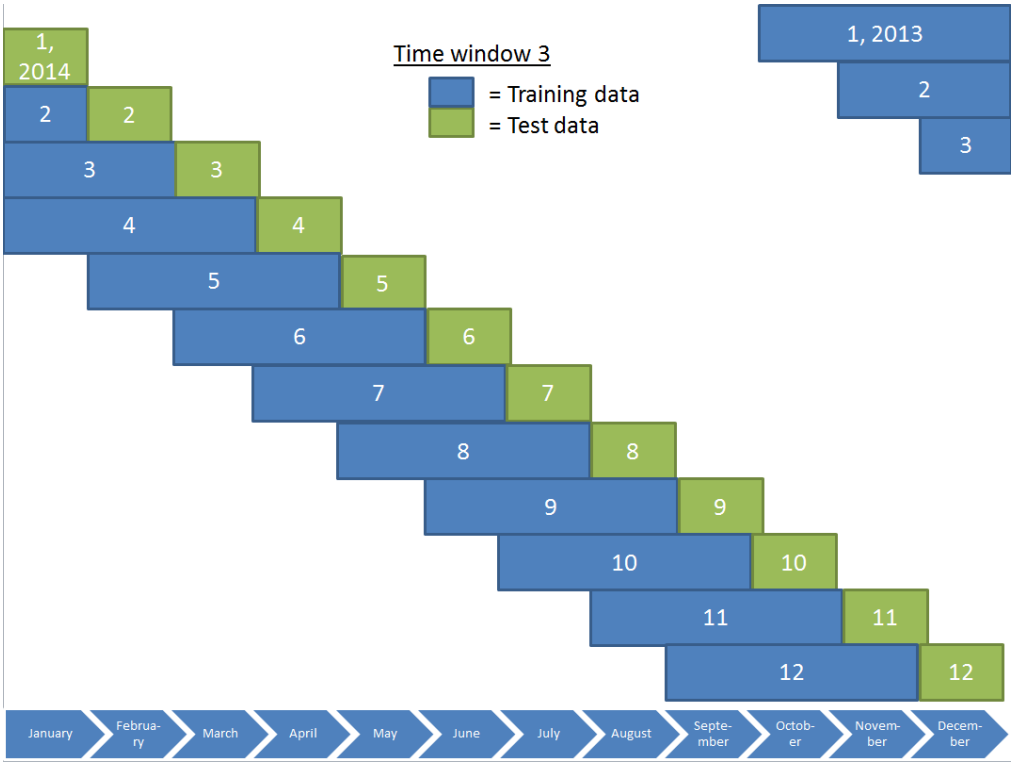


Figure 4.1: Illustration of the 12 subsets in time window 3.

Time window 1 is technically not utilizing the Naive forecasting method, it is just using the previous month's hotspot map as next month's. It is however a common technique used to forecast future hotspot maps, known as hotspot random walk [36]. Part of experiment 2 will evaluate how well time window 1, or random walk, performs for temporal hotspots and how well the Naive forecast method with time window 2 and 3 performs and compares to time window 1.

4.4.1 Use case subset Malmö

When using data for all of Sweden, dataset 2, geographical differences can disappear in the mass. Therefore the use case of the city of Malmö was added. To evaluate how the method performs on data from a small geographical area, the city Malmö in Sweden, where there possibly could exist a pattern for when burglaries happen. For this use case only data for the city of Malmö, extracted from dataset 2, will be used in the same setup as Experiment 2. The intention of this is to remove the impact of geographical differences i.e. burglaries follow a different pattern in the north of Sweden than the south, west and east. Furthermore if the police were to use the Naive forecast method they would use it on a small geographical area, a district or precinct. This use case better emulates a real use case for the method.

4.4.2 Use case Server log

To showcase how the prototype can also be used for other data than burglary data, this use case has been added. The Naive forecasting method will be used with dataset 3. This use case will display the possibilities of this prototype with other sorts of data as it includes both creating hotspot maps and comparing hotspot maps. The use case will have the same setup as the previous use cases for experiment 2. If the Naive forecasting method works with server logs and hotspots can be predicted, the developers of a server can predict when not to update the server or when the server needs to be ready to handle extra traffic.

4.4.3 Evaluation

Coldspots will not be part of the evaluation, before each evaluation method the coldspots will be removed so only the prediction of hotspots are evaluated. This is because the purpose of one of the method used for evaluation, PAI which is explained below, is to calculate how valuable hotspots are. Because coldspots contain low crime values if they are part of the evaluation it will lower PAI scores for hotspots. Experiment 2 will be evaluated by measuring the overlap between a hotspot map created with the forecast method and the hotspot map created on the test data, in other words if the forecasted hotspots are in the same location as the real. This is referred to as accuracy of a prediction. The accuracy will be measured using Jaccard, as a result from RQ2. Jaccard is explained in section 2.4.1

Additionally to evaluate how valuable the predictions are, precision of the forecasts will be measured. Precision will be evaluated with PAI. PAI is explained in the section below 4.4.3. PAI was selected as the evaluation method over Hit rate. Hit rate was discarded because it does not take into account the area of hotspot [32]. Which means that useless hotspots might get good accuracy scores.

Another common evaluation metric used when evaluating hotspot techniques is Recapture Rate Index (RRI). RRI is used to quantify how well a hotspot map predicts increase or decrease of number of crimes that happen [18]. The intention of the Naive forecasting method used in this thesis is not to predict the number of crimes but to predict when crime will happen. Therefore RRI is not relevant to use as an evaluation method.

Prediction Accuracy Index

To evaluate the accuracy of a hotspot map's ability to predict where crime happen Chainey et al. [32] developed the Predictive Accuracy Index (PAI). Unlike the methods used before PAI was invented, PAI takes into calculation the size

of the areas of the hotspots where crimes are predicted. E.g. a hotspot over the entire study area is useless for a policeman who plans resources. The point of a hotspot map is to find clusters of crime and if the entire map is a hotspot then there are no hotspots. Additionally PAI can handle multiple hotspots in the study area which some methods can not.

$$\frac{\frac{n}{N} * 100}{\frac{a}{A} * 100} \tag{4.1}$$

The equation for PAI can be seen in equation 4.1. n is the number of crimes that happened in the hotspot area. N is the total number of crimes that happened in the study area, the entire hotspot map. a is the total size of all the hotspots located in the study area. A is the total size of the study area. The smaller the hotspot area is compared to the study area, the hotspot map, and the greater the number of crimes in it achieves a higher PAI value.

A PAI value of 2 or higher is considered good, it is achieved when 80 percent of the total crimes in the study ares is found in hotspots which make up 40 percent of the study area. A value of 1 is achieved when the hotspots make up 100% of the study area, which means that 100% of the crimes are in the hotspots. This value however is bad because a hotspot that makes up the entire study is not a hotspot as hotspots are supposed to represent clusters in the study area.

The opinion that PAI is a misleading name, as accuracy refers to size and shape of a hotspot, have been raised [33]. Rather, a more appropriate word is precision as precision refers to the point capture ability. Therefore PAI was renamed to Forecast Prediction Index (FPI). Furthermore it was suggested that to measure a hotspot map's ability to predict, a measurement of accuracy is necessary. Accuracy refers to number of crimes in hotspots between the training and testing datasets. However their definition of accuracy is not relevant for this thesis. Here accuracy will refer to the location of the hotspots, if hotspots are forecasted in the same cells they are located in the test data. Furthermore, this thesis will continue to refer to PAI as PAI as it is the more commonly used name. If a new name were to be used for PAI an even better name would be Hotspot Precision Index. PAI is used to measure the value of hotspots, how well hotspots correlated to the actual data, not necessarily how correct the hotspots are in the future. Furthermore the names Prediction Accuracy Index and Forecast Precision Index assumes that hotspots are used to predict the future while they just as well could just be used to visualize data. Therefore the authors of this thesis believe that Hotspot Precision Index (HPI) would be a more appropriate name.

The implementation of PAI can be found in Appendix C.

4.5 Theoretical analysis

A theoretical analysis will be performed to answer RQ4. Relevant sources that are available about software packaging methods will be used to find appropriate and relevant methods and then to find out how the methods work in regards to the defined aspects below. Relevant sources includes scientific literature and technical documentation about the methods. When choosing the methods to analyze, the aim is to have the most popular methods for each of the top three most popular operating systems. The most popular operationg systems (OS) is Windows, the second most is Linux and the third most popular is Mac OS [7][8].

4.5.1 Aspects

Packaging the software is one of the last processes when developing an application. Which method the developer selects can both have an impact on the developer and the end user of the application. The developer wants the work with packaging the software to be as easy and smooth as possible so they can release it to their users. For the users, the packaging method is the entry point to the application, it decides how they install and run it. Which aspects to compare for each packaging method were produced when the authors tried to find areas that are affected by the packaging method. The areas can affect both the developer and the end user of the application.

Which operating systems (OS) the packaging method is supported on impacts the user base the developer can target with the application. This can have a big impact on the size of the user base and the type of user base the developer expect the application to have. Furthermore it can also influence if extra work is needed to support an OS. How the packaging method handles dependency concerns both the developer and the user. For the developer it is interesting to see if the dependencies can be part of the package and used from inside or if the dependencies need to be external and connected to the application somehow. If the dependencies are not part of the package that means they not only need to interact with the package, but also the external dependencies, i.e. complicating the process of using the application. If the packaging method have a built-in distribution system it should alleviate work from the developer and ease the experience for the end user to retrieve the application. With a built-in distribution system the developer does not need their own hosting and distribution network which means they can save time and resources.

How different processes work is also of relevance, for the developer they are mainly how it works to setup and create a package for the first time and then how

[7] http://gs.statcounter.com/os-market-share/desktop/worldwide
[8] https://w3techs.com/technologies/overview/operating_system/all

packages can be updated when there is a new version of the software. Software is continuously updated which means that the package also needs to be updated continuously. This process should be as minimal as possible for the developer to assure a smooth updating process. Two processes that have impact on the user is the installation and updating process. The user wants an easy and problem free experience when installing a package. Complex and tiresome operations can lead to users not knowing how to finish the installation or not bothering to finish. Both the user and the developer wants the user to use the latest version of an application as it has the latest features, bug and security fixes. To assure that the user have the latest version the process to update an application need to be minimal otherwise the user will not update the application.

The following aspects have been selected and will be used for evaluating the different methods:

A1: Platforms

Platforms examines which operating systems the package is supported on. Which OS's a method is supported on is a vital part of the process for selecting how to package the software. As platform compatibility can limit the user base to specific OSs which is bad if the developer wants a user base as large as possible. This limitation can also be sought after, if the developers only want to maintain support for the software on one specific OS or if the developers know that the users they are targeting use a specific OS.

This aspect has no complexity. The points given will relate in a 1-to-1 mapping to the number of operating systems that has support for the created package.

A2: Creation of package

This complexity aspect means to explore how complex the process of packaging an application for the first time, what the needed steps are and what additional programs and libraries are required, if any.

A baseline for low complexity would be to run a single command. The creation process of a package has a complexity to it. The process may involve thresholds if the package is created for the first time such as requirement of external applications or libraries, difficult settings or the process may be time consuming. The complexity will be judged based on baseline and in relation to each other. The complexity will be defined as high, medium or low.

A3: Updating a package as a developer

What is the process for updating or recreating the package when the code have been updated? Does the developer need to redo the entire create process or is the update process simplified in some manner? A developer does not want to spend unnecessary time to recreating processes over and over. Computers should help automate tasks and having to redo the entire create process for updating the package with new code is undesirable compared to being able to rerun something that was created during the installation process.

A baseline for low complexity would be to reuse the artifact that was created in the creation process. The process of updating the package as a developer has a complexity to it. The complexity may involve external libraries or a time consuming process to upload the updated package. The complexity will be judged based on the baseline and in relation to each other. The complexity will be defined as high, medium or low.

A4: Installation of package

Here the type of installation process will be investigated. If it is a simple installation process, only click next or if advanced settings or steps are to be done by the user. This is relevant because novice users with little computer knowledge can have problems or be intimidated by the installation process and in turn can cause them to not installing it at all.

A baseline for low complexity would be to run a single command or execute a file. The process of installing the package as a user has a complexity to it. The complexity may involve the need for external libraries and programs or the process may be hard for a general user. The complexity will be judged based on the baseline and in relation to each other, the complexity will be defined as high, medium or low.

A5: Updating a package as user

If there is an easy built-in mechanic in the package to update the program automatically or if the user have to manually download a new file to update the software is explored in "Updating a package as user". As development of a software continues and new versions are release, the new versions should have fixed bugs, security holes and added new features compared to the original version. Because of this it is in both the user and the developers interest for the newest version to be used. The less manual work needed to update the program the more users will update them, therefore it is helpful if the process for updating is as minimal as possible.

The process of updating the package as a user has a complexity to it. The complexity in this case refers to if the package updates itself automatically, by a single command or execution or has a need for long commands in a terminal and so on. The complexity will be defined as high, medium and low. Where low is automatic updating, medium is one click or command and complex is other.

A6: Dependencies

A program or application most often utilize other libraries or modules to work in its full extent. This aspect means to explore if the package method handles the applications dependencies automatically or if the end user will have to install the necessary libraries or modules themselves. This aspect is relevant and chosen because it demands a lot more from the end user to install additional dependencies instead of installing just one, the software. If an extra program is needed to execute the package itself will also be included here.

The aspect of dependencies has a complexity connected to it. The complexity lies in the area of included dependencies. If the package contains all dependencies, no dependencies or if the process includes some interaction from the user or not. The points given for this aspect maps to the levels described if the dependences may be included, where an all inclusion is worth 3 points. The levels are defined as included, semi-included and not included.

A7: Distribution

Is the packaging method supported with a distribution system to help the developers host the package and distribute to the users? A distribution system alleviates the developer's need for hosting their own server, in other words less expenses. This is important as not all developers have the resources necessary to setup their own server for hosting and distribution.

The points given will reflect the availability and ease of use of a distribution system for a normal user and is defined as not available, available and easier to use system.

4.5.2 Evaluation

To quantify how complex a packaging method is, the authors will assign a number to each aspect for each method, representing the complexity in relation to each other. The reason behind that is because it is hard to put a number on aspects like the creation of a package, described in section 4.5.1. Additionally, there is no standard procedure on creating a package. The authors are therefore comparing the methods and their procedures to get an estimation of how complex they are.

Points	A1	A2	A3	A4	A5	A6	A7
1	One OS	high	high	high	high	not included	not available
2	Two OS	medium	medium	medium	medium	semi-included	available
3	Three OS	low	low	low	low	included	easier

Table 4.4: Points for evaluating aspects

The resulting table will consist of a summation of the numbers each method's aspects get, as will be seen in their intersection. The complexity points are presented in table 4.4.

Chapter 5

Results

In this section one can find the result for all the methods used to answer the research questions described in section 1.3. The results are ordered in the same order as they were written about in the method chapter, firstly Experiment 1, secondly the case study, thirdly Experiment 3 and lastly the theoretical analysis.

5.1 Experiment 1

The result for experiment 1 is divided into two parts, first the result for Getis-Ord* is presented then the result for the Aoristic method. The result for each method is kept and discussed separately.

5.1.1 Getis-Ord*

Type	n	Mean	Standard Deviation
Sequential	10	2.266	0.057
Parallel	10	114.479	1.636

Table 5.1: Average execution time and standard deviation of sequential and parallel. Times are displayed in milliseconds.

There was a significant difference in the scores for sequential (M=2.266, SD=0,057) and parallel (M=114.479, SD=1,636) conditions; t (18)=-217, p=4,816E-18, d=-94.941. The resulting p-value is far less than 0.0001 and by that, clearly statistical significant.

The effect size for this analysis (d=-94.941) was found to exceed the convention for a large effect, table 4.3. These results suggest that multiprocessing has a significant impact, with a large effect size, on execution time. Specifically, our results suggest that when implemented with multiprocessing the execution time is increased.

5.1.2 Aoristic method

The presentation of the result for the Aoristic method is split into two sections, one for each dataset tested. Because of the partitioning in the parallel implementation mentioned in section 3.2.3, is not only the execution time for the entire implementation but also the execution time for only the Parallel calculation presented. The Parallel calculation is part two of the multiprocessing implementation, explained in section 3.2.3.

Dataset 1

Implementation	n	Mean	Standard Deviation
Sequential	10	0.605	0.018
Parallel	10	0.834	0.069
Parallel calculation	10	0.369	0.019

Table 5.2: Average execution time and standard deviation of Sequential, Parallel and Parallel calculation, with dataset 1. Times are displayed in seconds.

There was a significant difference in the scores for sequential (M=0.605, SD=0.018) and multiprocess (M=0.834, SD=0.069), conditions; t(18)=-10.169, p =1.361E-06, d=4.548. The effect size for this analysis (d = 4.548) was found to exceed the convention for a large effect, table 4.3. These results suggest that multiprocessing has a significant impact, with a large effect size, on execution time. Specifically, our results suggest that when implemented with multiprocessing the execution time is increased.

As seen in table 5.2 Parallel calculation, which does the actual Aoristic calculations, is faster than the Sequential implementation.

Dataset 1_{extended}

Implementation	n	Mean	Standard Deviation
Sequential	10	6.319	0.134
Parallel	10	9.489	0.382
Parallel calculation	10	3.130	0.078

Table 5.3: Average execution time and standard deviation of Sequential, Parallel and Parallel calculation, with dataset 1_{extended}. Times are displayed in seconds.

There was a significant difference in the scores for sequential (M=6.319, SD=0.134) and multiprocess (M=9.489, SD=0.382), conditions; t(18)=-24.729, p = 5.424E-11, d=11.650. The effect size for this analysis (d = 11.650) was found to exceed the convention for a large effect, table 4.3. These results suggest that multiprocessing has a significant impact, with a large effect size, on execution time. Specifically, our results suggest that when implemented with multiprocessing the execution time is increased.

As seen in table 5.3 Parallel calculation, which does the actual Aoristic calculations, is faster than the Sequential implementation.

5.2 Case study

The case study tested the four similarity measures with the data presented in table 4.2 to see how they resulted compared to the expected result. The result from all 16 tests of similarity measures are shown in table 5.4. Note that the type called HC is the test for both hotspots and coldspots and the resulting score is multiplied by 100 and rounded to one decimal, to get a clear view of the percentage. The column *Expected %* is showing the expected result from the tests. The tests for no similarity has no matches at all and is expected to result to 0.0%. The tests for *identical* is expected to result in 100.0 % because they are identical. The *partial* tests are expected to result in 60.0% because there are 3 out of 5 positive matches, which is equal to 60%. The tests for HC should result in 50% because considering both coldspots (-1) and hotspots (1), there are 4 out of 8 positive matches.

Test	Type	Method	Result (%)	Expected (%)
1	No similarity	Jaccard:	0.0	0.0
2	No similarity	Sørensen-Dice:	0.0	0.0
3	No similarity	Kulzcynski:	0.0	0.0
4	No similarity	Ochai:	0.0	0.0
5	Identical	Jaccard:	100.0	100.0
6	Identical	Sørensen-Dice:	100.0	100.0
7	Identical	Kulzcynski:	100.0	100.0
8	Identical	Ochai:	100.0	100.0
9	Partial	Jaccard:	60.0	60.0
10	Partial	Sørensen-Dice:	75.0	60.0
11	Partial	Kulzcynski:	80.0	60.0
12	Partial	Ochai:	77.5	60.0
13	HC	Jaccard:	50.0	50.0
14	HC	Sørensen-Dice:	66.6	50.0
15	HC	Kulzcynski:	68.6	50.0
16	HC	Ochai:	67.6	50.0

Table 5.4: Result for similarity tests.

The result shows that the Jaccard is the only measure that resulted in a correct score on all tests. All measures had a correct result on tests 1-8, the tests for no similarity and the test on identical lists.

Figure 5.1 displays how the hotspot maps to be compared is visualized in the prototype. The input data is the authentic crime data from Gothenburg and Karlskrona. Figure 5.2 displays how the overlap is visualized in the prototype. From the result in Figure 5.1, the similarity is calculated and presented in figure 5.5 and figure 5.6 displays the calculated delta.

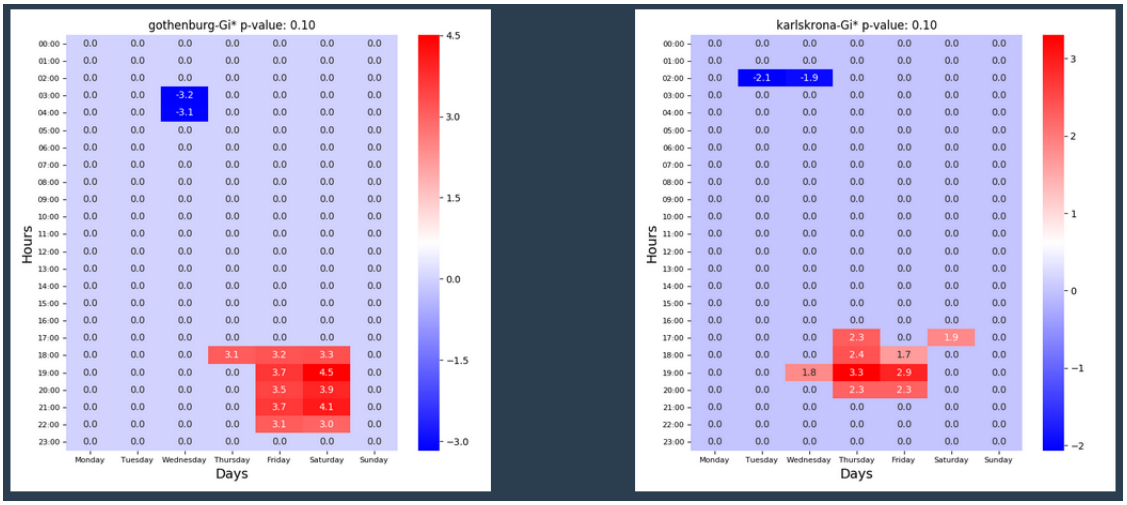


Figure 5.1: Screenshot of the compared hotspot maps, Gothenburg and Karlskrona from 2014, showing hotspots and coldspots.

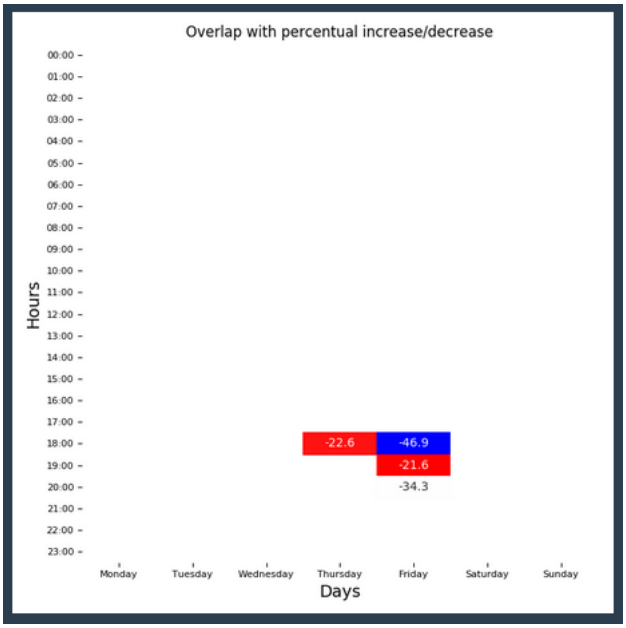


Figure 5.2: Screenshot of the resulting hotspot map, showing overlap and percentual change in z-score

Type	Similarity	Unique
All	20.0%	80.0%
Hotspots	25.0%	75.0%
Coldspots	0.0%	100.0%

Table 5.5: The resulting Jaccard similarity calculation.

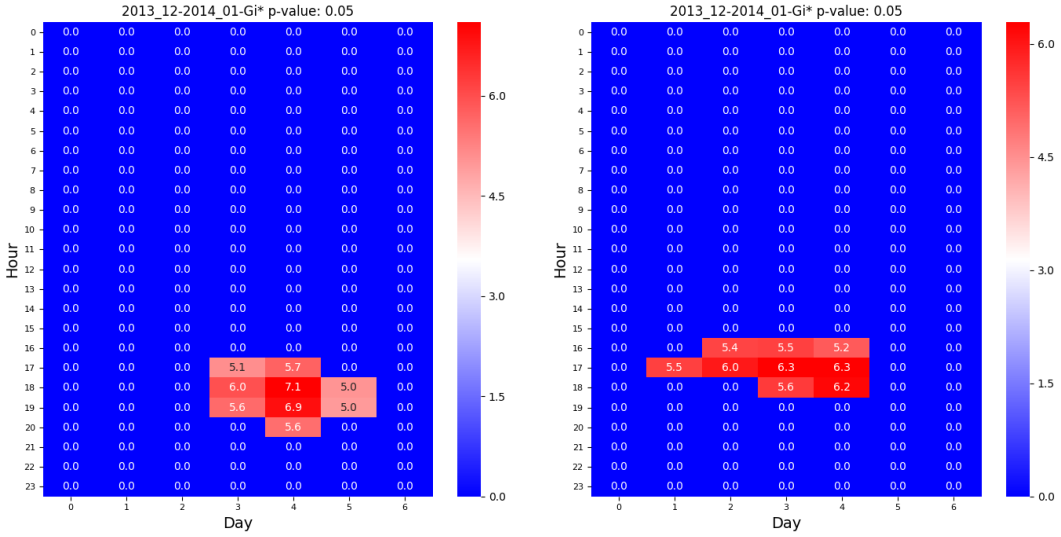
Type	Change +/-
Amount of hotspots/coldspots	-15.38%
Amount of occurrences	-94.27%

Table 5.6: The resulting delta calculation.

5.3 Experiment 2

The result from each use case is presented individually. The raw data from experiment 2 can be found in Appendix A. In this section the result is presented as mean, standard deviation and median for the Jaccard and PAI values, based on the tables in Appendix A. There is one table for each time window the experiment was run with. Each table contains the Jaccard and PAI value for each month predicted.

5.3.1 The full dataset 2



(a) Test data, 02/2014.

(b) Training data, 12/2013 + 01/2014.

Figure 5.3: Hotspot maps created on test and training data from the full dataset using time window 2.

An example of hotspot maps created for this use case can be seen in figure 5.3. The training hotspot was created using the Naive forecasting method with time window 2. The Jaccard value for these two is 28.6% and the PAI score is 1.798.

The results for time window 1 exhibit an unreliable method for forecasting temporal hotspot maps. Analyzing the overlap between the predicted hotspots and the actual hotspots, table 5.7, displays varied and poor performance. A low mean and median and quite high standard deviation. Furthermore, in the raw data, table A.1 it shows that 7 out of 12 months produced a Jaccard value of 0, i.e. no hotspots overlapped. At best there was an overlap of 54% achieved, for February. Similar results, with a slight improvement, are found for time window 2. The median rose to 4.6 as only 5 out of 12 months got a score of 0, table A.2. Interesting to note that January went from 0% to 60%. Time window 3 had the lowest overlap but still very similar to time window 1 and 2. Time window 3 also had 7 months with Jaccard 0, table A.3.

Time window	n	Mean	Standard deviation	Median
1	12	12.1	18.4	0
2	12	12.3	18.1	4.6
3	12	10.4	16.1	0

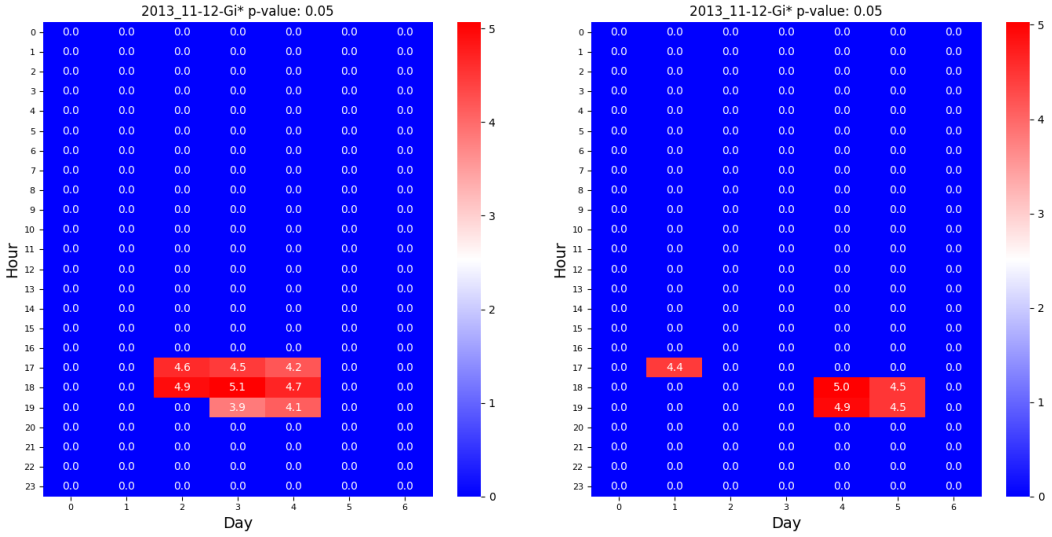
Table 5.7: Jaccard values from experiment 2, Sweden. Mean, SD, and median values are displayed in percentage. The data is based on tables in Appendix A, section A.1.

Likewise the PAI value for the three time windows are similar to each other with small variations, table 5.8. The values hover around 1.5 meaning that crime happens where the predicted hotspots are but no major crime activities takes place at those times. In the tables in Appendix A, section A.1, it can been seen that the lowest PAI values connect to the 0 Jaccard values and mostly higher Jaccard values leads to higher PAI value. There are some anomalies, e.g. February and November in table A.1. February has a much higher Jaccard value than November, but November has a Higher PAI value. In all three tables in section A.1 the lowest values are located during the summer months. The Jaccard values are mostly 0 and the PAI values are the lowest. Similarly most non-zero values are located during the winter months and have the highest PAI values.

Time window	n	Mean	Standard deviation	Median
1	12	1.531	0.337	1.545
2	12	1.586	0.349	1.518
3	12	1.415	0.649	1.402

Table 5.8: PAI values from experiment 2, Sweden. The data is based on tables in Appendix A, section A.1.

5.3.2 Subset Malmö



(a) Test data, 01/2014.

(b) Training data, 11/2013 + 12/2013.

Figure 5.4: Hotspot maps created on test and training data from subset Malmö using time window 2.

An example of hotspot maps created for this use case can be seen in figure 5.4. The training hotspot was created using the Naive forecasting method with time window 2. The Jaccard value for these two is 18.1% and the PAI score is 2.603.

The Jaccard values produced in the experiment, table 5.9, display poor overlap and varied results from month to month, which can further be demonstrated in the raw data, the tables in section A.2. Time window 1 contains six months where Jaccard is 0, time window 2 four months and time window 3 five months. Which time window achieved best result for Malmö is ambiguous. Time window 3 is the inferior time window while time window 1 and 2 achieve differently. Time window 1 achieve higher values but lower number of overlaps compared to time window 2. Which means time window 2 is more stable than time window 1. For Malmö the Jaccard scores are more spread out between summer and winter than for the full dataset results where the overlap is mostly located around the winter months.

Time window	n	Mean	Standard deviation	Median
1	12	9.3	14.1	5.5
2	12	6.8	12.1	5.5
3	12	4.3	6.8	2.75

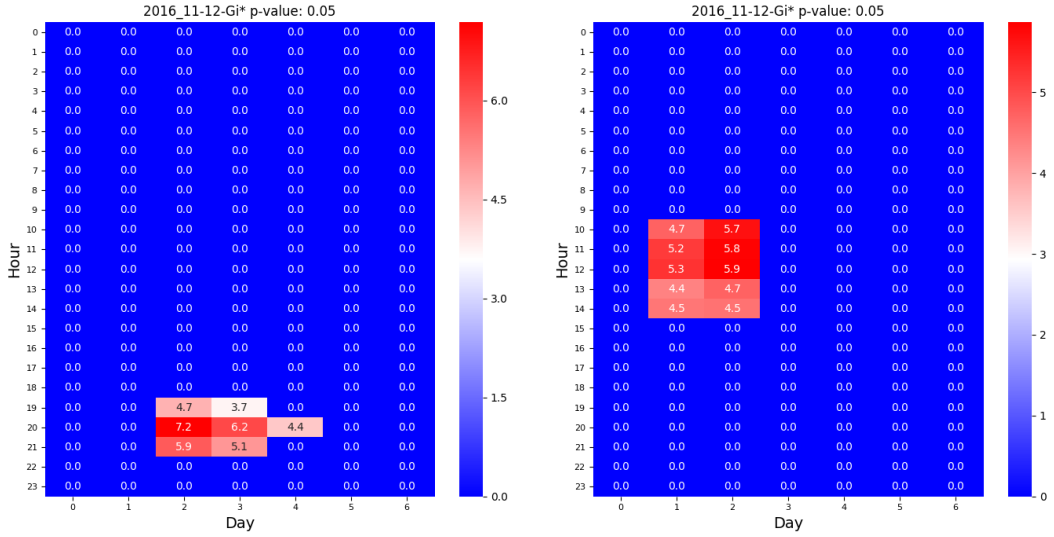
Table 5.9: Jaccard values from experiment 2, Malmö. Mean, SD, and median values are displayed in percentage. The data is based on tables in Appendix A, sectionA.2.

The PAI scores for time window 1 is highest and varies the least compared to time window 2 and 3, table 5.10, although the values do not distinguish much between the different time windows. The PAI average and median are quite high, but the standard deviation values are high, which is bad. Thus is the precision of the Naive forecast method not very stable. It is notable that some high PAI scores were acquired, multiple values above two and two months had over three, section A.2.

Time window	n	Mean	Standard deviation	Median
1	12	1.899	0.522	1.945
2	12	1.852	0.627	1.810
3	12	1.806	0.576	1.811

Table 5.10: PAI values from experiment 2, Malmö. The data is based on tables in Appendix A, section A.2.

5.3.3 Server log



(a) Test data, 01/2018.

(b) Training data, 11/2017 + 12/2017.

Figure 5.5: Hotspot maps created on test and training data from dataset 3 using time window 2.

An example of hotspot maps created for this use case can be seen in figure 5.5. The training hotspot was created using the Naive forecasting method with time window 2. The Jaccard value for these two is 0% and the PAI score is 1.409.

The Naive forecasting method, with dataset 3, produced very poor results, both Jaccard and PAI. As seen in table 5.11, time window 3 had the highest mean overlap with only 7.5% and a median of 0%, same as the other time windows. Both time window 1 and 2 had 10 months where no overlap was found, table A.7 and A.8. Furthermore as seen in table A.9, merely 7 months got 0% as mean overlap.

Time window	n	Mean	Standard deviation	Median
1	12	2.1	5.8	0
2	12	2.5	6.5	0
3	12	7.5	11.9	0

Table 5.11: Jaccard values from experiment 2, Server log. Mean, SD, and median values are displayed in percentage. The data is based on tables in Appendix A, sectionA.3.

The PAI values are also low across the board with time window 2 having minimally better values for mean and median, table 5.12. The PAI values suggest very low accuracy as a majority of the months got below 1.0 as value and the lowest one being 0.243, table A.8. Each time window had one month where no hotspots were found in the predicted hotspot map, exposed by a 0 as PAI value.

Time window	n	Mean	Standard deviation	Median
1	12	0.756	0.488	0.968
2	12	0.866	0.495	0.993
3	12	0.881	0.585	0.789

Table 5.12: PAI values from experiment 2, Server log. The data is based on tables in Appendix A, section A.3.

5.4 Theoretical analysis

This section contains the result from the theoretical analysis executed to answer RQ4. Firstly, the selected packaging methods are introduced and why they were chosen. Secondly, each package is explained in context of the aspects. Lastly, the result is summarized. A quick summary and the values for each package method's aspect can be seen below in table 5.13. In the table, higher scores are better, i.e. the package with the highest score is best in regards to the aspects.

	A1	A2	A3	A4	A5	A6	A7	Sum
Python package	3	2	3	3	2	2	3	18
Docker	3	1	3	2	2	3	3	17
EXE	1	2	3	3	1	2	1	13
APP	1	2	3	3	1	2	2	14

Table 5.13: Table summarizing findings for the theoretical analysis of RQ4

The methods, to package software, that this review will investigate are Python package, Docker, Executable (EXE) and APP. Below you can find what each method is and the reasoning for selecting the method. Both cross platform and operating system specific packaging methods are of interest. These choices were made mostly because of their popularity in different areas. Linux have several different distributions, Debian, Fedora, Red Hat and so on, and each has their own packaging method [1]. To avoid selecting one Linux distribution and in turn a packaging method, the authors chose to use Python package to cover Linux

[1] https://www.lifewire.com/guide-to-linux-packages-2202801

OS. Python package is the most popular for Python code and Python comes pre-installed on most Linux distributions [2].

Python package

The Python Package Authority, *PyPA* [3] provides recommendations on how packaging Python code should be conducted. They recommend to use the program *setuptools* [4] to package the code. The packaged code may then be uploaded to the community maintained repository Python Package Index *PyPI* [5]. PyPI hosts a great deal of Python software that can be installed with the PyPA recommended package manager *pip* [6].

Since the prototype is written in Python and due to the official recommendation from PyPA, the method Python package is chosen as one of the packaging methods.

Docker

Docker is the leading solution for virtual software containers, it can be used for application development, packaging and management. A Docker container allows for independence between an application and the infrastructure it runs on. A container image is a standalone executable package. The image contains the whole environment needed to run the application: code, runtime, system tools, system libraries and settings [7]. There are different editions of Docker available, this thesis will focus on the Community edition [8]. An application can be developed on one OS with specific dependencies and then be converted to a container image. The image, or software package, can then be distributed to users who only need to have Docker installed to start the container and use the application. Furthermore Docker offers the service Docker Hub [9]. Docker Hub is a public registry where developers can, among other things, publish their images. This allows the developers to easily distribute their software and it also simplifies the process for users to retrieve and update the software package [10].

Docker was selected as one of the packaging methods because it is a relatively new technology (launched in 2013), it is very popular and it offers encapsulation

[2] https://docs.python.org/3/using/unix.html
[3] https://www.pypa.io/en/latest/
[4] https://packaging.python.org/key_projects/#setuptools
[5] https://pypi.org/
[6] https://pypi.org/project/pip/
[7] https://www.docker.com/what-container
[8] https://docs.docker.com/install/
[9] https://hub.docker.com/
[10] https://docs.docker.com/docker-hub/

for the entire application environment [7]. Furthermore Docker has been suggested as a solution to the lack of reproducibility in computer science research [1, 4]. This adds to the reasons for looking into Docker.

Executable

Executable (EXE) is a file format for the Windows operating system that contains a program, or software. The software packaged inside the EXE file will be executed when the EXE file is opened [11]. By creating an EXE file for Python software the developer can target the Windows platform directly. EXE was chosen as a target packaging method as it is the popular choice for Windows applications.

To create EXE files of Python code a tool is necessary and there are multiple available. The most popular one is called py2exe [12]. The footnote also states that py2exe is only available for Python2 code but it since has been updated and now also work for Python3 code [13]. Therefor py2exe was chosen as the tool to create EXE packages. py2exe extends the Python Distutils [14] package with the a new command, "py2exe", which allows for the creation of EXE files, both 32-bit and 64-bit.

APP

APP is the filetype for Mac OS packages. It contains the applications bundle which means it contains the application and all of its resources [15]. It appears as an ordinary file in the meaning that a user can double-click the bundle to launch the application. If the package includes dependencies such as startup items, kernel extensions or other system wide resources, an APP is depreciated in favor for an *installer*[16]. For a user, APP works similar to EXE file on Windows. Instead of having dependencies and resources in different folders, an APP bundle is holding all information [17]. To create an APP bundle from python code, one can install an extra command to setuptools that is used for creating Python packages 5.4 that allows the developer to create standalone bundles, called *py2app* [18].

Because the command py2app is available as a command to setuptools [19], and

[11] https://whatis.techtarget.com/fileformat/EXE-Executable-file-program
[12] https://docs.python.org/3/faq/windows.html#how-do-i-make-an-executable-from-a-python-script
[13] https://pypi.org/project/py2exe/0.9.2.2/
[14] https://docs.python.org/3/library/distutils.html
[15] https://developer.apple.com/library/content/documentation/FileManagement/Conceptual/FileSystemProgram
[16] https://developer.apple.com/library/content/documentation/Porting/Conceptual/PortingUnix/distributing/dis
[17] https://fileinfo.com/extension/app
[18] https://py2app.readthedocs.io/en/latest/index.html
[19] https://py2app.readthedocs.io/en/latest/

setuptools is the recommendation from PyPA, py2app is selected as the packaging method for creating the APP bundle.

5.4.1 Python package

Platforms Python package is available on all three major operating systems, Windows, Mac OS and Unix-based systems such as Linux. The only requirement is that the operating system has Python installed.

Creation of package To create a Python package of the application with setuptools, the developer needs to create a script with meta data to the build process, commonly named *setup.py*. Some examples of meta data provided are name, version and dependencies [20]. The documentation lists all available options [21]. When the script is executed it creates a *source distribution*, ready to be uploaded to the PyPI repository.

Updating a package as a developer The steps a developer needs to take to update a project that resides on PyPI is few. After the code is updated, the developer needs to tag it with a new version [22]. It is also preferable to acknowledge the changes made in a special file, a change log, so the user can see what has been updated. The last step is to upload the updated project to PyPI for distribution.

Installation of package The most used process to install a Python package is with the program *pip*. Pip is the recommended way to download and install a package that is hosted on PyPI, which is the largest repository hosting Python packages. The installation process is most often one command, *pip install package*. Pip then pulls the package from PyPI and installs it on the used computer.

Updating a package as user To update an already installed package, the package manager pip is used. Pip includes several built-in methods and functions to handle the installed packages. When using the command *pip install –upgrade SomeProject* the latest version found on PyPI is downloaded and installed [23].

Dependencies If the project's dependencies are hosted on PyPI, the developer can alter the main file for creating a package, *setup.py*, and tell the installation process to install the required libraries or modules[24]. If the dependencies are

[20]https://setuptools.readthedocs.io/en/latest/setuptools.html#basic-use
[21]https://setuptools.readthedocs.io/en/latest/setuptools.html#options
[22]https://setuptools.readthedocs.io/en/latest/setuptools.html#making-official-non-snapshot-releases
[23]https://packaging.python.org/tutorials/installing-packages/#upgrading-packages
[24]http://python-packaging.readthedocs.io/en/latest/dependencies.html

not hosted on PyPI, the developer specifies the *dependency links* in the setup file. When the user then installs the application, the dependencies will also be installed.

Distribution The distribution of Python packages is mainly handled by PyPI which is the default repository for Python packages [25] and recommended by PyPA [26]. The developer needs an user account on PyPI [27] before being able to upload the application to the repository. The uploading process is preferably handled by the utility *twine* which offers a more secure way to upload instead of the built-in command for setuptools [28]. Before an upload to PyPI, the developer needs to make a dry run of the upload process to verify the setup process is satisfactory before an actual publication can be done [29].

5.4.2 Docker

Platforms Docker works on Windows, Mac OS and a variation of Linux systems: CentOS, Debian, Fedora and Ubuntu. Additionally it also have support on AWS and Azure, two Infrastructure as a Service clouds [30].

Creation of package When packaging software with Docker the developer needs to create a Dockerfile. The Dockerfile defines what should be part of the container's environment. What dependencies are needed, files and other resources [31]. The start of a Docker file is the *FROM* command. FROM specifies another Docker image to use as *parent* and continue to build on or it can be set to *scratch*. "scratch" is used if no parent image is to be used. Parent images can be chained so one image builds on another which builds on another and so on. At the base of these images is a base image which has no parent image. Moreover the developer defines files and resources to copy over to the image when it is created, with the *ADD* command. Dependencies can also be installed with the *RUN* command. As an example for a package with Python code the image needs Python installed, either a parent image based on Python can be used or Python needs to be installed with the RUN command. If the container is used as a server of some kind and needs to use ports the *EXPOSE* command is used to expose a port from inside the container to the computer that runs the container [32]. All these commands and more are available for the developer to configure everything that is needed

[25] https://packaging.python.org/glossary/#term-python-package-index-pypi
[26] https://www.pypa.io/en/latest/specifications/
[27] https://pypi.org/account/register/
[28] https://packaging.python.org/key_projects/#twine
[29] https://setuptools.readthedocs.io/en/latest/setuptools.html#basic-use
[30] https://www.docker.com/community-edition
[31] https://docs.docker.com/get-started/part2/#define-a-container-with-dockerfile
[32] https://docs.docker.com/engine/reference/builder/

to run the software. When Dockerfile is done an image can be created with the Docker *build* command in the terminal.

Updating a package as a developer For the developer to update an image, the developer only needs the new code and run the same build command that was used in the creation process. If a dependency was changed the developer also needs to update the Dockerfile to remove or add the dependency.

Installation of package The user does not need to install the image itself as everything is already installed in the image. The user only needs to have Docker installed and then run a command to start the image, package. This means that there isn't a typical installation process for the user, the user only needs to have the image and then it can then be started with the run command [33]. However the user might need to configure the run command used to start the container, this is the closest thing there is to an installation process in Docker for a user [34]. So to install an image can be defined as to configure the run command, which can be an advanced task depending on what the application does. The user could possibly need to be have knowledge about network and other advanced computer settings [33].

Updating a package as user To get the latest version a software package as a user there is the "docker pull" command, it downloads the latest version from Docker Hub [35]. Only the parts that have been changed are downloaded, so dependencies and installation processes that are the same as last version are not updated. This saves both bandwidth and time. However this command only works if the developers have uploaded the image to Docker Hub, if not the user have to download it manually from somewhere else. In either case, to use the new version the user only needs to start the container with the same command used in the installation process.

Dependencies All dependencies the software have are included in the container/package, nothing extra is needed than the container [36]. Docker solves the so called "dependency hell". Conflicting dependencies can be separated with different containers, dependencies should never be missing in new environments as all dependencies care packaged in the container [21]. To be able to run the Docker container however Docker needs to be installed.

[33]https://docs.docker.com/engine/reference/run/
[34]https://docs.docker.com/get-started/part2/#run-the-app
[35]https://docs.docker.com/engine/reference/commandline/pull/
[36]https://docs.docker.com/engine/faq/#what-does-docker-technology-add-to-just-plain-lxc

Distribution Docker comes with the fully integrated distribution system Docker Hub. It both hosts and distributes packages. There is no need for the developer themselves to host the package. However if they want to it is possible to host a private Docker registry [37].

5.4.3 EXE

Platforms Natively, EXE files only work for the Windows OS. However there are other applications which allow a user to run EXE files on other OS's. Wine is capable of running EXE files on Linux, Mac OS and BSD [38].

Creation of package The creation process is similar to the one with setuptools,in Python package, and py2APP, in APP, as all three build on the Distutils package. There are two ways to create a package with py2exe, one is to create a script, like with setuptools 5.4.1, or one can use the built in command-line utility [39]. This thesis focuses on the script method as it seems to be a more reproducible and clear method. Although the command-line tool should work the same just in another format. For the create process a Python-dll is needed, this makes it so that the EXE can be used on a computer without Python installed. Furthermore all the Python modules the software use, any possible Python-extension modules compiled and other dll's if there are any together with the source code and other resources need to be in a folder with the creation script. When building the package, one important settings is "bundle-files". "bundle-files" control how many files the output should be divided in. The value for "bundle-files" can be between 0 and 3 and it decides how the files should be bundled. Using 0 as value creates a single EXE files as result from the create process [40].

Updating a package as a developer To upgrade the package the developer only need to re-run the creation script that was created in the creation phase [41].

Installation of package An EXE file doesn't have to be installed to work. If all dependencies are encapsulated inside the EXE file, only the file itself is needed on the computer work. Nonetheless a installer can be created for the EXE, using other tools together with py2exe. This can be of interest if other EXE files are used or dependencies can't be packaged inside the EXE, if the developer want to package DLL's with the application so the user doesn't need to install them separately [42].

[37] https://github.com/docker/distribution
[38] https://www.winehq.org
[39] https://pypi.org/project/py2exe
[40] http://www.py2exe.org/old
[41] http://www.py2exe.org/index.cgi/Tutorial
[42] http://www.py2exe.org/index.cgi/Tutorial

Updating a package as user An EXE doesn't have any built in functionality to update itself. The user needs to replace the old EXE file with a new one. EXE files not being able to update itself is in part because files that are executed in Windows are locked, meaning they can't be written to or be deleted [43]. A common way to get around this is by creating a separate EXE, only for updating the other EXE file.

Dependencies An EXE file have the ability to contain dependencies and py2exe is capable of including dependencies in EXE files [44]. The user of the package doesn't even need to have Python installed as py2exe can package Python as a dll inside the EXE file [45].

Distribution There is no official distribution and hosting system for EXE packages.

5.4.4 APP (py2app)

Platforms APP is exclusively used by Mac OS. To the authors knowledge, there do not exist any stable solutions to run an APP on a different operating system. The preferred way is still natively on Mac OS or running VirtualBox [46] with Mac OS installed.

Creation of package py2app is installable by pip as an external Python package and can then be used by setuptools as a command that allows the developer to create a standalone APP-bundle. The creation process is similar to the process for creating Python packages, described in section 5.4.1. The developer runs the setup script with the command *py2app* [47].

Updating a package as developer The process for updating an APP-bundle as a developer is similar to the process used for building a python package, described in 5.4.1. The difference is the additional build command used for building an APP bundle.

Installation of package To install an APP bundle, the user needs to drag and drop the file into the destination folder [48]. The user may have to manually bypass the built-in gatekeeper to install a third-party software [49].

[43] https://en.wikipedia.org/wiki/File_locking#In_Microsoft_Windows
[44] http://www.py2exe.org/index.cgi/FrontPage
[45] http://www.py2exe.org/index.cgi/Tutorial
[46] https://www.virtualbox.org/
[47] https://py2app.readthedocs.io/en/latest/index.html
[48] https://developer.apple.com/library/content/documentation/Porting/Conceptual/PortingUnix/distributing/dis
[49] http://osxdaily.com/2016/09/27/allow-apps-from-anywhere-macos-gatekeeper/

Updating a package as user If the bundle is installed via App Store, the user only need to check for updates from the built-in menu and then download and install it. Otherwise, the bundle may be replaced with the updated version, instead of an update.

Dependencies If the package does not include startup items, kernel extensions or other system wide resources, they can be encapsulated within the bundle, depending on the developers specifications. The process follows the same instructions as for the Python package 5.4.1.

Distribution The main distributional place for APP bundles is the Mac App Store. For a developer to be able to upload the application to App Store, the application is needed to pass the App Store Review Guidelines [50]. For distributing outside the App Store, there exist some additional rules to be followed, such as signing the application with a Developer ID [51].

Summary

None of the packaging methods were able to create a package simply by running a command. Instead for every method a script was necessary to create. The process for the scripts however have different complexity to them. Python package, APP and EXE need the same type of script as they are based on the same tool. Although to create APP and EXE extra tools needs to be installed compared to Python package but the extra process is not deemed to be worth less point because Docker is more complex in the creation process.

All the packaging methods are able to reuse the script created in the creation process. Therefore they are deemed to have the same level of complexity.

Docker was the packaging method that stood out compared to the others when installing a package. The others follow the baseline while Docker does that to some extend it also needs complex settings for the command. Consequently Docker got one less point for this aspect.

None of the packaging methods support automated updating. While both Python package and Docker only need one command, EXE and APP need the user to locate an updated package and replace the old and have therefore received a lower score.

[50] https://developer.apple.com/app-store/review/guidelines/#developer-information
[51] https://developer.apple.com/macos/distribution/

Only Docker have the possibility to include all types of dependencies as a package comes with its own runtime environment and can include other applications. The other methods however also have the ability to include most dependencies, but not other applications.

Both Docker and Python package have fully integrated distribution system for packages and they are easy to utilize for both the developer and user. EXE does not have a distribution system and therefore got the lowest score and APP have the Mac App Store but it is more complex for the developer to publish their package there.

Chapter 6

Analysis and Discussion

In this section one can find the analysis and discussion for the result for each research question. This chapter is ordered in the same order as the method and result chapters. Each method has a separate section, firstly Experiment 1, secondly the Case study, thirdly Experiment 3 and lastly the theoretical analysis. Each section starts with a small summary of what the method was about. Lastly in this chapter there will be a section with answers to all research questions.

6.1 Experiment 1

The purpose of experiment 1 was to discover if multiprocessing could be used to improve the execution time for Getis-Ord* and the Aoristic method. Each method was implemented with and without utilizing multiprocessing and then the execution time for each implementation was recorded.

6.1.1 Aoristic method

Comparing tables 5.2 and 5.3 a clear connection can be seen between the amount of data used and the execution time. Using ten times more data increased the execution time about ten times, there is a linear connection between data and execution time for the Aoristic method. Furthermore looking at standard deviation it can be seen that more data leads to more variation in execution time.

Additionally, when inspecting tables 5.2 and 5.3 the execution times for Sequential and Parallel calculation it is clear that utilizing multiprocessing for only the Aoristic calculations cuts the execution time in half compared to the sequential implementation. However because of the limitations of multiprocessing, not having shared memory, extra steps are required, which are incorporated in Parallel, to achieve the end matrix which contains the result of the method. The extra steps, explained in section 3.2.3, cause an increase in execution time which surpass the execution time of the Sequential implementation. Consequently the Sequential implementation is faster than Parallel. In regards to these results it would be interesting to explore if an implementation utilizing threading could

perform better. Threading is a better option when the calculations utilize shared memory and could therefore possibly compensate for the shortcomings in multiprocessing [23].

To answer part of RQ1, multiprocessing can be used to increase the performance of the Aoristic calculations, but in the end it is not worth it as extra steps are need which increase the overall execution time to the extent that it is slower than sequential.

6.1.2 Getis-Ord* method

As seen in table 5.1 the sequential implementation has about five times lower execution time. The input dataset is of a fixed size, 24x7 and that size may be too small to benefit from a parallel implementation because the time it takes to start a new process is too high compared to the total execution time of the sequential implementation [23]. A lot of the native functionality in Python is implemented in C, and are optimized and fast as is. To further research the subject, one could experiment with different resolutions of the matrix to see how that affect the execution time.

6.2 Case study

The case study was executed by implementing the four methods and test them with four different lists, representing four possible hotspot maps. The lists were representing identical maps, dissimilar maps, partially identical maps and both hotspots and coldspots. As a complement to the tests, the similarity measure that provided the most accurate result were implemented and evaluated with authentic data.

The Jaccard measure was the only measure evaluated that had a correct score on all tests as seen in table 5.4. It is also the only measure that calculates the input data as it is, meaning not multiplying, dividing or other ways weighing the input data described in section 3.3. The other measures gives weight to the variable *a*, which is the positive matches, TT. Sørensen-Dice is multiplying the variable a by 2, meaning giving it extra weight and importance. Kulzcynski changes the weight by summing the equations, where the numerator is the variable *a*. Ochai calculates the square root of the denominators in its equation, giving extra weight to the numerator, variable *a*.

The test with authentic data proved accurate. The resulting similarity and dissimilarity were presented for all hotspots and coldspots, only hotspots as well as only coldspots. Of a total of 20 possible overlaps, 4 overlapped, resulting in

a similarity of 20% and a dissimilarity of 80%. Of 16 possible overlaps of only hotspots, 4 overlapped, giving a resulting similarity of 25% and a dissimilarity of 75%. The coldspots had no possible overlap and resulted in 0% similarity and 100% dissimilarity.

One drawback of measuring overlap with Jaccard index is that it does not take close matches into consideration. In the margin of this thesis one coordinate up or down in the original matrices is just one hour and would be helpful to get into the measurement.

The incitement of the overlap compare functionality in this thesis is to measure the absolute overlap. Jaccard index is therefore the most suitable measure of the four tested. Our result that Jaccard is suitable when negative matches are to be ignored correlates with the result from [11].

Furthermore, a calculation of the delta is also conducted based on the amount of hotspots or coldspots as well as the amount of occurrences, giving a percentage of increase or decrease between the compared maps.

6.3 Experiment 2

The purpose of experiment 2 was to discover if a months hotspots can reliably be predicted with the Naive forecasting method using data from previous months. The experiment was run with three different time windows, 1, 2 and 3. It was also run with three different datasets, the full dataset 2, subset Malmö and a server log from a website. The result for Sweden and Malmö are discussed jointly and the server data is discussed separately.

Dataset 2 and subset Malmö As one might expect, there is a clear correlation between low Jaccard values and low PAI values. However there are exceptions. These exceptions are likely to arise when there is not a high exact overlap but the hotspots have been predicted in close proximity to the actual hotspots. Therefore they might still be able to predict higher than average concentrations of crime but still not hotspot worthy values. Higher PAI values were found in Malmö compared to Sweden even though Sweden had higher Jaccard values. This is likely to linked the fact that there is a lot less data for only Malmö than when using the entire Dataset 2 for Sweden. Thus finding values, even outside of hotspots, have a bigger impact on percentage of found crimes. Unlike the result for Sweden where the method is best at predicting hotspots for winter months, for Malmö it is also able to predict some hotspots for the summer, but during the fall and spring Jaccard values with 0 are found.

Overall the three time windows showed best results for the winter months, for Sweden and Malmö. This demonstrate a bigger correlation between months during the winter than the summer. This should mean that it is easier to forecast hotspot maps for the winter than the rest of the year, especially against spring and fall where there seems to exist less of a pattern for when burglaries occur. This result demonstrate the existence of seasonality in burglaries which is an already known phenomenon. More burglaries tend to occur during fall and winter than spring and summer [31,36]. With this one can assume that, when burglaries occur is a more sporadic opportunity during spring and summer and during fall and winter it is more of a systematic job. If this is true it should be easier to predict burglaries during winter and fall, which is the case for this experiment, at least for winter. Furthermore the results suggest that there is a two month correlation window for when burglarize take place, as slightly better results were achieved when the time window increased from 1 to 2 and then the results got worse when the time window was set to 3.

Server log The results from the use case server log further amplify the poor result for the Naive forecasting method. The method performed even worse for this use case than the other two. Two possibilities for the low PAI values can be that the prediction method is extremely bad and predicted hotspot are actually coldspots. It can be that the events are very evenly spread out during the day. A possible outcome of this is that hotspots are not worth as much, as they are for burglaries where there are not that much data. This can also be seen with the two other use cases as higher PAI scores were attained in Malmö than Sweden even though Sweden had higher Jaccard values. In other words a hotspot in subset Malmö can yield a higher PAI score than a hotspot in dataset 3. One way to counter having too much data in the server log could be to change the resolution of the matrix and try to amass the data into less cells. As an example use the resolution days in months or days in weeks.

The lack of overlap in the predicted data suggest that there is not a strong, obvious, pattern between months for when people access server. It could be that the method is not capable of finding and utilizing the existing pattern.

Summary To answer RQ3, it is clear that the Naive forecast method is bad at forecasting when burglaries will happen and when people access a server. The different time windows had a small effect on the results. They all performed poorly with time window 2 being the slightly better of the three, for burglaries. Time window 1 is the same as the method Random walk, that have been evaluated for spatial hotspots [36]. They found that Random walk, or time window 1, is bad at forecasting one month ahead for spatial hotspots. As discovered in this

thesis the same can be said for temporal hotspots. The results show that there is a difference from month to month when crime happens and that there is a need for a good method that can forecast short term temporal hotspots. Using simple methods such as random walk or Naive forecast result in misleading hotspot maps, especially during the spring and fall, for burglaries.

6.4 Theoretical analysis

The purpose of the theoretical analysis was to research how complex different packaging methods are according to seven aspects. The packaging methods that were researched is Python package, Docker, EXE, and APP.

Python Package received the highest score with 18 points and Docker came in second with 17. These two are the best methods, using the authors criterions. APP and EXE fall behind partly because they only work for one OS each but also because of how they handle distribution and updating.

The methods Python package, EXE and APP are very similar in the packaging process. py2exe (EXE) and py2app (APP) are both commands to setuptools which is used for Python package. Compared to Python package, EXE and APP only benefit if the aim is to package a product for a single operating system and a requirement is that Python is not installed on the target machine. Python is the only asset that needs to be installed to be able to use Python package and Python package benefits of a fairly simple distribution system.

Docker has drawbacks in the complexity of installing it on the operating system, since it is somewhat complex in comparison to the other methods. The main benefit for using Docker is that, similar to Python Package, it is deployable on all three main operating systems, supported by a distribution system. Furthermore, it has its own runtime environment and all the code is intact and available to the user. This can prove useful when distributing a package for other developers or computer scientists that have a need for running the implemented tests or in some way need the original code. The developer can also make the assumption that if it works on their machine it should work on the users machine. This can be useful for research reproducibility and validation [1, 4].

From the perspective of a normal user, which the authors defines as a user that is not familiar with the terminal tool or has a need for the implemented code itself, the preferred method depends solely on the operating system the user has. Both APP and EXE are platform specific. Because of that, as well as the simplicity of handling updates and installation, they are the preferred methods for the normal user only.

An experienced user is defined by the knowledge of the terminal tool. For this user, the method Python Package is preferable because with Python Package, the user has a more unified handling process. Both the installation and update process is handled by pip. If the user wants to see the code or wants to have the ability to run tests on the application, Docker is the best choice. With Docker, all the code is preserved and will be available after installation.

Even though Python package received the highest score the authors of this thesis would use Docker as a packaging method. Because the end user of the prototype is either a researcher or a technical person in law enforcement, or at least has the help of an IT department, they should be able to handle the extra complexity. Docker allows for a easier updating and installation process for the law enforcement or IT department and allows for a researcher to verify the result of this thesis and further develop it.

6.5 Answers to the research questions

Here are short answers to the research question that were answered in this thesis.

RQ 1 To what degree can multiprocessing be utilized to increase performance of the methods Getis-Ord* and Aoristic compared to a serial implementation?

Multiprocessing was found to not improve the execution time for either Getis-Ord* or the Aoristic method.

RQ 2 To what extent can different hotspots be compared and provide significant similarity, difference and overlap?

Two hotspots can be compared and the method Jaccard Index provides an exact percent of similarity as well as dissimilarity, when calculating the overlap. The use of delta calculation can add a percentual increase or decrease in the amount of crimes as well as the amount of hotspots and coldspots.

RQ 3 How accurately can historic data be used to predict future temporal hotspots using naive methods?

The Naive forecasting method, and Random walk, that was used in this thesis was found to not be potent enough to predict temporal hotspot maps with a satisfactory results. The experiment was run with the resolution weekday by hour-of-day using both burglary data and when a website was accessed.

RQ 4 How do established software packaging methods compare in regards to usage complexity for developers and end users?

Four different packaging methods were investigated and they could be divided into two groups, simple methods and advanced methods. The simple methods are EXE and APP, they do not have as much functionality as the advanced, but they focus on an OS each. The advanced methods work for all three major operating systems and have distribution systems. Python package scored the highest score with the aspects and criteria the authors used. Docker is a bit more complex but have additional features which makes it desirable.

Chapter 7

Validity Threats

The following section presents and discusses the validity threats to the various methods chosen in this thesis based on the validity threats to empirical software engineering present by Robert Feldt and Ana Magazinius [12].

7.1 Experiment 1

To assure that treatment of the experiment is related to the outcome, *conclusion validity*, a t-test was performed to validate statistical significance between the different implementations.

The experiment was conducted on a system running other applications which can cause variance in the execution time because of job and resource scheduling. To mitigate this *internal validity*, each measurement in the experiment was executed 10 times and the average from these runs were used along side t-test. Another possible threat is how Python have implemented the multiprocessing functionality. A possible way to mitigate this would be to implement the experiment in several languages and compare their results.

The authors are aware of the *construct validity* threat of the possibility that the observed outcomes are not cause by the treatment. To offset this threat the sequential implementations were create first, followed by only adding code related to multiprocessing. This way the authors tried to avoid adding unnecessary code. To *generalize* experiment 1, different size datasets were used.

7.2 Experiment 2

The experiment was executed evaluated using two measurements Jaccard, for exact overlap, and PAI, for value of the predicted hotspots, to mitigate *conclusion validity*. If the predicted hotspots are located adjacent to the actual hotspot Jaccard would show show it as a bad prediction. However there can still be value in predicting adjacent to the actual hotspot if there are a lot of crimes. If

this is the case, the PAI score can display the prediction to be somewhat valuable.

A possible *internal validity* threat is only using data from 2014 and from Sweden, to mitigate this use case Malmö and server log was added.

The authors offset the *external validity* threat of generalization by conducting the experiment on both burglary data and access to a web server additionally to running each use case 12 times, each month for a year.

7.3 Case Study

To handle the *conclusion validity*, the authors identified four cases consisting different potential outcomes of a calculated hotspot map. The result were then validated with a test with real authentic data as input.

The *internal validity* threat is mitigated by using lists with a size of 10 in the tests. The result from the comparison tests are then easy to validate and evaluate, because the aim is to evaluate the percentage. The tests are also executed on four lists with different expected outcome.

The lists used in the case study contains only ones and zeros, which mitigates the *external validity* threats to some extent. The measure is *generalized* to lists with similar content, which is proven when validating with authentic data.

7.4 Theoretical Analysis

Credibility validity is assured by using official documentation and web sites specialized on certain areas as data sources.

A potential threat to the construct validity is that the authors have not tested the packaging methods. The analysis result may be different from the actual process and necessary steps needed to package, update and handle the code.

Chapter 8

Conclusions and Future Work

This thesis has researched temporal hotspot maps and related methods, to fill a scientific research gap. The thesis also investigates how the methods used for creating hotspot maps may benefit from multiprocessing. Furthermore different ways to package Python code was discussed. This section contains the conclusions and suggestions for future work.

The method Getis-Ord* is widely used in crime analysis as well as in statistical analysis. The method provides a metric, a z-score, that shows if a hotspot appears with some statistical significance or if it may have appeared by chance.

The thesis concludes that the Getis-Ord* method does not benefit from multiprocessing. The execution time was instead remarkably increased. The reason for this is that the data used as input is always in the resolution 7x24 and not big enough that multiprocessing is beneficial for balancing the workload. The calculation is so fast sequentially that just to startup a new process increases the execution time. Many of Pythons built-in functions are implemented and optimized in C for performance and therefore have a fast execution time. For future work, it would be interesting to measure the parallelized execution time on matrices with a larger resolution and see how it scales with resolution size.

Previous research suggests that the Aoristic method is the best and most popular choice for handling crime with a timespan. Better execution time was not achieved for the Aoristic method, when utilizing multiprocessing. Multiprocessing has the constraint of not being able to share memory fast and effectively. This created circumstances where dividing the task into multiple process did not help decrease the execution time but instead increased it. Multi threading on the other hand can handle shared memory and therefore it would be interesting to see a similar experiment where threading is utilized.

It is of interest to have the ability to compare two calculated hotspot maps, to see if the hotspots appear at the same time. Another reason is to see to what extent the compared maps have a similarity in their overlap. This thesis concludes that the method that most accurately provides an exact result is the Jaccard

Similarity Index. Of the methods tested, it is the only method that compares the maps as they are. Other methods weigh the positive matches and giving them more value in the calculation. This resulted in a non-exact percentage. The prototype also provides the percentage of the delta increment and decrement in the compared hotspot maps. An interesting angle in future work is to include the fuzziness in the comparison and give a weight to close matches. The hotspot maps in this thesis is using hours by weekdays as resolution and a hotspot one step up or down represents an hour. One hour is a small metric and may be worth some weight when comparing the overlap. The result may be more beneficial to the user if this is taken into consideration in the comparison.

Previously naive methods, such as random walk, have been researched on spatial hotspots. It is concluded that naive methods work poorly for prediction. This thesis reaches the same conclusion for temporal hotspots on burglary data and access logs from a server. Although the result from the Naive forecasting method suggested that for Swedish burglary data there is some connection of when burglaries occur. However the results were still too unreliable to be of use. To further establish the unreliable result one could do a similar experiment where more data is used. Have more years of data and have several smaller areas, not only Malmö and Sweden like in this thesis. Another option could be to investigate how different resolutions affect the efficiency of the forecasting, if some resolutions are more suitable. Furthermore only one type of crime was used, burglary, future research can be done with different types of crimes. However the authors suggest that resources should be spent on more advanced methods instead as related work suggest that more advanced methods have achieved better results.

The packaging methods investigated have different qualities and some are preferable depending on the target users' environment. The methods APP and EXE have a relatively low complexity to the packaging process and targets the operating systems Windows and Mac OS respectively. The disadvantage is that for a normal user, a simple distribution system is of interest. EXE has no distribution system connected to it, APP may be published on App Store or any other third party distribution system, but then the developer need to do extra steps in the packaging process, which in turn adds to the complexity of packaging the code. If the user has more experience in terms of knowledge of the terminal, the methods Python package and Docker may provide a more preferable approach. For both Python package and Docker, there exist a complete distribution system, which the developer and end user could utilize to both install the application as well as update it. If the user has a need for the original code or an isolated runtime, then Docker is the preferable packaging method, which was the only method that preserves the code. To further evaluate the packaging methods, the authors suggests that they are all implemented and tested. This would also be beneficial for the theoretical analysis done in this thesis and compare the outcomes.

Chapter 9

Contribution

9.1 Thesis contribution

The result from the thesis contributes to the academia by adding to the lacking amount of research done on temporal hotspots. The thesis more specifically provides methods for comparing temporal hotspot maps as well as proved that prediction should not be done with naive methods such as Random walk. Finally, the thesis contributes by listing four packaging methods and discusses their aspects, which has not been found in previous research.

The thesis contributes to the industry by the created prototype. The prototype is, as wanted by law enforcement, easy to use and interpret. The prototype provides functionality that both creates hotspot maps and visualizes the comparison, in an understandable way. Furthermore the thesis concluded that naive methods that law enforcement already used, Random walk, should not be used for predicting temporal hotspots. Additionally the comparison of the packaging methods can be used by developers to help with the decision of what methods to use for packaging their software.

9.2 Individual contribution

The workload for this thesis have been divided on a research question basis between the two authors. Andreas have researched the Aoristic method and worked on the parts surrounding that method while Kenneth researched Getis-Ord*. This way RQ1 was divided into two parts and both authors contributed. RQ2 on the other hand only Kenneth worked on at the same time as Andreas worked on RQ3. The authors collaborated again on RQ4, the aspects are a result of their combined work while the packaging methods were divided. Kenneth researched Python package and APP whilst Andreas worked on Docker and EXE. The prototype used throughout the thesis is implemented by both authors.

References

[1] Reinhard Laubenbacher Thibauld Favre Abdelrahman Hosny, Paola Vera-Licona. Algorun: a docker-based packaging system for platform-agnostic implemented algorithms. *Bioinformatics*, 32(15):2396–2398, 2016. Available `https://doi.org/10.1093/bioinformatics/btw120`.

[2] Luc Anselin. Local indicators of spatial association—lisa. *Geographical Analysis*, 27(2):93–115, 1995. Available `http://dx.doi.org/10.1111/j.1538-4632.1995.tb00338.x`.

[3] Silberschatz A.,Gagne G.,Galvin P. B. *Operating System Concepts 9th Edition*. John Wiley & Sons Ltd, 2009.

[4] Carl Boettiger. An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review - Special Issue on Repeatability and Sharing of Experimental Artifacts*, 49(1):71–79, 2015. Available `https://doi.org/10.1145/2723872.2723882`.

[5] Erik Johansson,Christoffer Gåhlin,Anton Borg. Crime hotspots: An evaluation of the kde spatial mapping technique. *Intelligence and Security Informatics Conference*, 2015. Available `https://doi.org/10.1109/EISIC.2015.22`.

[6] Martin Boldt, Anton Borg. Evaluating temporal analysis methods using residential burglary data. *ISPRS International Journal of Geo-Information*, 5(9):148, 2016. Available `http://dx.doi.org/10.3390/ijgi5090148`.

[7] Matthew PJ. Ashby,Kate J. Bowers. A comparison of methods for temporal analysis of aoristic crime. *An Interdisciplinary Journa*, 2(1), 2013. Available `https://doi.org/10.1186/2193-7680-2-1`.

[8] S.H.,Tappert C.C. Choi, S.S.,Cha. A survey of binary similarity and distance measures. *Journal of systemics, cybernetics and informatics*, (8):43, 2010. Available `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.352.6123&rep=rep1&type=pdf`.

[9] Andréia da Silva Meyer et al. Comparison of similarity coefficients used for cluster analysis with dominant markers in maize (zea mays l). *Genetics and*

Molecular Biology, 27(1):83–91, 2004. Available http://dx.doi.org/10.1590/S1415-47572004000100014.

[10] Carter C. Price et al. *Predictive Policing*. RAND Corporation, 2013.

[11] Daniel P. Faith. Asymmetric binary similarity measures. *Oecologia*, 57(3):287–290, 1983. Available https://link.springer.com/article/10.1007/BF00377169.

[12] Robert Feldt and Ana Magazinius. Validity threats in empirical software engineering research - an initial survey. In *SEKE*, 2010.

[13] Marcus Felson and Erika Poulsen. Simple indicators of crime by time of day. *International Journal of Forecasting*, 19(4):595 – 601, 2003. Available https://doi.org/10.1016/S0169-2070(03)00093-1.

[14] Michael Leitner,Marco Helbich. The impact of hurricanes on crime: A spatio-temporal analysis in the city of houston, texas. *Cartography and Geographic Information Science*, 38(2):213–221, 2011. Available http://dx.doi.org/10.1559/15230406382213.

[15] Z. Hubalek. Coefficients of association and similarity, based on binary (presence-absence) data: An evaluation. *Biological Reviews*, 57(4):669–689, 1982. Available https://doi.org/10.1111/j.1469-185X.1982.tb00376.x.

[16] J.Andrew Ware Jonathan J.Corcoran, Ian D.Wilson. Predicting the geo-temporal variations of crime and disorder. *International Journal of Forecasting*, 19(4):623–634, 2003. Available https://doi.org/10.1016/S0169-2070(03)00095-5.

[17] Sergio J. Rey,Elizabeth A. Mack,Julia Koschinsky. Exploratory space–time analysis of burglary patterns. *Journal of Quantitative Criminology*, 28(3):509–531, 2012. Available https://doi.org/10.1007/s10940-011-9151-9.

[18] Ned Levine. The "hottest" part of a hotspot: Comments on "the utility of hotspot mapping for predicting spatial patterns of crime". *Security Journal*, 21(4):295–302, 2008. Available https://doi.org/10.1057/sj.2008.5.

[19] William J. Bratton, Sean W. Malinowski. Police performance management in practice: Taking compstat to the next level. *Policing*, 2(3):259–265, 2008. Available http://assets.lapdonline.org/assets/pdf/WJB%20SWM%20Article%20Oxford%20Journal.pdf.

[20] Jerry H. Ratcliffe,Michael J. McCullagh. Aoristic crime analysis. *International Journal of Geographical Information Science*, 12(7):751–764, 1998. Available `http://dx.doi.org/10.1080/136588198241644`.

[21] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journa*, (239), 2014. Available `https://dl.acm.org/citation.cfm?id=2600241`.

[22] Amanda J. Keledjian, Sarah Mesnick. The impacts of el nino conditions on california sea lion (zalophus californianus) fisheries interactions: Predicting spatial and temporal hotspots along the california coast. *Aquatic Mammals*, 39(3):221–232, 2013. Available `http://dx.doi.org/10.1578/am.39.3.2013.221`.

[23] B. Nichols, D. Buttlar, J. Farrell, and J. Farrell. *PThreads Programming: A POSIX Standard for Better Multiprocessing*. A POSIX standard for better multiprocessing. O'Reilly Media, Incorporated, 1996.

[24] Arthur Getis,J.K. Ord. The analysis of spatial association by use of distance statistics. *Geographical Analysis*, 24(3):189–206, 1992. Available `https://10.1111/j.1538-4632.1992.tb00261.x`.

[25] Xueming Shu Peng Chen, Hongyong Yuan. Forecasting crime using the arima model. *Fuzzy Systems and Knowledge Discovery*, 5:627–630, 2008. Available `https://doi.org/10.1109/FSKD.2008.222`.

[26] Jerry H. Ratcliffe. Aoristic analysis: the spatial interpretation of unspecific temporal events. *International Journal of Geographical Information Science*, 14(7):669–679, 2000.

[27] Jerry H. Ratcliffe. The hotspot matrix: A framework for the spatio-temporal targeting of crime reduction. *Police Practice and Research*, 5(1):05–23, 2004. Available `http://dx.doi.org/10.1080/1561426042000191305`.

[28] Jerry H. Ratcliffe. Aoristic signatures and the spatio-temporal analysis of high volume crime patterns. *Journal of Quantitative Criminology*, 18(1):23–43, 2012. Available `https://doi.org/10.1023/A:1013240828824`.

[29] Spencer Chainey,Jerry H. Ratcliffe. *GIS and crime mapping*. John Wiley & Sons Ltd, 2005.

[30] David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures, Fifth Edition*. CRC PRESS, 2011.

[31] David W.M Sorensen. *Temporal patterns of Danish residential burglary*. Research Department III, Faculty of Law University of Copenhagen, 2004.

[32] Sebastian Uhlig Spencer Chainey, Lisa Tompson. The utility of hotspot mapping for predicting spatial patterns of crime. *Security Journal*, 21(1–2):4–28, 2008. Available https://doi.org/10.1057/palgrave.sj.8350066.

[33] Andrew W Swain. A comparison of hotspot mapping for crime prediction, 2012. Available https://www.geos.ed.ac.uk/~gisteac/proceedingsonline/GISRUK2012/Papers/presentation-72.pdf (Accessed: 2018-03-19).

[34] Lisa Tompson and Michael Townsley. (looking) back to the future: Using space—time patterns to better predict the location of street crime. *International Journal of Police Science & Management*, 12(1):23–40, 2010. Available https://doi.org/10.1350/ijps.2010.12.1.148.

[35] Elizabeth A. Mack Tony H. Grubesic. Spatio-temporal interaction of urban crime. *Journal of Quantitative Criminology*, 24(3):285–306, 2008. Available https://doi.org/10.1007/s10940-008-9047-5.

[36] Yvonne Thompson Wilpen Gorr, Andreas Olligschlaeger. Short-term forecasting of crime. *International Journal of Forecasting*, 19(4):579–594, 2003. Available https://doi.org/10.1016/S0169-2070(03)00092-X.

[37] Kim M. Wong K.S. Privacy-preserving similarity coefficients for binary data. *Computers & Mathematics with Applications*, 65(9):1280–1290, 2013. Available https://doi.org/10.1016/j.camwa.2012.02.028.

Appendix A

Experiment 2 tables

A.1 Forecasting Sweden

Month	Jaccard %	PAI
1	0	1.165
2	54	2.024
3	40	1.722
4	0	1.593
5	12.5	1.446
6	0	1.131
7	0	1.293
8	0	1.086
9	0	1.496
10	0	1.067
11	14.3	1.966
12	23.5	1.887

Table A.1: Result for time window 1.

Month	Jaccard %	PAI
1	60	2.294
2	28.6	1.798
3	16.7	1.439
4	0	1.595
5	0	1.313
6	8.3	1.273
7	0	1.423
8	25	1.613
9	0	1.315
10	0	1.087
11	7.1	1.899
12	21.4	1.974

Table A.2: Result for Naive forecast with time window 2.

Month	Jaccard	PAI
1	44.4	2.676
2	50	1.970
3	18.1	1.343
4	0	1.486
5	0	1.307
6	0	0
7	0	1.404
8	0	1.331
9	0	1.399
10	0	0.730
11	8.3	1.427
12	14.2	1.9

Table A.3: Result for Naive forecast with time window 3.

A.2 Forecasting Malmö

Month	Jaccard %	PAI
1	0	1.948
2	50	3.204
3	0	1.495
4	0	1.081
5	18.7	2.260
6	5.5	1.459
7	7.6	2.179
8	0	1.650
9	0	1.039
10	6.2	1.811
11	12.5	1.825
12	0	1.866

Table A.4: Result for time window 1.

Month	Jaccard %	PAI
1	18.1	2.603
2	36.3	3.013
3	0	1.542
4	0	1.143
5	40	2.690
6	13.3	1.530
7	11.7	1.952
8	5.8	1.777
9	5.2	1.177
10	0	1.114
11	5.8	1.844
12	0	1.844

Table A.5: Result for Naive forecast with time window 2.

Month	Jaccard	PAI
1	20	2.550
2	45.4	2.912
3	0	1.308
4	0	1.837
5	13.3	2.170
6	6.6	1.351
7	20	2.085
8	5.5	1.635
9	0	1.731
10	0	1.097
11	6.2	2.064
12	0	2.052

Table A.6: Result for Naive forecast with time window 3.

A.3 Forecasting Server log

Month	Jaccard %	PAI
1	0	1.21
2	0	0.968
3	20	0.939
4	0	0.975
5	5.2	0.458
6	0	1.012
7	0	0.243
8	0	0.368
9	0	1.411
10	0	0
11	0	1.485
12	0	1. 213

Table A.7: Result for Naive forecast with time window 1.

Month	Jaccard %	PAI
1	0	1.409
2	0	1.022
3	0	1.264
4	0	0.553
5	9	0.963
6	0	0.433
7	0	1.029
8	0	0.418
9	0	1.177
10	21.4	0.45
11	0	0
12	0	1.145

Table A.8: Result for Naive forecast with time window 2.

Month	Jaccard	PAI
1	0	0.57
2	11	1.237
3	0	1.367
4	0	0.996
5	6.6	0.605
6	16.7	0.972
7	0	0.43
8	38.9	1.968
9	0	0.314
10	0	0.482
11	16.7	1.636
12	0	0

Table A.9: Result for Naive forecast with time window 3.

Appendix B

Aoristic parallelism algorithms

B.1 Multiprocessing shared memory pseudo code

Algorithm 9 Aoristic method, multiprocessing with shared memory

```
{input is events as Datetime objects in a list}
#Init matrix of resolution Days-of-week x Hours-in-day
matrix[7][24] = 0
shared_matrix = create_shared_matrix(matrix)
# number of events for each process to process
chunk = events.length / number of processes
event_counter = 0
for each process do:
    while event_counter < chunk
      event = events[event_counter]
      nr_hours_span = ceiling((event.total_seconds()/3600))
      aoristic_value = (1 / nr_hours_span)
      duration_counter = 0
       while duration_counter < nr_hours_span
         x = event.get_weekday()
         y = event.get_hour()
         new_value = shared_matrix[x][y] + aoristic_value
         shared_matrix[x][y] = new_value
         duration_counter ++
         event.add_hour(1)
       end while
  event_counter ++
   end while
end for
{the variable 'shared_matrix' now contains the temporal probability of when
the events took place and is ready to be analyzed with Local Getis-Ord*}
```

B.1.1 Multiprocessing pseudo code

Algorithm 10 Aoristic method, multiprocessing without shared memory

```
{input is events as Datetime objects in a list}
list_matrices = list()
{Part 1}
chunk = events.length / number of processes
event_counter = 0
for each process do:
    while event_counter < chunk
      matrix[7][24] = 0{Init matrix of resolution Days-of-week x Hours-in-day}

      event = events[event_counter]
      nr_hours_span = ceiling((event.total_seconds()/3600))
      aoristic_value = (1 / nr_hours_span)
      duration_counter = 0
       while duration_counter < nr_hours_span
         x = event.get_weekday()
         y = event.get_hour()
         new_value = matrix[x][y] + aoristic_value
         matrix[x][y] = new_value
         duration_counter ++
         event.add_hour(1)
       end while
   event_counter ++
   list_matrices.add(matrix)
    end while
end for
{Part 2}
sum_matrix[7][24] = 0
for matrix in list_matrix
   for row_index in matrix.length
      for col_index in matrix[row_index].length
         value = matrix[row_index][col_index]
         sum_value=sum_matrix[row_index][col_index] + value
         sum_matrix[row_index][col_index] = sum_value
{the variable 'sum_matrix' now contains the temporal probability of when the
events took place and is ready to be analyzed with Getis-Ord*}
```

Appendix C

PAI Implementation

Algorithm 11 PAI implementation

```
hotspot_map = remove_coldspots(predicted_hotspot_map) # replace coldspot values with value 0
hotspot_indexes = find_indexes(hotspot_map) # Find indexes of hotspots in matrix
n = 0 # Nr Crimes in hotspots
for index in hotspot_indexes do
   n = n + study_area[index]
end for
N = sum(study_area) # calculate number of crimes in study area
a = count_nonzero_values(hotspot_map) # Count number of hotspots in matrix
A = study_area.size # get total number of spots in study_area
pai = ((n / N) * 100) / ((a / A) * 100) # Calculate PAI value
{The variable pai holds the PAI value for predicted_hotspot_map}
```

Appendix D

Similarity Measures Implementations

Algorithm 12 Implementation of Jaccard Index

```
# the function setup_data() returns TT, TF, FT
a, b, c = setup_data(left, right)
return a / (a + b + c)
```

Algorithm 13 Implementation of Sørensen-Dice Index

```
# the function setup_data() returns TT, TF, FT
a, b, c = setup_data(left, right)
return 2*a / ((2*a) + b + c)
```

Algorithm 14 Implementation of Kulczynski Index

```
# the function setup_data() returns TT, TF, FT
a, b, c = setup_data(left, right)
return 0.5 * ( (a / (a + b)) + (a / (a + c)) )
```

Algorithm 15 Implementation of Ochai Index

```
# the function setup_data() returns TT, TF, FT
a, b, c = setup_data(left, right)
return a / sqrt( (a + b) * (a + c) )
```

Appendix E

Authentic data for RQ2

The following tables is the authentic data used in the test in RQ2. The actual values have been cleared and replaced with -1 and 1. The matrices will be flattened before passed to the similarity methods functions.

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	-1	0	0	0	0
0	0	-1	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	1	1	0
0	0	0	0	1	1	0
0	0	0	0	1	1	0
0	0	0	0	1	1	0
0	0	0	0	1	1	0
0	0	0	0	0	0	0

Table E.1: Presentation of the Gothenburg data.

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	-1	-1	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	0	1	1	0	0
0	0	1	1	1	0	0
0	0	0	1	1	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Table E.2: Presentation of the Karlskrona data.