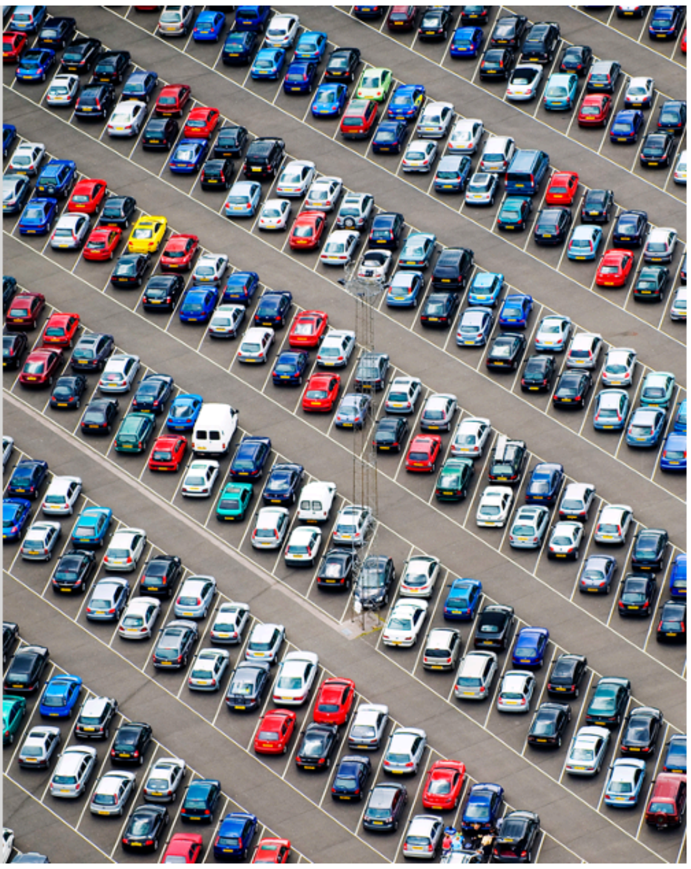
APPLIED RESEARCH PROJECT [MATH 2191]
Due: 06/06/2021



PARKING SENSOR ANALYSIS

SPATIAL TEMPORAL ANALYSIS OF IoT DATA

Authors:

Shayan Ahmed [s3799970],
Shivani S. [s3763179],
Bhargav Rele [s3761977]

Supervisor:

Sevvandi Kandanaarachchi

Acknowledgment

Thanks are extended to RMIT University, our project supervisor Sevvandi Kandanaarachchi, and our subject supervisors Yan Wang, Linda To and Denwick Munjeri.

Disclaimer

Executive Summary

Spatio-temporal statistical operations were used on parking sensor IoT data, in the city of Melbourne, to identify anomalous events within parking behaviour. Using the first week of January 2019, as our sample's temporal domain, and the suburb of North Melbourne as our sample's spatial domain, the project entailed constructing a Kriging model that predicts parking durations for parking bays by taking into account spatio-temporal variability. Followed by which, outlier detection was used to detect anomalous parking durations amongst these predictions.

Table of Contents

1. Introduction

Parking sensor data, when analysed is a great tool that enables more efficient city-planning and alleviation of traffic congestion. By analysing sensor data within an entire city, we gain an understanding of what affects parking behaviour across space and time, and how they are affected due to surrounding factors. For instance, at the origination of an event, parking behaviour within a locality may be affected due to road and street closures. Hence, an event of this sort may be unplanned (anomalous and short term – for example, emergency closures) or planned (expected and long term – for example, construction work). The primary goal is to predict durations of parking events and identify anomalous behaviour, using parking sensor data collected in the City of Melbourne for the year 2019. During anomalous events, emergency service's (or city planner's) activities may demand the closure, or temporary cordoning off of parking bays and their surrounding streets. This may cause changes in parking behaviour and decision making, thus exacerbating traffic congestion within the events locality and nearby regions. Hence, by observing *predicted* durations for parking bays, we can gain an understanding of what we can *expect* parking behaviour to look like at a point in space and time. By observing *actual* durations for the same parking bays, we can gain an understanding of what the behaviour *actually* looks like at the same point in space and time. By finding outliers within the distribution for the deviation between actual durations and expected durations, we can then identify anomalous parking durations. Clustering of anomalous parking durations within close proximity of each other, can then indicate the horizon of an anomalous event.

2. Literature Review

With the advent and usage of IoT data, smart cities such as Los Angeles (Dance 2014, p. 26), San Francisco (Pierce & Shoup 2013, p. 67) and Barcelona (Worldsensing 2016) have made efforts to analyse parking sensor data to monitor large urban areas. Real-time parking availability information may also enable civilians to efficiently find empty parking spots, thus reducing traffic congestion and improving city-planning capabilities. Such applications are explored in detail by Yang, Portilla & Riesgo (2012, p. 25), through the creation of a Parking Guidance and Information System. Research conducted by Shoup (2006, p. 479), Wang and He (2011, p. 690) and, Geng and Cassandras (2013, p. 1129) also focus on applications of IoT parking data in predicting real-time parking availability. Gupta et al. (2014) use swarm-based intelligence to provide real-time dashboard visualizations to drivers looking for parking spots, through online wireless communication. In Melbourne itself, Salim (2021) in her project monitors 650 parking spaces in addition to other amenities such as bins, BBQ facilities, toilet blocks and shopping streets using sensor data, in order to ensure proper functioning of parking systems and alleviation of traffic congestion caused by parking unavailability. Lastly, Vlahogianni et al. (2016, p. 192) use Weibull parametric models to make predictions on parking space availability for several time periods ahead, using historic data in the city of Santander, Spain. Hence, a wide range of literature focuses on predicting parking availability for usage in parking reservation and recommender systems, that may help alleviate traffic congestion and aid city-planning. However, only the works of a few focus on anomaly detection using parking sensor data. Zheng et al. (2014, p. 1) utilize temporal clustering on parking sensor data, for anomaly detection. Piovesan et al. (2015, p. 192) also do the same however; not only is outlier detection used to identify anomalies, but also clustering is used to group distinct regions based on patterns in parking behaviour. Their study also entails the usage of a more original method for unsupervised clustering, termed "self-organizing maps". For our research, the most relevant related work, is that of Zheng et al. (2014, p. 1), Piovesan et al. (2015, p. 192) and Lu and Liao (2018, p. 500), given that their aim was to predict parking behaviour and detect anomalies using clustering methodology. However, instead of using a naïve formula for quantifying parking behaviour, as had been done in their work, we simply use duration figures of parking bays. Furthermore, instead of using the clustering methodology to identify outliers, we wish to use Spatio-temporal statistical methodology to first predict parking durations, and then identify outliers using a simpler z-score method.

Given that parking durations are measured a certain point in space and time, spatio-temporal statistics may be of better use in predicting them. We first go through the subsetting operations required to create a spatio-temporal dataset. Followed by which we visualize the spatio-temporal covariability of durations using a Variogram. We then fit a model Variogram on our data to predict durations. Lastly, we find deviations between actual and predicted durations, and find outliers within its distribution.

3. Methodology

The dataset is provided by the City of Melbourne and, consists of parking bay sensor information for on-street parking spots in the CBD for the year 2019. Each parking within the CBD has an in-ground parking sensor that comes active when a vehicle enters the parking, up until it departs, thus recording information about time, location and duration for each parking instance. This information, for the year 2019, had then been added to the dataset that had been updated every 2 minutes.

Given that each parking instance for the year of 2019 had been recorded, the dataset consists of approximately 42.7 million rows and 20 features. Hence our primary goal would be to reduce the size of our dataset to more manageable dimensions.

3.1 Dataset Operations

3.1.2 Subsetting using Time and Space

From all features, the information we are most concerned with can be found in ‘DepartureTime’ and ‘DurationMinutes’. Other features include information about street corners, parking restrictions and parking signs. When dealing with space-time datasets, we essentially need locations in space, locations in time and the variable we are observing/modelling (Pebesma 2021). Hence, we can reduce the dimensionality of our data by reducing the number of features to 2 (coordinates and departure time). Despite this, given computational limitations and time-constraints, working with 42.7 million rows of data on R could be tedious even on 2 features. Downloading and importing datasets of such calibre into R was impossible.

Hence, our next dataset operation involved choosing an appropriate way to subset the rows of our data without losing any valid information. This is particularly tricky given that; firstly, parking durations observed in January may be lower, on average, than parking durations observed in December (perhaps due to increased activity during festivities in December). Secondly, each week of the month may exhibit different average durations. For instance, the last week of December and first week of January may have different average parking durations than surrounding weeks, due to Christmas and New Year celebrations. Thirdly, each day of the week may exhibit different behaviour patterns given that residential areas may have higher parking durations during weekends and lower durations during weekdays. Differences in parking behaviour patterns may also depend on the hour of the day. Parking durations in the CBD may be higher during office hours while parking durations in residential areas may be higher after office hours.

Hence, the temporal seasonality is observable across months, weeks, days and hours. In order to prevent losing information associated with such seasonality, it is may be appropriate to first subset our data to only include observations for January of 2019. By doing so, we reduce the number of rows to 1.4 million. We further subset the data to only include the first week of January. Therefore, we manage to maintain the hourly and daily variation at least, if not the monthly and weekly variation. Furthermore, instead of using durations for parking spots themelves,, we group them together as parking bays. To do so, we use the feature ‘BayID’, and its coordinates (available inn another dataset on the City of Melbourne website). Having merged both datasets using ‘DeviceID’ as the key variable, we proceed with grouping parking spots into their respective parking bays. ‘DepartureTime’ and ‘BayID’ are used as key variables in the grouping. The dataset now consists of 600,000 rows. The features include ‘DepartureTimes’, ‘DurationMinutes’, ‘BayID’ and coordinates. We can visualise the parking bays (grouped parking spots), as shown in *figure 3.1*. It is evident that parking bays outside of the CBD (averaged across time), have higher durations than parking bays within the CBD. If we subset any further, we will lose most of our temporal seasonality. Hence, further subsets would have to be made using spatial locations, and not temporal ones.

To subset spatially, we filter out observations using latitude and longitude figures. Due to the widespread availability of parking spots in a small locality, we can subset a residential area, as opposed to the CBD. Hence, we manage to reduce the number of rows to about 100,000, by including only North Melbourne parking bays. Therefore, the final dataset that we will be working with consists of 100,000 measurements of parking durations for all on-street parking bays in North Melbourne, as average durations of all parking spots within a given bay, for the first week of January 2019. A visualisation of the spatial-subsetting is provided in *figure 3.2*.

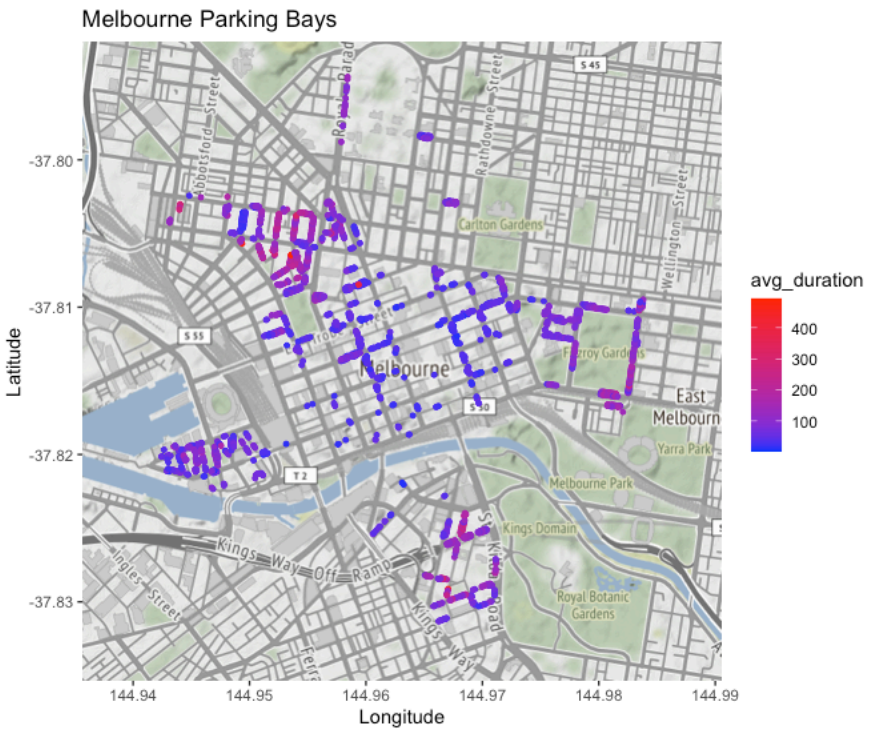


Figure 3.1: Durations for parking bays, in the first week of January 2019, Melbourne.

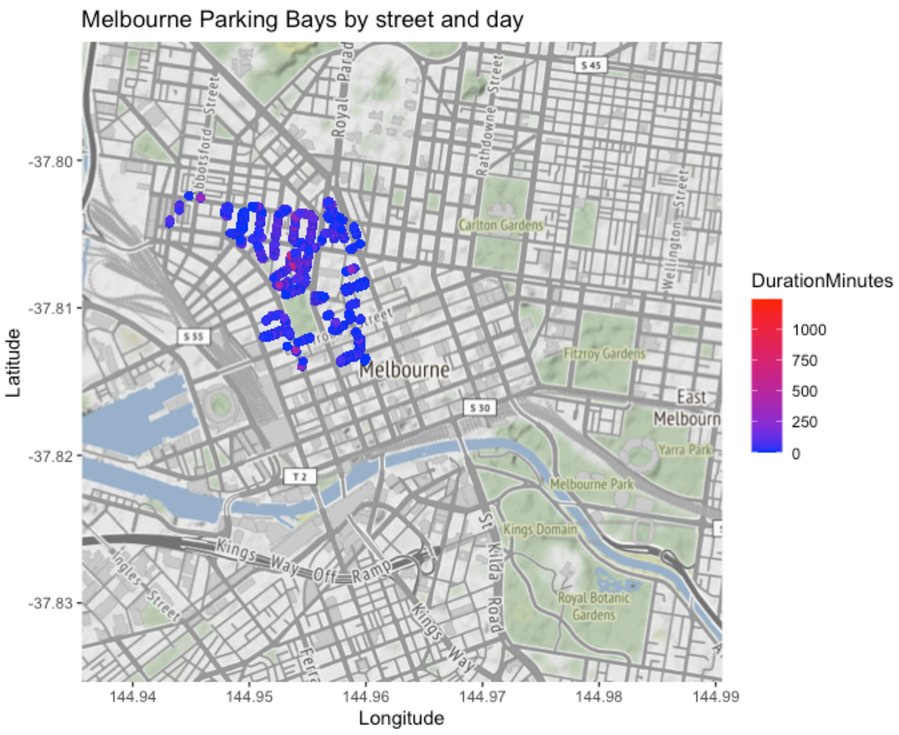


Figure 3.2: Durations for parking bays, in the first week of January 2019, North Melbourne.

3.1.3 Train-test Split

Given that Kriging models take a long time to run with simply 10,000 observations (Veronesi 2015), we can further subset our final dataset by randomly sampling 10,000 rows. We ensure all operations from here onwards are carried out on this dataset, termed 'sub'. The 'sub' dataset is further divided into a testing and training sample using a 75-25% split. Hence, the training sample consists of 7500 observations while the testing sample consists of 2500 observations. All variogram modelling shall be carried out on 'sub_train' and all predictions shall be made on 'sub_test'.

3.1.4 Space-time Indexed Data Frame

A Space-time Indexed Data Frame ('STIDF') consists of measurements observed at certain locations across time. Each measurement is indexed by a spatial location and a temporal location (Wikle et al. 2019). Therefore, an STIDF takes into account three components; 1) a spatial component, 2) a time component and 3) the measurements themselves (Veronesi 2015). The spatial locations are given as coordinates for each parking bay. The temporal locations are given as departure times for each parking instance, averaged across all parking spots within a bay. The measurements themselves are parking bay durations (in minutes).

The durations, departure times and coordinate locations are contained in 'sub' and its ensuing datasets; 'sub_train' and 'sub_test'. Using the *gstat* package in R, we convert these datasets into STIDF objects using the *gstat* package. Firstly, we create a UTM (Universal Transverse Mercator) object consisting of all information from 'sub', called 'durations.UTM'. We specify the latitude and longitude as coordinates. To further format our spatial locations, we convert the coordinates into Mercator Projections such that distances between observations across space, are calculated in metres. We use the 'durations.UTM' dataset to specify each component of the STIDF object. Firstly, the spatial component, 'durationsSP', consists of the coordinate indexes. Secondly, the time component, 'durationsTM', consists of the departure times (formatted as date-time objects). Lastly, the measurement component, ‘durationsDF’, consists of duration figures. By passing ‘durationsSP’, ‘durationsTM’ and ‘durationsDF’ through the STIDF function, we create ‘sub_STIDF’. For all spatio-temporal operations prior to modelling, we shall use ‘sub_STIDF’, while the predictions shall be made on ‘sub_test_STIDF’. ‘sub_test’ goes through the same operations in the creation of ‘sub_test_STIDF’, as ‘sub_train’ did in the creation of ‘sub_STIDF’.

3.2 Empirical Variogram Construction

In order to understand spatio-temporal dependence and variability within our distribution for parking durations, we can construct a variogram. A spatio-temporal variogram maps out the difference in durations between bays that are *h* metres apart in space and τ periods apart in time (Wikle et al. 2019). The equation for a spatio-temporal variogram provided by Wikle et al. (2019) is given below.

$$\widehat{\gamma}_z(\mathbf{h};\tau) = \frac{1}{|N_{\mathbf{s}}(\mathbf{h})|}\frac{1}{|N_t(\tau)|}\sum_{\mathbf{s}_i,\mathbf{s}_k \in N_{\mathbf{s}}(\mathbf{h})}\;\sum_{t_j,t_\ell \in N_t(\tau)} (Z(\mathbf{s}_i;t_j) - Z(\mathbf{s}_k;t_\ell))^2, \qquad (2.8)$$

Hence, we are finding the average difference in durations between pairs of parking bays that are *h* metres apart from one and other in space, and τ periods apart in time. The average difference in durations for all bays that are 100 metres apart, and 1 time period apart from one and other, represents the variogram value associated with a distance of 100m on the spatial axis and a time period of 1 on the temporal axis. These calculations are then repeated for all distance values of *h,* and time period values of τ, to create an empirical variogram. The main purpose of the variogram is to observe the similarity between parking bay durations based on where the bay is located and what time of the day the duration had been recorded.

The construction of a variogram in R, is straightforward using to the gstat package (Pebesma 2021). Particularly, the STIDF object needs to be passed through the variogramST() function. Firstly, we specify our Z values to be the durations of parking bays. Secondly, we specify the sample as ‘sub_STIDF’. Lastly, we specify the time unit (‘tunit’) as “hours”. Hence, τ is represented as hours (in the default of 30-minute increments), and h is represented as metres. It should also be noted that the average difference in durations (i.e., the difference in Z values) for each distance (in space and time), is denoted by γ (‘Gamma’) within a variograms. Once the variogram has been made, three kinds of visualisations can be used to understand the structure; (1) a scatter and line plot, (2) a heatmap and (3) a wireframe. A variogram’s structure is determined by three main parameters; the sill, nugget and range. The interpretation of these parameters as per the work of Gräler (2014) is as follows.

3.2.1 Nugget

If the initial value of h is 100 metres, γ values are calculated for parking bays that are 100 metres apart and higher. However, it is highly likely that we have significant γ values for data points that are closer than 100m. The γ value associated with the lowest observable distance (below the lowest h value), is called the nugget. Specifically, it is the y-axis intercept observable in the scatter and line plot for the empirical variogram. Formally, it is the minimum difference in duration between closest parking bays, that do not have their distance accounted for in h.

3.2.2 Sill

The sill is the variance for our distribution for γ. Simply put, it is the variance of our sample. Graphically, it is the γ value (on the y-axis of the scatter and line plot) where stability amongst γ is achieved. Up until it stabilises at the sill, the γ value is expected to increase gradually due to covariability in our data. At the sill however, the covariability is simply the variance of all γ values, i.e., sample variance, thus causing stability.

3.2.3 Range

The range is the distance value h at which sill is achieved. Hence, it is the distance in metres (on the x-axis of the scatter and line plot) at which the γ value begins stabilising.

3.3 Variogram Modelling

3.3.1 Separable

The overall covariability is calculated by separately taking into account the spatial and temporal components. The model has a straightforward interpretation however, the intuition behind it is questionable in certain applications (Veronesi 2015). For example, given that γ values are dependent on parking durations, that vary depending on both time and space, it may be advisable for us to use models where γ values are determined by the spatial and temporal components together, not separately. The covariance structure of the separable model is as follows.

$$\boxed{C_{sep}(h, \tau) = C_s(h) \,.\, C_t(\tau)}$$

3.3.2 Product Sum

Despite being similar to the separable model, the product sum model does not assume separability between the spatial and temporal covariability (Veronesi 2015). The covariance structure of the model is given as follows.

$$\boxed{C_{prod}(h, \tau) = k \,.\, C_s(h) \,.\, C_t(\tau) + C_s(h) + C_t(\tau)}$$

3.3.3 Metric

The metric model takes into account a joint variogram structure with single values for the nugget, sill and range. In addition, the model also utilises a parameter k to account for spatial-temporal anisotropy (Veronesi 2015). The model equation is given as follows.

$$\boxed{C_m(h, \tau) = C_{joint}(\sqrt{h^2 + (k \,.\, \tau)^2})}$$

Parameter *k* ensures that distances in space and time are accounted for in the same units. Hence, instead of examining gaps in time as hours, we can examine them as metres, such that they can directly be compared to spatial distances (Gräler 2014).

3.3.4 Sum Metric

The sum metric model includes not only the joint component as in the metric model, but also separate spatial and temporal components. Once again, we account for spatio-temporal anisotropy by specifying the parameter k. This is the first model of all aforementioned ones that allows us to specify parameters for all three covariance structures i.e., spatial, temporal and joint (Veronesi 2015). Hence, we can imagine this model to have maximum flexibility when compared to previous ones. The equation for the sum metric model is as follows.

$$\boxed{C_m(h, \tau) = C_s(h) + C_t(\tau) + C_{joint}(\sqrt{h^2 + (k \,.\, \tau)^2})}$$

3.3.5 Simple Sum Metric

The simple sum metric model is similar to the sum metric model with one key difference; we specify a single nugget and not individual nuggets for each component as we had done so previously. Both; the simple sum metric and the sum metric model, yield similar results. To initialise the model, we would have to specify the sill and range for the spatial, temporal and joint component. We would also have to specify a single nugget value for the overall model in addition to the individual component's nuggets (Veronesi 2015).

Each variogram model requires us to specify three variogram structures; a spatial variogram, a temporal variogram and a joint variogram (Veronesi 2015). The structures together determine the model variogram shape. Hence, we would have a separate nugget, sill and range value for each component.

3.3.6 Hyperparameter Fine-tuning

To carry out hyperparameter fine-tuning, we run a nested for-loop for each individual model-type. Therefore, we have 5 nested loops. Each loop, over a range of parameter values, finds the Mean Squared Error (MSE) of the model when fitted onto the training data, and stores this information in a data frame. The parameter values that give us the lowest MSE are selected as parameters for the best model, for any given model type. Hence, the 5 for-loops give us 5 best models – 1 for each model type.

To specify the range of each parameter value within a model's loop, we simply observe the empirical variogram of our training data. If for instance, we observe the empirical variogram's nugget to be 0.4, we would specify the range of nugget values to be between 0.1 and 0.6 in increments of 0.1. We use similar logic for all parameter ranges within each loop. The approximate parameter specifications can be found in *table 3.1* provided below.

Component	*Parameter*	*Inferred Value*
Space	Nugget	0.4
	Sill	1.1
	Range	350
Time	Nugget	0.4
	Sill	1.1
	Range	10
Joint	Nugget	0.5
	Sill	1.1
	Range	350

Table 3.1: approximate parameter specifications inferred by evaluating figure 4.1

In addition to specifying the range, nugget and sill, the variogram structure of each component needs to be specified, within each model. Of the various options in gstat (Gaussian, Spherical, Matrix, Exponential etc.), the most commonly used variogram structure is the Spherical variogram (Pebesma 2021). The empirical variogram reveals that for each lag, we have what seems to be a Spherical variogram shape. Hence, for the spatial and temporal components we shall specify Spherical variogram structures. For the joint component on the other hand, we specify Matrix variogram structures, as Veronesi did in his blog (Veronesi 2015).

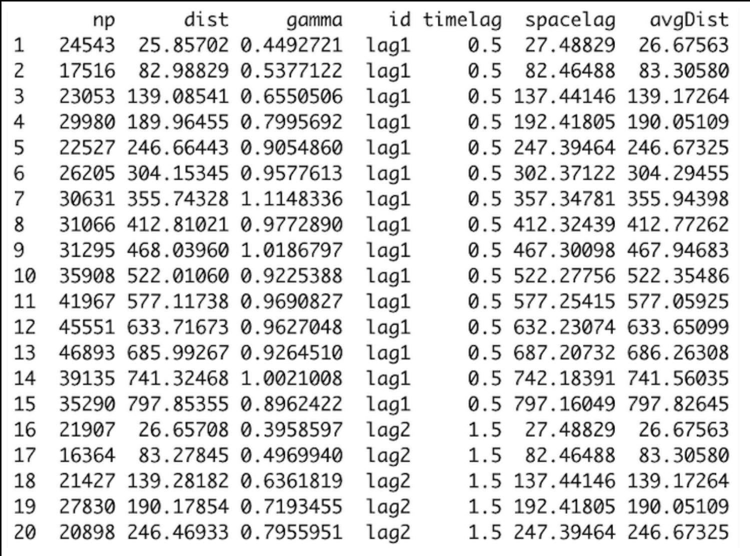
3.4 Model Comparison

The 5 best models (1 for each model-type) can then be compared visually, and by using MSE values. The nested for-loops help us find the lowest-MSE models. Visually, we would prefer to use the model that closely represents the empirical (sample) variogram. Wireframe variograms of fitted models will be utilised to make visual comparisons.

4. Results

4.1 Empirical Variogram Results

The variogram output, consisting of the data used in the variogram visualisations, can be found in *table 4.1* below. They exhibit the gamma values for each distance neighbourhood in space and time. For instance, the first row of the variogram output consists of 24,543 pairs of parking bays with an average distance of 26.67563 metres between them. The distance in time between these points is 0.5 hours, i.e., 30 minutes, and the average difference in standardised duration between them, is approximately 0.4492721. The gamma values for observations that are even further apart in space and time, can be examined.



```
      np      dist     gamma   id timelag  spacelag   avgDist
1  24543  25.85702 0.4492721 lag1     0.5  27.48829  26.67563
2  17516  82.98829 0.5377122 lag1     0.5  82.46488  83.30580
3  23053 139.08541 0.6550506 lag1     0.5 137.44146 139.17264
4  29980 189.96455 0.7995692 lag1     0.5 192.41805 190.05109
5  22527 246.66443 0.9054860 lag1     0.5 247.39464 246.67325
6  26205 304.15345 0.9577613 lag1     0.5 302.37122 304.29455
7  30631 355.74328 1.1148336 lag1     0.5 357.34781 355.94398
8  31066 412.81021 0.9772890 lag1     0.5 412.32439 412.77262
9  31295 468.03960 1.0186797 lag1     0.5 467.30098 467.94683
10 35908 522.01060 0.9225388 lag1     0.5 522.27756 522.35486
11 41967 577.11738 0.9690827 lag1     0.5 577.25415 577.05925
12 45551 633.71673 0.9627048 lag1     0.5 632.23074 633.65099
13 46893 685.99267 0.9264510 lag1     0.5 687.20732 686.26308
14 39135 741.32468 1.0021008 lag1     0.5 742.18391 741.56035
15 35290 797.85355 0.8962422 lag1     0.5 797.16049 797.82645
16 21907  26.65708 0.3958597 lag2     1.5  27.48829  26.67563
17 16364  83.27845 0.4969940 lag2     1.5  82.46488  83.30580
18 21427 139.28182 0.6361819 lag2     1.5 137.44146 139.17264
19 27830 190.17854 0.7193455 lag2     1.5 192.41805 190.05109
20 20898 246.46933 0.7955951 lag2     1.5 247.39464 246.67325
```

Table 4.1: Variogram output

When we visualise these data points in a variogram, we can observe the overall similarities in standardised parking durations as we move through space and time. The scatter and line plot provided in *figure 4.1*, indicates that the difference in standardised durations between any two parking bays, seems to increase as the spatial distance between the parking bays increase. Furthermore, this similarity tapers off, as the time gaps between pairs of recorded durations increase. Parking bays that are closer to one and other have more similar durations than bays that are further apart. This relationship is observed for measurements recorded 0.5 minutes apart, 1 hour apart, 1.5 hours apart, so on and so forth. The colours themselves represent the time lags in terms of hours in 30-minute increments. For instance, lag 15 includes pairs of measurements that are 7.5 hours apart. Hence, we can see a similar line shape being taken up at every lag. The relationship is more gradual amongst pairs of measurements that are 30 mins apart, as opposed to measurements that are 7.5 hours apart.

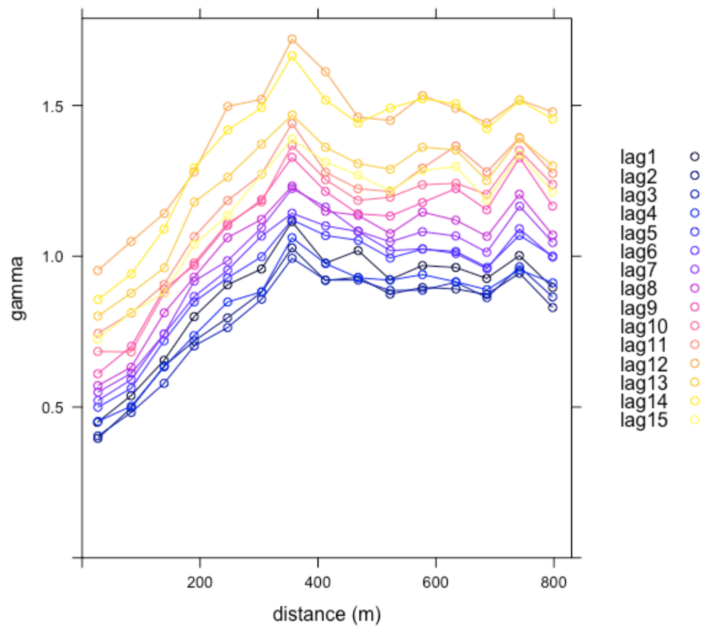


Figure 4.1: Scatter and line plot for sample variogram

The heatmap and the wireframe plot provided in *figure 4.2 and 4.3* exhibit the same phenomenon. We can see how the gamma values are higher (yellow) where pairs of measurements have a greater distance, in space and time, between them. Likewise, gamma values are lower (dark blue) where pairs have a lower distance between them in space and time.

Once again reemphasising the idea that parking bays that are closer in space and time, have similar duration figures.

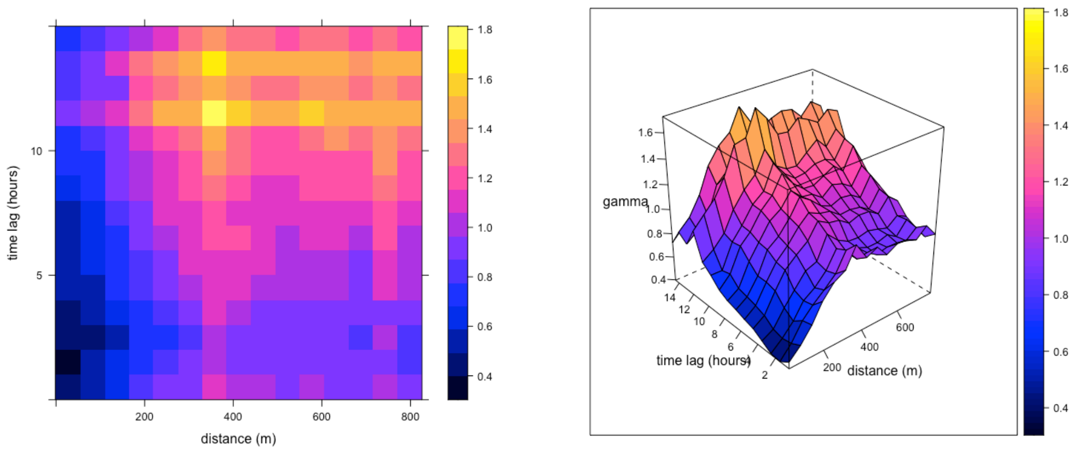


Figure 4.2 and figure 4.3: Heatmap (left) and Wireframe plot (right) for sample variogram

The scatter and line plot for the variogram is particularly useful in helping us identify the parameter values needed for the modelling phase. Note, we do not need specific values given that we intend on using for-loops for each model over a range of potential values for each parameter. Instead, we use approximate values.

4.1.1 Nugget

At the distance of 26m, the gamma value is 0.449. Particularly, the average difference in standardised durations for pairs of points that are less than 83m apart in space, is 0.449.

4.1.2 Sill

The sill of the sample is achieved at a gamma value of 1. We can hypothesise that our data has a sample variance of 1. This is expected given that we had standardised our duration figures to begin with. Prior to reaching the sill, the model has spatio-temporal covariance that gradually converges to the sample variance, where gamma stabilises.

4.1.3 Range

The range of our variogram is approximately 350m. From 350m spatial-distance onwards, the gamma values stabilise and fluctuate around the sill. This range is applicable across all time lags.

4.2 Variogram Modelling & Hyperparameter Fine-tuning

4.2.1 Separable

The separable for-loop runs over 1,225 iterations. For the space component, the loop evaluates range values between 300m to 400m (in increments of 25m), and nugget values between 0 to 0.6 (in increments of 0.1). For the time component, the loop evaluates range values between 7-time gaps to 11-time gaps (in increments of 1), and nugget values between 0 to 0.6 (in increments of 1). Off all iterations within the loop, we achieve the lowest MSE when we use; a spatial component with a range of 325m and a nugget of 0.4, and a temporal component with a range of 8 (total of 4 hours) and a nugget of 0.2. When fitted onto our data, the MSE itself is 0.01546163, and the resulting variogram is provided in *figure 4.4*.

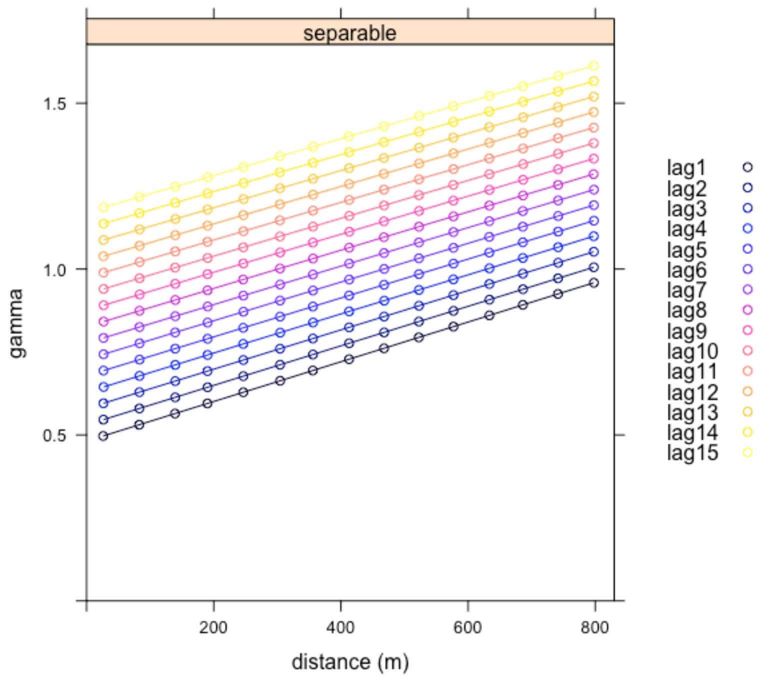


Figure 4.4: Lowest-MSE Separable model fitted onto sample variogram

4.2.2 Product Sum

The product sum for-loop runs over 1,225 iterations. The specifications for the spatial and temporal component remain unchanged from that of the separable model. We specify the spatio-temporal anisotropy parameter, k 0.1, arbitrarily. The specification of k shall not affect our results (as long as k > 0) too much given that we are still using an automatic fit in each iteration (Veronesi 2015). Off all iterations within the loop, we achieve the lowest MSE when we use; a spatial component with a range of 325m and a nugget of 0.4, and a temporal component with a range of 8 (total of 4 hours) and a nugget of 0.2. When fitted onto our data, the MSE itself is 0.008729154, and the resulting variogram is provided in *figure 4.5*.

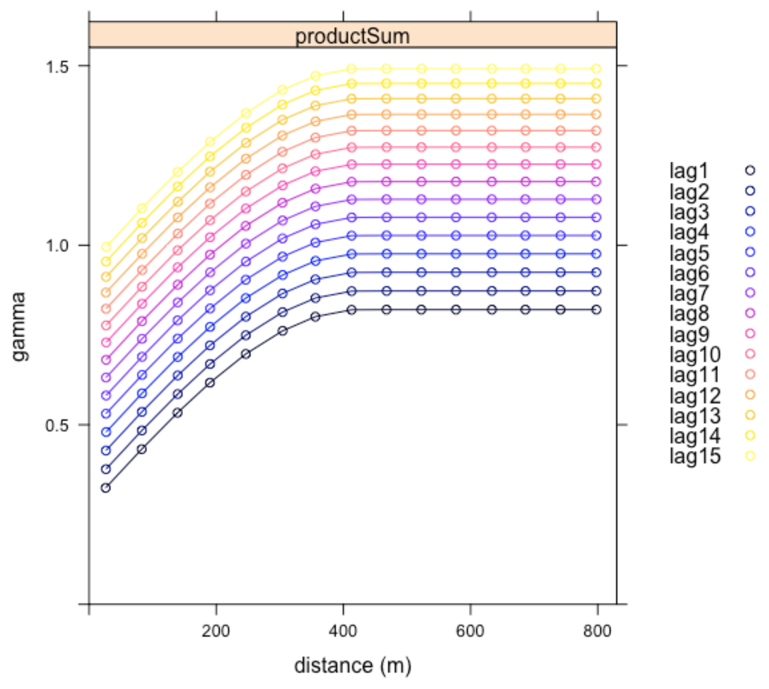


Figure 4.5: Lowest-MSE Product Sum model fitted onto sample variogram

4.2.3 Metric

The metric for-loop runs over 35 iterations. For the only component – the joint component – the loop evaluates range values between 300m to 400m (in increments of 25m), and nugget values between 0 to 0.6 (in increments of 0.1). Furthermore, we specify the spatio-temporal anisotropy parameter, stAni as 100m and the sill as 1.5. Off all iterations within the loop, we achieve the lowest MSE when we use; a joint component with a range of 375m and a nugget of 0.2. When fitted onto our data, the MSE itself is 0.02069102, and the resulting variogram is provided in *figure 4.6*.

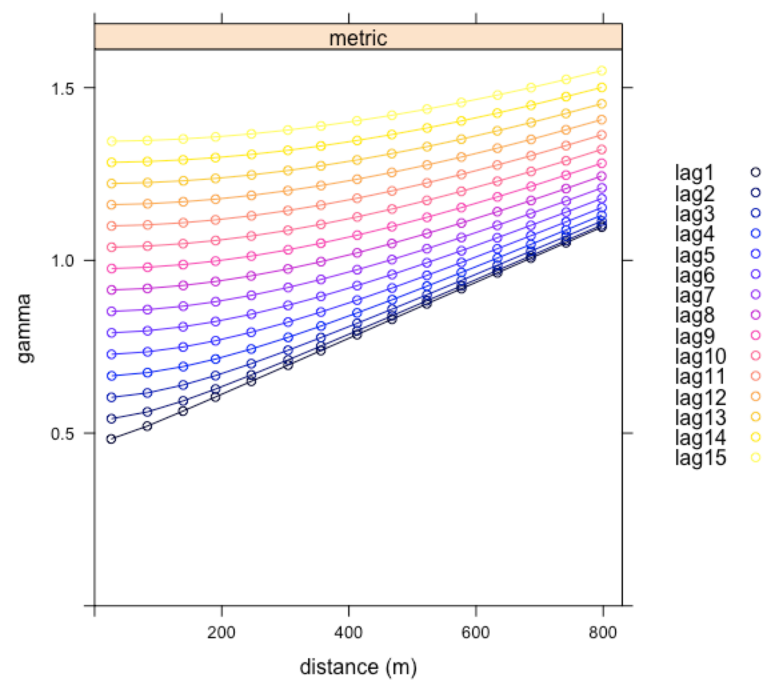


Figure 4.6: Lowest-MSE Metric model fitted onto sample variogram

4.2.4 Sum Metric

The sum metric for-loop runs over 42 iterations. For the joint component, the loop evaluates range values between 300m to 425m (in increments of 25m), and nugget values between 0 to 0.6 (in increments of 0.1). Furthermore, we specify the spatio-temporal anisotropy parameter, stAni as 100m. We can also specify ranges of values for the spatial and temporal component's nugget, range and sill. However, the for loop takes long to run when we do so. Hence, for the space component, we fix the range at 325m, and the nugget at 0.4. For the time component, we fix the range at 8 (4 hours in total), and the nugget at 0.2. In the product sum model, these specifications for the spatial and temporal component seemed to yield the lowest MSE, hence, we shall stick to them. We achieve the lowest MSE when we use a joint component with a range of 375m and a nugget of 0.1. When fitted onto our data, the MSE itself is 0.008729157, and the resulting variogram is provided in *figure 4.7*.

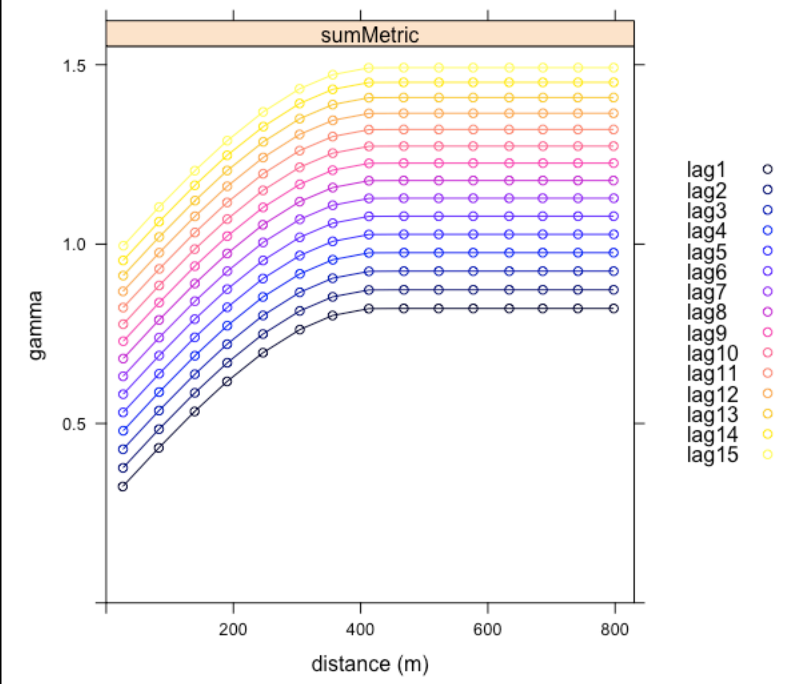


Figure 4.7: Lowest-MSE Sum Metric model fitted onto sample variogram.

4.2.5 Simple Sum Metric

The simple sum metric for-loop runs over 42 iterations. For the joint component, the loop evaluates range values between 300m to 450m (in increments of 25m), and a nugget value of 0.1. Furthermore, we specify the spatio-temporal anisotropy parameter, stAni as 100m. As was the case with the sum metric for-loop, we do not specify a range of parameter values for the nugget, range and sill of the spatial and temporal components due to the size and run-time for such for-loops. Hence, for the space component, we fix the range at 325m, and the nugget at 0. For the time component, we fix the range at 8 (4 hours in total), and the nugget at 0. Lastly, we specify a range of values for the overall nugget of the model, that is not included in any of the three components. The for-loop will evaluate nugget values between 0 and 0.6 (in increments of 0.1). Off all iterations within the loop, we achieve the lowest MSE when we use; a joint component with a range of 325m and an overall nugget value of 0.4. When fitted onto our data, the MSE itself is 0.008781948, and the resulting variogram is provided in *figure 4.8*.

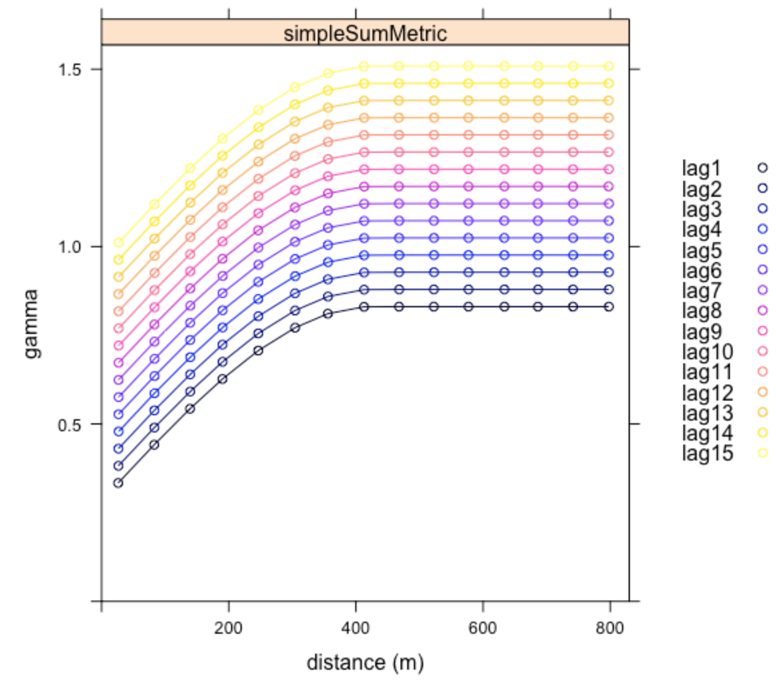


Figure 4.8: Lowest-MSE Simple Sum Metric model fitted onto sample variogram

4.3 Model Comparison

A visual comparison of the 5-best models can be found in *figure 4.9*. We can see how each model approximates the spatio-temporal relationship of our sample data. However, for the separable and metric models, it seems as though we do not have any spatial and temporal stability i.e., any discernible range or sill values to go off of. The variograms for the product sum, sum metric and simple sum metric models seem to better represent the sample variogram.

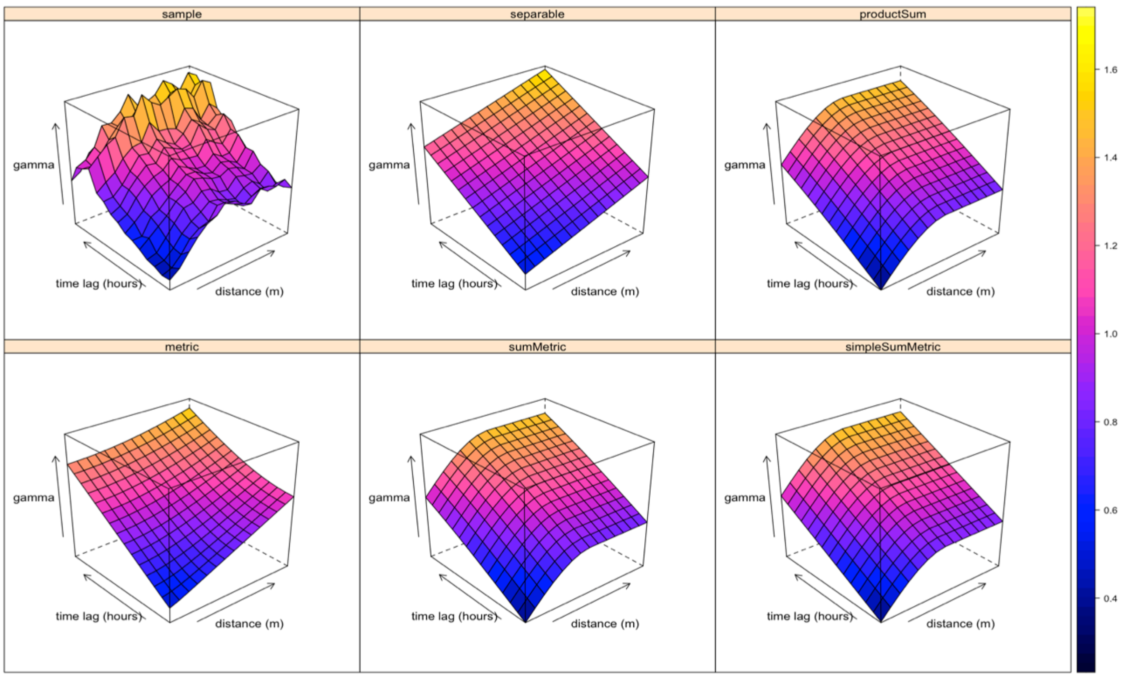


Figure 4.9: Variograms for each lowest-MSE model and the sample (right upper corner)

From the product sum, sum metric and simple sum metric models, the model with the lowest MSE is the product sum model. Hence, we use the product sum model to predict durations of parking bays.

5. Predictions

The accuracy of a prediction can be evaluated by finding the estimation variance between an actual value and its predicted equivalent. In Kriging, the function for the estimation variance takes into account the covariance structure amongst known observations, and the covariance structure between known and unknown observations, within the neighbourhood of the testing-observation. Hence, by using the First Order Condition, we can find weights associated with the local minimum of the estimation variance function. These weights are expected to minimise the variance between actual values and predicted ones (Pyrcz 2018). Once these weights have been found by the algorithm, Kriging utilises them on duration values of known observations, in order to predict the duration of unknown observations.

5.1 Prediction Methodology

To create a Kriging model, we simply use the KrigeST function (provided in gstat) (Pebsema 2021 & Veronesi 2015). To run the function, we specify the dependent variable as durations and specify the data as 'sub_STIDF'. Additionally, we need to specify the model type as the product sum model. Lastly, we specify the testing data as 'sub_test_STIDF'.

5.2 Prediction Results

The density plot provided in *figure 5.1* exhibits the density curves for the original testing data and the predicted testing data.

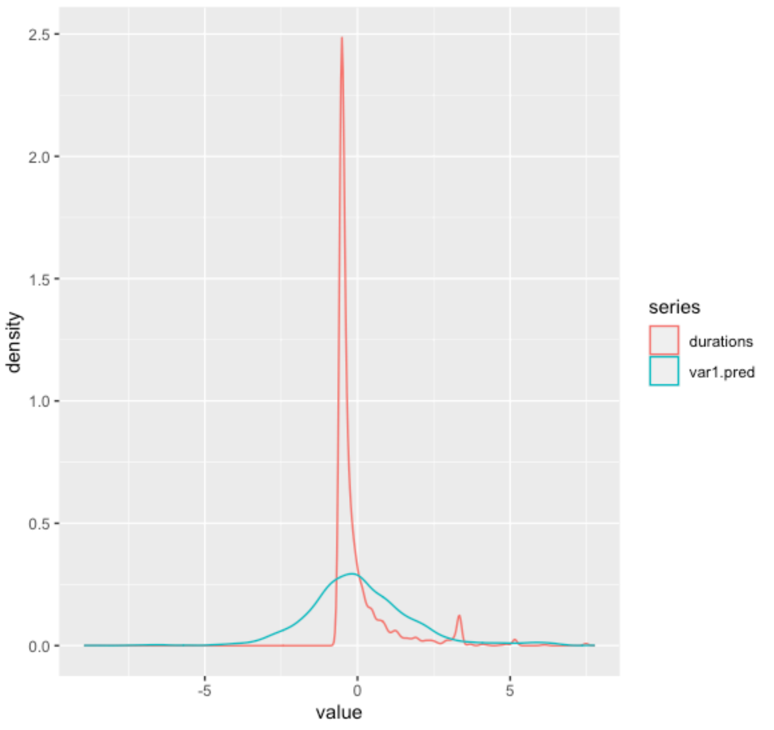
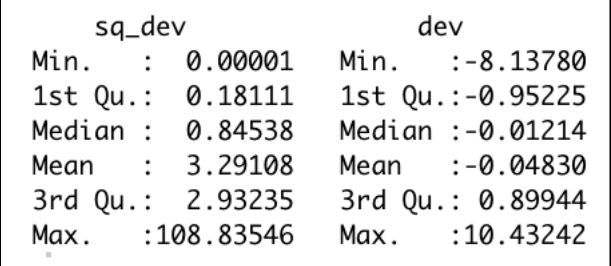


Figure 5.1 – density curves for actual durations and predicted durations

The predictions seem to underpredict a large number of observations. Although we do not have actual standardised duration values below 0, our predictions do. In order to summarise the deviations between both, we simply run a summary function on a dataset consisting of actual durations, predicted durations, and their differences. The summary output can be found in Table 5.1 below.



```
     sq_dev                 dev
 Min.   :  0.00001   Min.   :-8.13780
 1st Qu.:  0.18111   1st Qu.:-0.95225
 Median :  0.84538   Median :-0.01214
 Mean   :  3.29108   Mean   :-0.04830
 3rd Qu.:  2.93235   3rd Qu.: 0.89944
 Max.   :108.83546   Max.   :10.43242
```

Table 5.1 – summary of squared deviations and deviations

The summary output indicates that on average, our predictions are off by -0.04830. In other words, predicted durations exceed actual durations, on average by 0.04830. Perhaps our model's inadequacy in predicting durations accurately, is due to the lack of other independents variables. The inclusion of additional features that affect parking behaviour, such as weather forecasts, may be useful in improving such predictions. Furthermore, hyperparameter fine-tuning using larger for-loop may enable us to find better parameters.

6. Outlier Detection

The rationale behind predicting durations is to identify observations that do not fit the usual spatio-temporal behaviour of the sample. The kriging predictions determine what we can expect the duration to be, for a given parking bay. We hypothesise that large scale deviations between expected durations and actual durations are indicative of anomalous behaviour. Hence, arbitrarily speaking, if our model predicts that a given parking bay should have an average duration of 10 minutes, while the actual average duration is 50 minutes, we can term the parking bay's behaviour as anomalous, as it vastly varies from behaviour we can expect.

In order to achieve this, we simply look at outliers within the distribution for deviations between actual values and predicted values, using the z-score method. This method is perhaps most suited, given that our data is univariate in nature (Anil 2019). When we use the sum metric model, we can observe several outliers within this distribution. It is to be noted that these outliers can be termed as 'under-parking' or 'over-parking' depending on which tail of the distribution we are observing.

If we then retrace our steps, we can identify the latitude and longitude (or space indexes) associated with these outliers and visualise them over our spatial domain, as is examinable in *figure 6.1.*

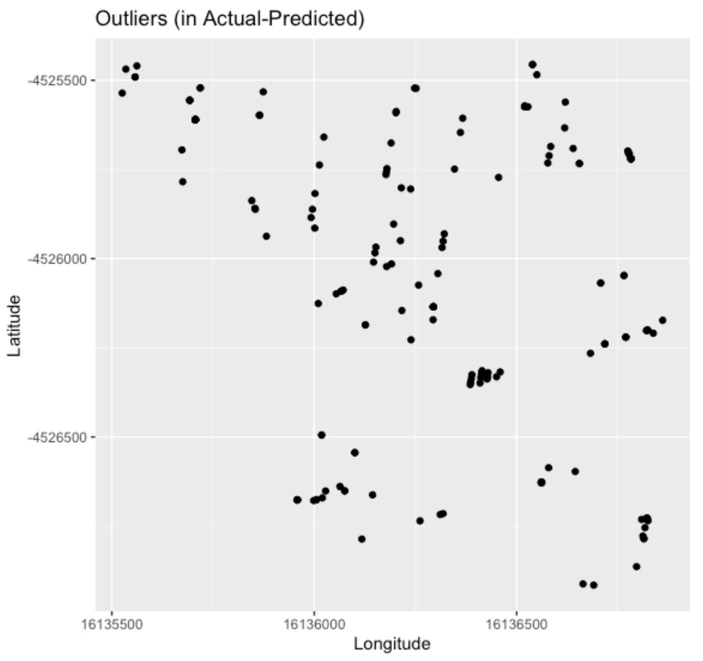


Figure 6.1: Anomalous parking bay durations amongst predictions

If a group of parking bays in close proximity to each other are detected as outliers, they can be clustered together and identified as the locality of an event. The specifications of a cluster would have to be determined, however. Particularly, we would need to identify the optimal number of parking bay outliers that should fall under a cluster for it to be considered an event. We can also further classify events based on the number of parking bays being affected. For instance, is this a large-scale event or a small-scale event? An event can be classified even further to represent its type – 'underparking' or 'overparking'. For instance, are cars being asked to vacate the premises due to the event, or are they being prevented from leaving due to emergency closure of streets? Hence, the description of an event can be established through multiple lenses.

Given computational limitations, Kriging predictions could only be made on durations of parking bays, as opposed to parking spots. However, during an event, we can expect multiple spots in a given bay to be affected by the event. Hence, there would have been more outliers to cluster, had we used parking spots.

7. Discussions

The study had aimed to make spatio-temporal predictions on IoT parking data and identify anomalous behaviour. Most research on parking sensor data, aims to predict parking durations to alleviate issues of traffic congestion and navigation. However, only the works of Zheng at al. (2014, p. 1), Piovesan et al. (2015, p. 192) and Lu and Liao (2018, p. 500) had evaluated its application in identifying anomalous parking behaviour. With the hopes of contributing to this field of research, we had examined parking sensor data in the City of Melbourne for the year 2019.

The spatio-temporal variability between parking bay durations had been examined, followed by which relevant Kriging models had been utilised to predict said variability. The empirical variogram had enabled us to observe that; amongst all parking bay duration dispersed across our spatio-temporal domain, those that are closer in space and time are more similar than those that are further apart. By using the product sum Kriging model, we were able to model this behaviour and achieve an MSE of only 0.008729154. The model seems to also approximate the empirical variogram shape quite well, through a visual comparison. On having fitted the product sum model on our sample data, we were able to make Kriging predictions on the durations within our testing data.

Once the predictions had been made, actual values and predicted values had been differenced to find deviations between what we expect, and what we observe. Outliers within the distribution for these deviations had been found using the z-score method. Using the index numbers of these outliers, we were able to visualise them on our spatial domain. Our study assumes that these outliers are indicators of anomalous parking behaviour as they represent duration observations that exceed expectations due to 'under-parking' or 'over-parking'.

Further research needs to be conducted in the usage of clustering methodology to group together outliers based on spatio-temporal closeness. An event would first have to be defined, and classified into various types, in order to determine the most appropriate clustering methodology.

Due to lack of relevant literature that utilises spatio-temporal statistics, there were several limitations encountered during the study. Analysing spatial variability and time-series for durations would have helped us gain an understanding of how durations behave across space and time, separately. However, given that both domains cause durations to fluctuate in unison, using a spatio-temporal approach may prove to be more valuable. Literature in spatio-temporal statistics is particularly lacking when it comes to carrying out cross-validation and hyperparameter fine-tuning. Hence, for-loops had been arbitrarily set to evaluate performance over a range of parameters. No cross validation had been carried out, given that the testing and training datasets had to be separately formatted into STIDF objects. Furthermore, spatio-temporal operations can only be conducted for a limited number of observations. Hence, most of the allocated project time had been spent in trying to subset the data appropriately. This had been a particularly difficult task given the seasonality in the temporal variability, and the spatial variability of durations figures. Using a larger number of observations (individual parking spots instead of parking bays) would have perhaps improved our model's performance. Lastly, predictions may be improved upon through the usage of additional variables. For instance, by accounting for the weather, live traffic conditions etc., we could have derived predictions that were more representative of actual behaviour.

References

Anil, D 2019, 'Module 6: Outliers', PowerPoint slides, MATH2349, RMIT University, Melbourne.

City of Melbourne Data Centre 2020, *On-street Car Parking Sensor Data,* data file, City of Melbourne, Melbourne, viewed 17 April 2021,

<https://data.melbourne.vic.gov.au/Transport/On-street-Car-Parking-Sensor-Data-2019/7pgd-bdf2>.

Geng, Y & Cassandra, C 2013, ‘New “Smart Parking” system based on resource allocation and reservations,’ *IEEE Trans. Intelligence. Transport System,* Vol. 14, pp. 1129-1139.

Gräler, B 2014, ‘Spatial temporal Kriging’ PowerPoint slides, GEOSTAT, University of Muenster, Muenster.

Gupta, A, Sharma, V, Ruparam, NK, Jain, S, Alhammad & Ripon, MK 2014, ‘Integrating pervasive computing InfoStations and swarm intelligence to design intelligent context-aware parking space location mechanism,’ *Proceedings of the International Conference on Advances in Computing Communications and Informatics (ICACCI)*, pp. 24-27.

Pebesma, E 2021, ‘The meuse dataset: a brief tutorial for the gstat R package’, pp. 1-13.

Pebesma, E 2021, ‘Handling and analysing spatial-temporal Data’, CRAN Task View, viewed 17 April 2021, < https://cran.r-project.org/web/views/SpatioTemporal.html>.

Piovesan, N, Turi, L, Toigo, B & Martinez, M 2016, ‘Data analytics for smart parking applications,’ *Sensors*, vol. 16, pp. 10.

Pyrcz, MJ 2018, ‘Lec 12b: Kriging’, PGE337, University of Texas, Austin.

Salim 2021, *Sensors take headache out of summer beach parking*, RMIT, viewed 17 Apr 2021, <https://www.rmit.edu.au/news/all-news/2019/jan/sensors-summer-beach-parking>.

Veronesi, F 2015, ‘R tutorial for Spatial Statistics’, *Blogger.com*, blog, viewed 20 May 2021, < http://r-video-tutorial.blogspot.com/2015/08/spatio-temporal-kriging-in-r.html>.

Vlahogianni, EI, Kepaptsoglou, K, Tsetsos, V & Karlaftis, MG 2016, ‘A Real-Time Parking Prediction System for Smart Cities’, *Journal of Intelligent Transportation Systems*, vol. 20:2, pp. 192-204.

Wang, H, & Wenbo, H 2011, ‘A reservation-based smart parking system’, *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 690-695.

Wikle, CK, Zammit-Mangion, A & Cressie, N 2019, *Spatio-temporal statistics with R*, Boca Raton, Florida.

Zheng, Y, Rajasegarar, S, Leckie, C & Palaniswami, M 2014, ‘Smart car parking: Temporal clustering and anomaly detection in urban car parking,’ *IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pp. 1-6.

Appendix

```
library(gstat)
library(sp)
library(spacetime)
library(ggplot2)
library(dplyr)
library(tidyr)
library(raster)
library(rgdal)
library(rgeos)
library(magrittr)
library(xts)
library(reshape2)
library(outliers)
library(devtools)
library(ggmap)
library(maptools)
library(broom)



#Importing ready made datasets------------------------------------



january <- read.csv("~/Desktop/Applied Research Project/Parking Sensor Data/subsetted
datasets/January.csv")
bay_locations <- read.csv("~/Desktop/Applied Research Project/Parking Sensor
Data/subsetted datasets/bay_locations.csv")

jan <- january[!january$DurationMinutes < 0,]
jan <- jan %>% dplyr::select(-c(1))

january$BayId <- factor(january$BayId, levels=c(unique(january$BayId)))
jan$DeviceId <- factor(jan$DeviceId,levels=c(unique(jan$DeviceId)))
bay_locations$BayId <- factor(bay_locations$BayId,
levels=c(unique(bay_locations$BayId)))

jan$DurationMinutes <- jan$DurationMinutes %>% as.numeric()

jan$ArrivalTime <- strptime(jan$ArrivalTime,format="%d/%m/%Y %I:%M:%S %p")
jan$DepartureTime <- strptime(jan$DepartureTime,format="%d/%m/%Y %I:%M:%S %p")


## jan_grouped_bays
```

```
jan_grouped_bays <- jan %>% group_by(BayId, DepartureTime) %>% summarise(duration
= mean(DurationMinutes, na.rm=T))
write.table(jan_grouped_bays,
        file = "~/Desktop/Applied Research Project/Parking Sensor Data/subsetted
datasets/jan_grouped_bays.csv",
        sep = ",", row.names = F)


## jan_with_coords

jan_with_coords <- merge(jan_grouped_bays, bay_locations, by="BayId")
jan_with_coords <- jan_with_coords %>% dplyr::select(1,2,3,5,6) #subsetting BayId,
durations, departure time, lat, lon
write.table(jan_with_coords,
        file = "~/Desktop/Applied Research Project/Parking Sensor Data/subsetted
datasets/jan_with_coords.csv",
        sep = ",", row.names = F)




# sub, sub_train, sub_test, sub_train_standardised, sub_test_standardised,
sub_train_standardised_log, sub_test_standardised_log----------------------------




sub <- jan_with_coords[jan_with_coords$lat>=-37.815 & jan_with_coords$lat<=-37.80,]
sub <- sub[sub$lon>=144.94 & sub$lon<=144.96,]
sub <- sub[sub$DepartureTime>=strptime("01/01/2019 12:00:00 am",format="%d/%m/%Y
%I:%M:%S %p") &
         sub$DepartureTime<=strptime("07/01/2019 11:59:00 pm",format="%d/%m/%Y
%I:%M:%S %p") ,]
nrow(sub)

write.table(sub,
        file = "~/Desktop/Applied Research Project/Parking Sensor Data/subsetted
datasets/sub.csv",
        sep = ",", row.names = F)

sub_full <- sort(sample(nrow(sub), nrow(sub)*0.75))
sub_train <- sub[sub_full, ]
sub_test <- sub[-sub_full, ]
nrow(sub_test)+nrow(sub_train)

write.table(sub_train,
        file = "~/Desktop/Applied Research Project/Parking Sensor Data/subsetted
datasets/sub_train.csv",
        sep = ",", row.names = F)
write.table(sub_test,
	file = "~/Desktop/Applied Research Project/Parking Sensor Data/subsetted
datasets/sub_test.csv",
	sep = ",", row.names = F)

sub_train_standardised <- sub_train
sub_train_standardised$duration <- as.numeric(scale(sub_train_standardised$duration))
sub_test_standardised <- sub_test
sub_test_standardised$duration <- as.numeric(scale(sub_test_standardised$duration))

write.table(sub_train_standardised,
	file = "~/Desktop/Applied Research Project/Parking Sensor Data/subsetted
datasets/sub_train_standardised.csv",
	sep = ",", row.names = F)
write.table(sub_test_standardised,
	file = "~/Desktop/Applied Research Project/Parking Sensor Data/subsetted
datasets/sub_test_standardised.csv",
	sep = ",", row.names = F)

sub_train_standardised_log <- sub_train_standardised
sub_train_standardised_log$duration <- (sub_train_standardised_log$duration+0.5481534)
sub_train_standardised_log$duration <- log(sub_train_standardised_log$duration)

sub_test_standardised_log <- sub_test_standardised
sub_test_standardised_log$duration <- (sub_test_standardised_log$duration+0.55)
sub_test_standardised_log$duration <- log(sub_test_standardised_log$duration)

write.table(sub_train_standardised_log,
	file = "~/Desktop/Applied Research Project/Parking Sensor Data/subsetted
datasets/sub_train_standardised_log.csv",
	sep = ",", row.names = F)
write.table(sub_test_standardised_log,
	file = "~/Desktop/Applied Research Project/Parking Sensor Data/subsetted
datasets/sub_test_standardised_log.csv",
	sep = ",", row.names = F)




# following workflow from: http://r-video-tutorial.blogspot.com/




# sub_STIDF creation----------------------------------------------------------------------------




## durations.UTM,	durationsSP, durationsTM, durationsDF,	sub_STIDF creation
```

```
x <- 10000 #respecify this sample size number and run the following chunk again for each
new variogram trial
durations.UTM <- sample_n(sub_train_standardised, x)
coordinates(durations.UTM) = ~lon+lat
projection(durations.UTM)=CRS("+init=epsg:4326")
durations.UTM <- spTransform(durations.UTM, CRS("+init=epsg:3395"))

durationsSP <- SpatialPoints(durations.UTM@coords,CRS("+init=epsg:3395")) #coordinates
dataset
duplicates <- zerodist(durationsSP) #no duplicate values allowed in the values and time
dataset
durationsDF <- data.frame(durations = durations.UTM$duration)#[-duplicates[,2]]
durationsTM <- durations.UTM$DepartureTime#[-duplicates[,2]]

sub_STIDF <- STIDF(durationsSP, durationsTM, data=durationsDF)
stplot(sub_STIDF) #saved as stplot_10000_hours
summary(sub_STIDF)



# Empirical Variogram Construction-------------------------------------------------------------



var_10000_hours <- variogramST(durations~1, data=sub_STIDF, tunit="hours",
assumeRegular = F, na.omit = T)

plot(var_10000_hours, map=F)
plot(var_10000_hours, map=T)
plot(var_10000_hours, wireframe=T, scale=list(arrows=F))
var_10000_hours

var_nonstandard
var_standardised <- var_10000_hours


# Variogram Modelling---------------------------------------------------------------------------



## Separable

range.s = c()
range.t = c()
nugget.s = c()
nugget.t = c()
mse = c()
```

```
for (is in (seq(from=300, to=400, by = 25))){
  for (it in (seq(from=7, to=11, by = 1))){
    for (ns in (seq(from=0.1, to=0.6, by = 0.1))){
      for (nt in (seq(from=0.1, to=0.6, by = 0.1))){
        range.s <- append(range.s, is)
        range.t <- append(range.t, it)
        nugget.s <- append(nugget.s, ns)
        nugget.t <- append(nugget.t, nt)
        mse <- append(mse, attr(fit.StVariogram(var_standardised,method = "L-BFGS-B",
                                  vgmST("separable",
                                        space = vgm(0.5, "Sph", range = is, nugget = ns),
                                        time = vgm(0.5, "Sph", range = it, nugget = nt),
                                        sill = 1.2)), "MSE"))


      }
    }
  }
}
parameter_tuning_standardised_separable <- data.frame(range.s=range.s, range.t=range.t,
                                        nugget.s=nugget.s, nugget.t=nugget.t,
                                        mse=mse)

## ProductSum

range.s = c()
range.t = c()
nugget.s = c()
nugget.t = c()
mse = c()
for (is in (seq(from=300, to=400, by = 25))){
  for (it in (seq(from=7, to=11, by = 1))){
    for (ns in (seq(from=0, to=0.6, by = 0.1))){
      for (nt in (seq(from=0, to=0.6, by = 0.1))){
        range.s <- append(range.s, is)
        range.t <- append(range.t, it)
        nugget.s <- append(nugget.s, ns)
        nugget.t <- append(nugget.t, nt)
        mse <- append(mse, attr(fit.StVariogram(var_standardised,method = "L-BFGS-B",
                                  vgmST("productSum",
                                        space = vgm(0.5, "Sph", range = is, nugget = ns),
                                        time = vgm(0.5, "Sph", range = it, nugget = nt),
                                        k=0.1)), "MSE"))


      }
    }
  }
}
parameter_tuning_standardised_prodsum <- data.frame(range.s=range.s, range.t=range.t,
                                    nugget.s=nugget.s, nugget.t=nugget.t,
                                    mse=mse)

## Metric

range.st = c()
nugget.st = c()
mse = c()
for (ij in (seq(from=300, to=400, by = 25))){
  for (nj in (seq(from=0, to=0.6, by = 0.1))){
    range.st <- append(range.st, ij)
    nugget.st <- append(nugget.st, nj)
    mse <- append(mse, attr(fit.StVariogram(var_standardised, method = "L-BFGS-B",
                                vgmST("metric",
                                      joint = vgm(0.5, "Mat", range = ij, nugget = nj),
                                      sill=1.5, stAni=100)), "MSE"))
  }
}

parameter_tuning_standardised_metric <- data.frame(range.st=range.st,
                                    nugget.st=nugget.st,
                                    mse=mse)


## SumMetric

range.st=c()
nugget.st=c()
mse = c()

for (ij in (seq(from=300, to=425, by = 25))){
  for (nj in (seq(from=0, to=0.6, by = 0.1))){
    range.st <- append(range.st, ij)
    nugget.st <- append(nugget.st, nj)
    mse <- append(mse, attr(fit.StVariogram(var_standardised, method = "L-BFGS-B",
                                vgmST("sumMetric",
                                      space = vgm(0.5, "Sph", range=325, nugget=0.4),
                                      time = vgm(0.5, "Sph", range=8, nugget = 0.2),
                                      joint = vgm(0.1,"Mat", range=ij, nugget=nj), stAni=100)),
"MSE"))

  }
}
parameter_tuning_standardised_summetric <- data.frame(range.st=range.st,
nugget.st=nugget.st,
                                      mse=mse)

## SimpleSumMetric

range.st=c()
```

```
nugget.st=c()
mse = c()
for (ij in (seq(from=300, to=450, by = 25))){
  for (nj in (seq(from=0.1, to=0.6, by = 0.1))){
    range.st <- append(range.st, ij)
    nugget.st <- append(nugget.st, nj)
    mse <- append(mse, attr(fit.StVariogram(var_standardised, method = "L-BFGS-B",
                                    vgmST("simpleSumMetric",
                                          space = vgm(0.5, "Sph", range=325, nugget=0.1),
                                          time = vgm(0.5, "Sph", range=8, nugget = 0.1),
                                          joint = vgm(0.1,"Mat", range=ij, nugget=0.1),
                                          nugget=nj,stAni=100)), "MSE"))
  }
}

parameter_tuning_standardised_simplesummetric <- data.frame(range.st=range.st,
nugget.st=nugget.st,
                                                  mse=mse)

stani.list

# Hyperparameter Fine-tuning--------------------------------------------------------------------



## Separable

parameter_tuning_standardised_separable
parameter_tuning_standardised_separable %>%
filter(mse==min(parameter_tuning_standardised_separable$mse))
separable <- vgmST("separable",
                space = vgm(0.5, "Sph", range=325, nugget=0.4),
                time = vgm(0.5, "Sph", range=8, nugget=0.2),sill=1.5)
plot(var_standardised,separable,map=F)

separable_Vgm <- fit.StVariogram(var_standardised, separable,method = "L-BFGS-B")
plot(var_standardised,separable_Vgm, wireframe=T)
plot(var_standardised,separable_Vgm, map=F)

attr(separable_Vgm, "MSE")

## ProdSum

parameter_tuning_standardised_prodsum
parameter_tuning_standardised_prodsum %>%
filter(mse==min(parameter_tuning_standardised_prodsum$mse))
prodSumModel <- vgmST("productSum",
                  space = vgm(0.5, "Sph", range=325, nugget=0.4),
                  time = vgm(0.5, "Sph", range=8, nugget = 0.2), k=0.1)
```

```
plot(var_standardised,prodSumModel,map=F)

prodSumModel_Vgm <- fit.StVariogram(var_standardised, prodSumModel,method = "L-
BFGS-B")
plot(var_standardised,prodSumModel_Vgm, wireframe=T)
plot(var_standardised,prodSumModel_Vgm, map=F)

## Metric

parameter_tuning_standardised_metric
parameter_tuning_standardised_metric %>%
filter(mse==min(parameter_tuning_standardised_metric$mse))
metric <- vgmST("metric",
             joint = vgm(0.5,"Mat", range=375, nugget=0.2),
             sill=1.5, stAni=100)
plot(var_standardised,metric,map=F)

metric_Vgm <- fit.StVariogram(var_standardised, metric, method="L-BFGS-B")
plot(var_standardised,metric_Vgm, wireframe=T)
plot(var_standardised,metric_Vgm, map=F)

## SumMetric

parameter_tuning_standardised_summetric
parameter_tuning_standardised_summetric %>%
filter(mse==min(parameter_tuning_standardised_summetric$mse))
sumMetric <- vgmST("sumMetric",
               space = vgm(0.5, "Sph", range=325, nugget=0.4),
               time = vgm(0.5, "Sph", range=8, nugget = 0.2),
               joint = vgm(0.1,"Mat", range=375, nugget=0.1), stAni=100)
plot(var_standardised,sumMetric,map=F)


sumMetric_Vgm <- fit.StVariogram(var_standardised, sumMetric, method="L-BFGS-
B",tunit="hours")
plot(var_standardised,sumMetric_Vgm, wireframe=T)
plot(var_standardised,sumMetric_Vgm, map=F)

## SimpleSumMetric

parameter_tuning_standardised_simplesummetric
parameter_tuning_standardised_simplesummetric %>%
filter(mse==min(parameter_tuning_standardised_simplesummetric$mse))
SimplesumMetric <- vgmST("simpleSumMetric",
                   space = vgm(0.5, "Sph", range=325, nugget=0),
                   time = vgm(0.5, "Sph", range=8, nugget = 0),
                   joint = vgm(0.1,"Mat", range=325, nugget=0), nugget=0.4, stAni=100)
plot(var_standardised,SimplesumMetric,map=F)
```

```
SimplesumMetric_Vgm <- fit.StVariogram(var_standardised, SimplesumMetric,method =
"L-BFGS-B")
plot(var_standardised,SimplesumMetric_Vgm, wireframe=T)
plot(var_standardised,SimplesumMetric_Vgm, map=F)

attr(SimplesumMetric_Vgm, "MSE")

## Comparing all models

best_models_standardised <-
data.frame(model.type=c("separable","prodsum","metric","summetric","simplesummetric"),
                              lowest.mse.achievable =
c(unique(filter(parameter_tuning_standardised_separable,mse==min(parameter_tuning_stand
ardised_separable$mse))[,"mse"]),

unique(filter(parameter_tuning_standardised_prodsum,mse==min(parameter_tuning_standar
dised_prodsum$mse))[,"mse"]),

unique(filter(parameter_tuning_standardised_metric,mse==min(parameter_tuning_standardis
ed_metric$mse))[,"mse"]),

unique(filter(parameter_tuning_standardised_summetric,mse==min(parameter_tuning_stand
ardised_summetric$mse))[,"mse"]),

unique(filter(parameter_tuning_standardised_simplesummetric,mse==min(parameter_tuning
_standardised_simplesummetric$mse))[,"mse"])))
plot(var_standardised,list(separable_Vgm, prodSumModel_Vgm, metric_Vgm,
sumMetric_Vgm, SimplesumMetric_Vgm),all=T,wireframe=T)




# sub_test_STIDF creation-----------------------------------------------------------------------




## durations_test.UTM,    durations_testSP, durations_testTM, durations_testDF,
sub_test_STIDF creation

y <- 2500 #respecify this sample size number and run the following chunk again for each
new variogram trial
durations_test.UTM <- sample_n(sub_test_standardised, y)
coordinates(durations_test.UTM) = ~lon+lat
projection(durations_test.UTM)=CRS("+init=epsg:4326")
durations_test.UTM <- spTransform(durations_test.UTM, CRS("+init=epsg:3395"))

durations_testSP <- SpatialPoints(durations_test.UTM@coords,CRS("+init=epsg:3395"))
#coordinates dataset
```

```
duplicates_test <- zerodist(durations_testSP) #no duplicate values allowed in the values and
time dataset
durations_testDF <- data.frame(durations = durations_test.UTM$duration)#[-duplicates[,2]]
durations_testTM <- durations_test.UTM$DepartureTime#[-duplicates[,2]]

sub_test_STIDF <- STIDF(durations_testSP, durations_testTM, data=durations_testDF)
stplot(sub_test_STIDF)
summary(sub_test_STIDF)




# Kriging------------------------------------------------------------------------------------------




## predictions
pred <- krigeST(durations~1, data=sub_STIDF, modelList=prodSumModel_Vgm,
newdata=sub_test_STIDF)

## prediction_comparison_df creation
pred_df <- data.frame(pred)
sub_test_df <- data.frame(sub_test_STIDF)
prediction_comparison_df <- merge(pred_df, sub_test_df, by=c("sp.ID", "timeIndex"))[,
c("lat.x", "lon.x",
                                                          "sp.ID","timeIndex",
                                                          "durations", "var1.pred")]

## prediction comparison density plots
original_vs_duration <- melt(prediction_comparison_df ,
                    id.vars = c("sp.ID","timeIndex","lat.x", "lon.x"),
                    variable.name = 'series')
prediction_comparison_plot <- ggplot(original_vs_duration, aes(value)) +
geom_density(aes(colour = series))
prediction_comparison_plot

## stplot for pred and sub_test_df
stplot(pred)
stplot(sub_test_STIDF)

## squared deviations
prediction_comparison_df$sq_dev <- ((prediction_comparison_df$durations -
prediction_comparison_df$var1.pred)^2)
prediction_comparison_df$dev <- ((prediction_comparison_df$durations -
prediction_comparison_df$var1.pred))
summary(prediction_comparison_df)
hist(prediction_comparison_df$sq_dev)
hist(prediction_comparison_df$dev)
```

```
#density plots of all models in predicting
pred1 <- krigeST(durations~1, data=sub_STIDF, modelList=separable_Vgm,
newdata=sub_test_STIDF)
pred2 <- krigeST(durations~1, data=sub_STIDF, modelList=prodSumModel_Vgm,
newdata=sub_test_STIDF)
pred3 <- krigeST(durations~1, data=sub_STIDF, modelList=metric_Vgm,
newdata=sub_test_STIDF)
pred4 <- krigeST(durations~1, data=sub_STIDF, modelList=sumMetric_Vgm,
newdata=sub_test_STIDF)
pred5 <- krigeST(durations~1, data=sub_STIDF, modelList=SimplesumMetric_Vgm_Vgm,
newdata=sub_test_STIDF)

pred_df_all <- data.frame(pred1, pred2, pred3, pred4, pred5)
prediction_comparison_df_all <- merge(pred_df_all, sub_test_df, by=c("sp.ID",
"timeIndex"))[, c("lat.x", "lon.x",
                                                  "sp.ID","timeIndex",
                                                  "durations", "var1.pred")]
original_vs_duration_all <- melt(prediction_comparison_df_all ,
                      id.vars = c("sp.ID","timeIndex","lat.x", "lon.x"),
                      variable.name = 'series')
prediction_comparison_plot_all <- ggplot(original_vs_duration_all, aes(value)) +
geom_density(aes(colour = series))
prediction_comparison_plot_all

# Outlier Detection-------------------------------------------------------------------------



## z-score method
outlier.index <- which(abs(prediction_comparison_df$dev)>3)
length(outlier.index)
deviation_outliers <- prediction_comparison_df[c(outlier.index),]

register_google(key="AIzaSyCYIMu3bBtnw0_QVoGGq0XXpTaZ_07DDps")

melbourne_google <- get_googlemap("Melbourne, Australia",zoom=14, maptype = "terrain")
melbourne_stamen <- get_map("Melbourne,Australia", zoom=14, source= "stamen",
maptype = "terrain")
melb1 <- ggmap(melbourne_google)
melb2 <- ggmap(melbourne_stamen)

qplot(x=lon.x, y=lat.x, data=deviation_outliers, geom="point") + labs(title = "Outliers (in
Actual-Predicted)",
                                        x= "Longitude",
                                        y="Latitude")

melb_parking_outliers <- melb1 + geom_point(data = deviation_outliers,
                        aes(x=lon.x, y=lat.x),
                    size=1, colour = "red", alpha=0.75) + labs(title = "Melbourne
Parking Bays",
                                              x= "Longitude",
                                              y="Latitude")
```