Supplementary Material

Deriving and optimising the cloaking variables

When solving for M, we put an arbitrary (positive) bound on Δ of 1, as any scaling of Δ caused by manipulating M, will scale the determinant of M by the inverse amount.[6]

We will express our problem as trying to *maximise* the entropy of a k-dimensional Gaussian distribution with covariance $P = M^{-1}$;

$$\text{Maximise } \ln(|P|), \text{ subject to } n \text{ constraints, } 0 \leq \mathbf{c}_i^\top P \mathbf{c}_i \leq 1.$$

Considering just the upper bounds and expressing this as a constraint satisfaction problem using Lagrange multipliers we have $L = \ln|P| + \sum_i \lambda_i(1 - \mathbf{c}_i^\top P \mathbf{c}_i)$. Differentiating (and setting to zero), $\frac{\partial L}{\partial P} = P^{-1} - \sum_i \lambda_i \mathbf{c}_i \mathbf{c}_i^\top = 0$. We also have the slackness conditions (for all i), $\lambda_i(\mathbf{c}_i^\top P \mathbf{c}_i - 1) = 0$ and we also require that $\lambda_i \geq 0$. Rearranging the derivative, we have, $P^{-1} = \sum_i \lambda_i \mathbf{c}_i \mathbf{c}_i^\top$. Note that as $\lambda_i \geq 0$, P^{-1} is positive semi-definite (psd),[7] thus the initial lower bound (that $\mathbf{c}_i^\top P \mathbf{c}_i \geq 0$) is met. The upper bound (that $\mathbf{c}_i^\top P \mathbf{c}_i \leq 1$) is achieved if the λ_i are correctly chosen, such that the Lagrange and slackness conditions are met.

We now must maximise the expression for L wrt λ_j. To assist with this we rewrite our expression for $P^{-1} = \sum_i \lambda_i \mathbf{c}_i \mathbf{c}_i^\top = C \Lambda C^\top$, where $C = [\boldsymbol{c}_1, \boldsymbol{c}_2 ... \boldsymbol{c}_n] = K_{*f} K^{-1}$ and Λ is a diagonal matrix of the values of λ_i. The summation in the expression for L, $\sum_i \lambda_i(1 - \mathbf{c}_i^\top P \mathbf{c}_i)$ can be written as $\operatorname{Tr}\left(\Lambda - C^\top P C \Lambda\right)$. Substituting in our definition of P, we can write the summation as: $\operatorname{Tr}\left(\Lambda - C^\top (C \Lambda C^\top)^{-1} C \Lambda\right) = \operatorname{Tr}\left(\Lambda - C^\top C^{-\top} \Lambda^{-1} C^{-1} C \Lambda\right)$. Assuming C is invertible, the summation becomes $\operatorname{Tr}\left(\Lambda - \Lambda^{-1}\Lambda\right)$. Differentiating this wrt λ_j equals one. We can use this result to find the gradient of L wrt λ_j:

$$\begin{aligned}
\frac{\partial L}{\partial \lambda_j} &= \frac{\partial \ln|P|}{\partial \lambda_j} + \frac{\partial}{\partial \lambda_j} \sum_i \lambda_i (1 - \mathbf{c}_i^\top P \mathbf{c}_i) \\
&= \operatorname{Tr}\left(P^{-1} \frac{\partial P}{\partial \lambda_j}\right) + 1 \\
&= \operatorname{Tr}\left(-P^{-1} P \boldsymbol{c}_j \boldsymbol{c}_j^\top P\right) + 1 \\
&= -\operatorname{Tr}\left(\boldsymbol{c}_j \boldsymbol{c}_j^\top M^{-1}\right) + 1 \\
&= -\boldsymbol{c}_j^\top M^{-1} \boldsymbol{c}_j + 1
\end{aligned} \tag{10}$$

This can be solved using a gradient descent method, to give us those values of λ_i which minimise $\log|M|$ while ensuring $\Delta \leq 1$.

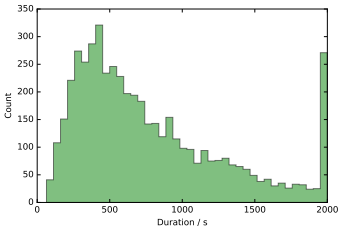

Figure 4: The duration of 5000 Citi Bike journeys. Note the effect caused by thresholding at 2000 s.

Citi Bike duration distribution

Figure 4 illustrates the distribution of journey times and the effect of thresholding.

Algorithms

Algorithm 2 describes the Cloaking method. Algorithm 1 describes the earlier method, from Section 2.

Hyperparameter selection

So far in this paper we have selected the values of the kernel hyperparameters *a priori*. Normally one may maximise the marginal likelihood to select the values or potentially integrates over the hyperparameters. In differential privacy we must take care when using private data to make these choices. Previous work exists to perform this selection, for example Kusner et al. [2015] describes a method for performing differentially private Bayesian optimisation, however their method assumes the training data is not private. Kusner et al. [2015] do suggest that the work of Chaudhuri and Vinterbo [2013] may allow Bayesian Optimisation to work in the situation in which the training data also needs to be private.

We decided instead that, due to the low-dimensionality of many hyperparameter problems, a simple grid search, combined with the exponential mechanism may allow the

[6] For example if we write a new Δ' with M as mM, we see that we can take out the m term from Δ''s inequality, leaving $\Delta' = m^{-1/2}\Delta$. When we scale Z, which has covariance mM by this new value of Δ' the covariance of the scaled Z equals $(\Delta')^2 mM = (m^{-1/2}\Delta)^2 mM = \Delta^2 M$, the magnitude change cancels, so any positive value of Δ (e.g. 1) will suffice for the optimisation. Also: in the following we ignore d and reintroduce it at the end of the derivation by defining $\Delta = d$ instead of it equalling 1.

[7] The summation is of a list of positive semi-definite rank-one matrices. One can see that such a sum is also positive semi-definite (to see this, consider distributing $\boldsymbol{z}^\top$ and $\boldsymbol{z}$ over the summation).

Algorithm 1 Using the initial DP algorithm

Require: $\mathcal{M}$, the GP model (the kernel, hyperparameters and training inputs and normalised outputs)
Require: $X_* \in R^{P\times D}$, (the matrix of test inputs)
Require: $d > 0$, data sensitivity (maximum change possible) & $\varepsilon > 0, \delta \geq 0$, the DP parameters

function DIFFERENTIALLYPRIVATEREGRESSION(X_*, M, d, ε, δ)
  $K \leftarrow \mathcal{M}$.GET_K ▷ covariance between training points
  $\Delta \leftarrow d^2 \text{B}(K^{-1})^2$
  $\boldsymbol{f}_*, \sigma_*^2 \leftarrow \mathcal{M}$.GET_PREDICTIONS($X_*$) ▷ Calculate non-DP predictions
  $\tilde{\boldsymbol{f}}_* \leftarrow \boldsymbol{f}_* + (\Delta d \text{C}(\delta)/\varepsilon)\boldsymbol{z}$ ▷ $\boldsymbol{z} \sim \mathcal{N}(0, K)$
  return $\tilde{\boldsymbol{f}}_*, \sigma_*^2$
end function

function B(K^{-1})
  return $\max(|-[K_A^{-1}]^-|_\infty, |[K_A^{-1}]^+|_\infty)$
end function

function C(δ)
  return $\sqrt{2log(2/\delta)}$
end function

selection of an acceptable set of hyperparameters. For the utility function we considered using the log marginal likelihood, with additional noise in the data-fit term to capture the effect of the DP noise. However for simplicity in estimating the bound and to avoid overfitting we simply used the sum square error (SSE) over a series of K-fold cross-validation runs, which for a given fold is $\sum_{i=1}^{N}\left(y_{*i} - y_{\text{t}i}\right)^2$, with predictions $\boldsymbol{y}_*$ and test values $\boldsymbol{y}_\text{t}$.

Before proceeding we need to compute a bound on the sensitivity of the SSE. To briefly recap, the DP assumption is that one data point has been perturbed by at most d. We need to bound the effect of this perturbation on the SSE. First we realise that this data point will only be in the *test set* in one of the K folds. In the remaining folds it will be in the training data.

If the perturbed data point is in the training data ($\boldsymbol{y}$), then we can compute the sensitivity of the SSE. The perturbation this would cause to the predictions ($\boldsymbol{y}_*$) is described using standard GP regression (and the cloaking matrix). Specifically a change of d in training point j will cause a $d\boldsymbol{c}_{jk}$ change in the test point predictions, where $\boldsymbol{c}_{jk}$ is the jth column of the cloaking matrix for the kth fold.

To compute the perturbation caused by the change in the training data, we note that the SSE is effectively the square of the euclidean distance between the prediction and the test data. We are moving the prediction by $d\boldsymbol{c}_{jk}$. The largest effect that this movement of the prediction point could have on the distance between prediction and test locations is if it moves the prediction in the opposite direction to the test points. Thus it can increase (or decrease) the distance between the test and predictions by the largest length of $d\boldsymbol{c}_{jk}$ over training points. Hence for one of the folds, the largest change in the SSE is $d^2 \max_j |\boldsymbol{c}_{jk}|_2^2$.

If the perturbed data point, j, was in the test data then the SSE will change by $\left(y_{*j} + d - y_{\text{t}j}\right)^2 - \left(y_{*j} - y_{\text{t}j}\right)^2 = d^2 + 2d(y_{*j} - y_{\text{t}j})$. The last part of the expression (the error in the prediction for point j) is unbounded. To allow us to constrain the sensitivity we enforce a completely arbitrary bound of being no larger than $\pm 4d$, thresholding the value if it exceeds this. Thus a bound on the effect of the perturbation is $d^2 + 2d \times 4d = d^2 + 8d^2 = 9d^2$.

The SSE of each fold is added together to give an overall SSE for the cross-validation exercise. We sum the $K-1$ largest sensitivities and add $9d^2$ to account for the effect of the single fold in which the perturbing data point, j, will be in the test set. The perturbation could have been in the test data in any of the folds. We assumed it was in the fold with the smallest training-data sensitivity to allow us to make the result a lower bound on the sensitivity of the SSE to the perturbation. If it had been in any other fold the sensitivity would have been less (as more sensitivity would be contributed by the sum over the training sensitivities). Thus the sensitivity of the SSE over the K folds is; $9d^2 + \sum_{k=1}^{K-1} d^2 \max_j |\boldsymbol{c}_{jk}|_2^2$ (where the k folds are ordered by decreasing sensitivity)

We compute the SSE and the SSE's sensitivity for each of the hyperparameter combinations we want to test. We then use the computed sensitivity bound with the exponential mechanism to select the hyperparameters. To summarise, to use the exponential mechanism one evaluates the utility $u(x, r)$ for a given database x and for elements r, from a range. One also computes the sensitivity of this utility function by picking the highest sensitivity of any

Algorithm 2 Using the cloaking algorithm

Require: $\mathcal{M}$, the GP model (the kernel, hyperparameters and training inputs and normalised* outputs)
Require: $X_* \in R^{P \times D}$, (the matrix of test inputs)
Require: $d > 0$, data sensitivity (maximum change possible) & $\varepsilon > 0, \delta \geq 0$, the DP parameters

function DifferentiallyPrivateCloakingRegression(X_*, M, d, ε, δ)
  $C \leftarrow \mathcal{M}$.get_C(X_*) ▷ Compute the value of the cloaking matrix ($K_{*f}K^{-1}$)
  $\boldsymbol{\lambda} \leftarrow$ findLambdas(C)
  $M \leftarrow$ calcM($\boldsymbol{\lambda}, C$) ▷ Calculate the DP noise covariance matrix
  $\Delta \leftarrow$ calcDelta($\boldsymbol{\lambda}, C$)†
  $c \leftarrow \sqrt{2log\frac{2}{\delta}}$
  $\boldsymbol{y}_*, \sigma_*^2 \leftarrow \mathcal{M}$.get_predictions($X_*$) ▷ Calculate non-DP predictions
  $\tilde{\boldsymbol{y}}_* \leftarrow \boldsymbol{y}_* + (\Delta dc/\varepsilon)\boldsymbol{z}$ ▷ $\boldsymbol{z} \sim \mathcal{N}(0, M)$
  return $\tilde{\boldsymbol{y}}_*, \sigma_*^2$
end function

function $\mathcal{M}$.get_C(X_*)
  From $\mathcal{M}$ compute K_{*f} and K^{-1} ▷ Compute covariances between training and test points
  $C \leftarrow K_{*f}K^{-1}$
  return C
end function

function findLambdas(C)
  $\boldsymbol{\lambda} \leftarrow$ Uniform$(0.1, 0.9)$ ▷ Initialise randomly‡
  $\alpha \leftarrow 0.05$ ▷ Learning rate
  do
    $\frac{dL}{d\boldsymbol{\lambda}} \leftarrow$ CalcGradient($\boldsymbol{\lambda}, C$)
    $\Delta_{\boldsymbol{\lambda}} \leftarrow -\frac{dL}{d\boldsymbol{\lambda}}\alpha$
    $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \Delta_{\boldsymbol{\lambda}}$
  while $\Delta_{\boldsymbol{\lambda}} > 10^{-5}$
  return $\boldsymbol{\lambda}$
end function

function calcGradient($\boldsymbol{\lambda}, C$)
  M $\leftarrow$ calcM($\boldsymbol{\lambda}, C$)
  for $0 \leq j < N$ **do** ▷ N, number of columns in cloaking matrix, C.
    $[\frac{dL}{d\boldsymbol{\lambda}}]_j \leftarrow -\operatorname{Tr}\left(M^+ C_{:j} C_{:j}^\top\right) + 1$
  end for
  return $\frac{dL}{d\boldsymbol{\lambda}}$
end function

function calcM($\boldsymbol{\lambda}, C$)
  $M \leftarrow \sum_i \lambda_i C_{:i} C_{:i}^\top$
  return M
end function

function calcDelta($\boldsymbol{\lambda}, C$)
  $M \leftarrow$ calcM($\boldsymbol{\lambda}, C$)
  $\Delta \leftarrow \mathbf{max}_j\ C_{:j}^\top M^+ C_{:j}$
  return Δ
end function

*We assume the user will handle normalisation.
†Although we should have optimised M such that $\Delta \leq 1$, it may not have completely converged, so we compute the Δ bound for the value of M we have actually achieved.
‡We have found that occasionally the algorithm fails to converge. To confirm convergence we have found it useful to reinitialise and run the algorithm a few times.

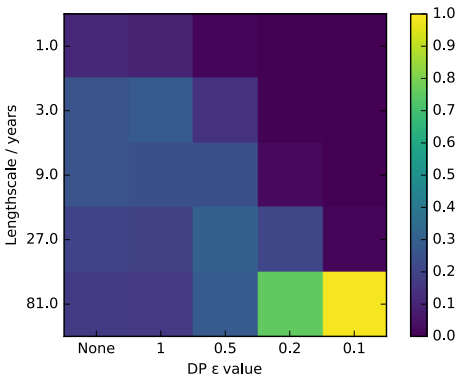

Figure 5: Effect of varying the differential privacy parameter, ε, on the likelihood of selecting each lengthscale. Colour indicates probability of parameter selection. With low privacy, a short lengthscale is appropriate which allows the GP to describe details in the data. With high privacy, a longer lengthscale is required, which will average over large numbers of individual data points.

of the utilities; in this case each utility corresponds to a negative SSE, and each sensitivity corresponds to the sum described above.

$$\Delta_u \triangleq \max_{r \in R} \max_{x,y} |u(x,r) - u(y,r)|$$

(where x and y are neighbouring databases).

The exponential mechanism selects an element r with probability proportional to:

$$\exp\left(\frac{\varepsilon u(x,r)}{2\Delta_u}\right)$$

Note that for a given privacy budget, some will need to be expended on this selection problem, and the rest expended on the actual regression.

Effect of privacy on optimum hyperparameter selection

A final interesting result is in the effect of the level of privacy in the regression stage on the selection of the lengthscale. This is demonstrated in the distribution of probabilities over the lengthscales when we adjust ε. Figure 5 demonstrates this effect. Each column is for a different level of privacy (from none to high) and each tile shows the probability of selecting that lengthscale. For low privacy, short lengthscales are acceptable, but as the privacy increases, averaging over more data allows us to give more accurate answers.

Privacy on the training inputs

To release the mean function such that the training inputs remain private, we need a general bound on the infinity norm of the covariance function, that does not depend explicitly on the values of X.

Varah [1975] show that if J is *strictly diagonally dominant*[8] then:

$$||J^{-1}||_\infty \leq \max_{1 \leq i \leq n} \frac{1}{\Delta_i(J)} = b(J)$$

where we have defined this bound as $b(J^{-1})$. We also define $\Delta_i(J) = |J_{ii}| - \sum_{j \neq i} |J_{ij}|$, i.e. the sum of the off diagonal elements in row i subtracted from the diagonal element.

So if K is strictly diagonally dominant (which is achieved if the inputs are sufficiently far apart, and/or if sufficient uncertainty exists on the outputs), then we have a bound on the sums of its rows. The above bound means that,

$$\sum_{i=1}^{n} \alpha_i - \alpha_i' \leq \Delta_y b(J^{-1}) \tag{11}$$

To ensure sufficient distance between inputs, we could use inducing variables, which can be arbitrarily placed, so that the above constraints on the covariance matrix are observed.

Integral Kernel

We used a custom kernel in the Citi Bike comparison to make predictions over the binned DP-noisy dataset. In summary the observations were considered to be the integrals over each bin (provided by multiplying the noisy means by the sizes of the bins). The predictions were made at points on the latent function being integrated. This allowed us to make predictions from the binned data in a principled manner.

Considering just one dimension. To compute the covariance for the integrated function, $F(\cdot)$, we integrate the original EQ kernel, $k_{ff}(\cdot,\cdot)$, over both its input values,

$$k_{FF}((s,t),(s',t')) = \alpha \int_s^t \int_{s'}^{t'} k_{ff}(u,u') \, \mathrm{d}u' \mathrm{d}u,$$

so that given two pairs of input locations, corresponding to two definite integrals, we can compute the covariance between the two.

Similarly we can calculate the cross-covariance k_{Ff} between F and f. Both k_{FF} and k_{Ff} can be extended to multiple dimensions. Each kernel function contains a unique lengthscale parameter, with the bracketed kernel subscript

[8] A matrix, J, is strictly diagonally dominant if $\Delta_i(J) > 0$ for all $1 \leq i \leq n$.

index indicating these differences. We can express the new kernel as the product of our one dimensional kernels:

$$k_{FF} = \prod_i k_{FF(i)}((s_i, t_i), (s_i', t_i')),$$

with the cross covariance given by

$$k_{Ff} = \prod_i k_{Ff(i)}((s_i, t_i), (s_i', t_i')).$$

The above expressions can then be used to make predictions of the latent function f given observations of its definite integrals.