EXPERIMENTAL DESIGN FOR FAST LINEAR ALGEBRA

STEPHEN HUAN* AND FLORIAN SCHÄFER†

Abstract. experimental design for linear algebra

Key words. to do

AMS subject classifications.

1. Introduction.

[3] test citation

2. Greedy selection for directed inference.

2.1. Conditional *k*-th nearest neighbors.

Consider the simple regression algorithm kth-nearest neighbors (k-NN). Given a training set $X_{\text{Tr}} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$ and corresponding labels $\boldsymbol{y}_{\text{Tr}} = \{y_1, \dots, y_n\}$, the goal is to estimate the unknown label y_{Pr} of some unseen prediction point $\boldsymbol{x}_{\text{Pr}}$ Stated informally, the k-NN approach is to select the k points in X_{Tr} *most informative* about $\boldsymbol{x}_{\text{Pr}}$ and combine their results.

Algorithm 2.1 Idealized k-NN regression

```
Given $(X_{\text{Tr}}, \boldsymbol{y}_{\text{Tr}}) = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n)\}$ and $\boldsymbol{x}_{\text{Pr}}$
    1. Select the $k$ points in $X_{\text{Tr}}$ most informative about $\boldsymbol{x}_{\text{Pr}}$
    2. Combine the labels of the selected points to generate a prediction
```

One specific approach is intuitively, points close to $\boldsymbol{x}_{\text{Pr}}$ should be similar to it. So we select the k closest points in X_{Tr} to $\boldsymbol{x}_{\text{Pr}}$ and pool their labels (e.g., by averaging).

Algorithm 2.2 k-NN regression

```
    1. Select the $k$ points $\{\boldsymbol{x}_{i_1}, \dots, \boldsymbol{x}_{i_k}\} \subseteq X_{\text{Tr}}$ closest to $\boldsymbol{x}_{\text{Pr}}$
    2. Compute $\boldsymbol{y}_{\text{Pr}}$ by $\frac{1}{k} \sum_{j=1}^{k} y_{i_j}$
```

However, we can generalize the notion of "closest" with the *kernel trick*, by using an arbitrary kernel function to measure similarity. For example, commonly used kernels like the Gaussian kernel and Matérn family of covariance functions are *isotropic*; they depend only on the distance between the two vectors. If such isotropic kernels monotonically decrease with distance, then selecting points based on the largest kernel similarity recovers k-NN. However, kernels need not be isotropic in general — they just need to capture some sense of "similarity", motivating kernel k-NN.

not sure whether "stationary" or "isotropic" are the right word(s) to use here

Algorithm 2.3 Kernel k-NN regression

```
Given kernel function $K(\boldsymbol{x}, \boldsymbol{y})$
    1. Select the $k$ points $\{\boldsymbol{x}_{i_1}, \dots, \boldsymbol{x}_{i_k}\} \subseteq X_{\text{Tr}}$ most similar to $\boldsymbol{x}_{\text{Pr}}$
    2. Compute $\boldsymbol{y}_{\text{Pr}}$ by an average weighted by similarity
```

*Georgia Institute of Technology

†Georgia Institute of Technology, S1317 CODA, 756 W Peachtree St Atlanta, GA 30332, florian.schaefer@cc.gatech.edu, Corresponding Author

Although the kernel k-NN approach is more general than its normed counterpart, it still suffers from a fundamental issue. Suppose the closest point to $\boldsymbol{x}_{\mathrm{Pr}}$ has many duplicates in the training set. Then the algorithm will select the same point multiple times, even though in some sense the duplicate point has stopped giving additional information about the prediction point. In order to fix this issue, we should be selecting new points *conditional* on the points we've already selected. This preserves the idealized algorithm of selecting points based on the information they tell us about the prediction point — once we've selected a point, conditioning on it reduces the information similar points tells us, encouraging diverse point selection.

Algorithm 2.4 Conditional kernel k-NN regression

```
1. Select the $k$ points $\{\boldsymbol{x}_{i_1}, \dots, \boldsymbol{x}_{i_k}\} \subseteq X_{\mathrm{Tr}}$ most informative to $\boldsymbol{x}_{\mathrm{Pr}}$
   after conditioning on all points selected beforehand
2. Compute $\boldsymbol{y}_{\mathrm{Pr}}$ by an average weighted by information
```

In order to make the notions of conditioning and information precise, we need a specific framework. Kernel methods lead naturally to Gaussian processes, whose covariance matrices naturally result from kernel functions and allows us to use the rigorous statistical and information-theoretic notions of conditioning and information.

mention sensor placement/spatial statistics literature

2.2. Sparse Gaussian process regression. A *Gaussian process* is a prior distribution over functions, such that for any finite set of points, the corresponding function over the points is distributed according to a multivariate Gaussian. In order to generate such a distribution over an uncountable number of points consistently, a Gaussian process is specified by a *mean function* $\mu(\boldsymbol{x})$ and *covariance function* or *kernel function* $K(\boldsymbol{x}, \boldsymbol{y})$. For any finite set of points $X = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$, $f(X) \sim \mathcal{N}(\boldsymbol{\mu}, \Theta)$, where $\boldsymbol{\mu}_i = \mu(\boldsymbol{x}_i)$ and $\Theta_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

In order to compute a prediction at $\boldsymbol{x}_{\mathrm{Pr}}$, we can simply condition the desired prediction $\boldsymbol{y}_{\mathrm{Pr}}$ on the observed outputs and compute the conditional expectation. We can also find the conditional variance, which will quantify the uncertainty of our prediction. If we block our covariance matrix $\Theta = \begin{pmatrix} \Theta_{\mathrm{Tr},\mathrm{Tr}} & \Theta_{\mathrm{Tr},\mathrm{Pr}} \\ \Theta_{\mathrm{Pr},\mathrm{Tr}} & \Theta_{\mathrm{Pr},\mathrm{Pr}} \end{pmatrix}$ where $\Theta_{\mathrm{Tr},\mathrm{Tr}}, \Theta_{\mathrm{Pr},\mathrm{Pr}}, \Theta_{\mathrm{Tr},\mathrm{Pr}}, \Theta_{\mathrm{Pr},\mathrm{Tr}}$ are the covariance matrices of the training data, testing data, and training and test data respectively, then the conditional expectation and covariance are:

$$\mathrm{E}[\boldsymbol{y}_{\mathrm{Pr}} \mid \boldsymbol{y}_{\mathrm{Tr}}] = \boldsymbol{\mu}_{\mathrm{Pr}} + \Theta_{\mathrm{Pr},\mathrm{Tr}} \Theta_{\mathrm{Tr},\mathrm{Tr}}^{-1} (\boldsymbol{y}_{\mathrm{Tr}} - \boldsymbol{\mu}_{\mathrm{Tr}}) \tag{2.1}$$

$$\mathrm{Cov}[\boldsymbol{y}_{\mathrm{Pr}} \mid \boldsymbol{y}_{\mathrm{Tr}}] = \Theta_{\mathrm{Pr},\mathrm{Pr}} - \Theta_{\mathrm{Pr},\mathrm{Tr}} \Theta_{\mathrm{Tr},\mathrm{Tr}}^{-1} \Theta_{\mathrm{Tr},\mathrm{Pr}} \tag{2.2}$$

Note that calculating the posterior mean and variance requires inverting the training covariance matrix $\Theta_{\mathrm{Tr},\mathrm{Tr}}$, which costs $\mathcal{O}(N^3)$, where N is the number of training points. This scaling is prohibitive for large datasets, so many *sparse* Gaussian process regression techniques have been proposed. These methods often focus on selecting a subset of the training data that is most informative about the prediction points, which naturally aligns with our k-NN perspective. If s points are selected out of the N, then the inversion will cost $\mathcal{O}(s^3)$, which could be substantially cheaper if s is significantly smaller than N. The question is then how to select as few points as possible while maintaining predictive accuracy.

cite sparse Gaussian regression papers

2.3. Problem: optimal selection. The natural criterion justified from the k-NN perspective is to maximize the *mutual information* between the selected points and the target point for prediction. Such information-theoretic objectives have seen success in the spatial statistics community [1], who use such criteria to determine the best locations to place sensors in a Gaussian process regression context. The mutual information, or *information gain* is defined as

$$\mathrm{I}[\boldsymbol{y}_{\mathrm{Pr}}; \boldsymbol{y}_{\mathrm{Tr}}] = \mathrm{H}[\boldsymbol{y}_{\mathrm{Pr}}] - \mathrm{H}[\boldsymbol{y}_{\mathrm{Pr}} \mid \boldsymbol{y}_{\mathrm{Tr}}] \tag{2.3}$$

We can use the fact that the entropy of a multivariate Gaussian is monotonically increasing with the log determinant of its covariance matrix to efficiently compute these entropies. Because the entropy of $\boldsymbol{y}_{\mathrm{Pr}}$ is constant, maximizing the mutual information is equivalent to minimizing the conditional entropy. From (2.2) we see that minimizing the conditional entropy is equivalent to minimizing the log determinant of the posterior covariance matrix. Note that for a single predictive point, this is monotonic with its variance. So another justification is that we are reducing the *conditional variance* of the desired point as much as possible. In particular, because our estimator is the conditional expectation (2.1), it is unbiased because $\mathrm{E}[\mathrm{E}[\boldsymbol{y}_{\mathrm{Pr}} \mid \boldsymbol{y}_{\mathrm{Tr}}]] = \mathrm{E}[\boldsymbol{y}_{\mathrm{Pr}}]$. Because it is unbiased, its expected mean squared error is simply the conditional variance since $\mathrm{E}[(\boldsymbol{y}_{\mathrm{Pr}} - \mathrm{E}[\boldsymbol{y}_{\mathrm{Pr}} \mid \boldsymbol{y}_{\mathrm{Tr}}])^2 \mid \boldsymbol{y}_{\mathrm{Tr}}] = \mathrm{Var}[\boldsymbol{y}_{\mathrm{Pr}} \mid \boldsymbol{y}_{\mathrm{Tr}}]$ where the expectation is taken under conditioning because of the assumption that $\boldsymbol{y}_{\mathrm{Pr}}$ is distributed according to the Gaussian process. So maximizing the mutual information is equivalent to minimizing the conditional variance which is in turn equivalent to minimizing the expected mean squared error of the prediction. Another perspective on the objective can be derived from comparing the mutual information to the EV-VE identity, which states

$$\begin{aligned}\mathrm{H}[\boldsymbol{y}_{\mathrm{Pr}}] &= \mathrm{H}[\boldsymbol{y}_{\mathrm{Pr}} \mid \boldsymbol{y}_{\mathrm{Tr}}] + \mathrm{I}[\boldsymbol{y}_{\mathrm{Pr}}; \boldsymbol{y}_{\mathrm{Tr}}] \\ \mathrm{Var}[\boldsymbol{y}_{\mathrm{Pr}}] &= \mathrm{E}[\mathrm{Var}[\boldsymbol{y}_{\mathrm{Pr}} \mid \boldsymbol{y}_{\mathrm{Tr}}]] + \mathrm{Var}[\mathrm{E}[\boldsymbol{y}_{\mathrm{Pr}} \mid \boldsymbol{y}_{\mathrm{Tr}}]]\end{aligned}$$

On the left hand side, entropy is monotone with variance. On the right hand side, the expectation of the conditional variance can be interpreted to be the fluctuation of the prediction point after conditioning, and is monotone with the conditional entropy. Because the expectation of conditional variance and variance of conditional expectation add to a constant, minimizing the expectation of the conditional variance is equivalent to maximizing the variance of conditional expectation, which we see corresponds to the mutual information term. Supposing $\boldsymbol{y}_{\mathrm{Pr}}$ was independent of $\boldsymbol{y}_{\mathrm{Tr}}$, then the conditional expectation becomes simply the expectation, whose variance is 0. Thus, the variance of the conditional expectation can be interpreted to be the "information" shared between $\boldsymbol{y}_{\mathrm{Pr}}$ and $\boldsymbol{y}_{\mathrm{Tr}}$, as the larger it is, the more the prediction for $\boldsymbol{y}_{\mathrm{Pr}}$ (the conditional expectation) depends on the observed results of $\boldsymbol{y}_{\mathrm{Tr}}$.

2.4. A greedy approach. We now consider how to efficiently minimize the conditional variance objective using a greedy approach. At each iteration, we pick the training point which most reduces the conditional variance of the prediction point. Let $I = \{i_1, i_2 \ldots i_j\}$ be the set of indexes of training points selected already. Let the prediction point have index n, the last index. For a candidate index k, elaborate on this argument, because iterative conditioning of the Gaussian process is Schur complementation and Cholesky factorization is recursive Schur complementation, we see that the amount selecting k will decrease the variance of $\boldsymbol{y}_{\text{Pr}}$ is

$$\frac{\operatorname{Cov}[\boldsymbol{y}_{\text{Pr}}, \boldsymbol{y}_{\text{Tr},k}]^2}{\operatorname{Var}[\boldsymbol{y}_{\text{Tr},k}, \boldsymbol{y}_{\text{Tr},k}]} = \frac{\Theta_{nk|I}^2}{\Theta_{kk|I}} \tag{2.4}$$

We can efficiently keep track of the conditional variance of each training point as well as the conditional covariance of each training point with the prediction point by maintaining a partial Cholesky factorization from the indices I already selected. finish describing algorithm(s), proofs, psuedocode

2.5. Supernodes and blocked selection. We now consider how to efficiently deal with multiple prediction points. justifying objective: mutual information leads to logdet of conditional covariance, information geometry perspective of logdet volume of uncertainty, scaling factor in probability density function for multivariate Gaussian

2.6. Near optimality by submodularity.

3. Greedy selection for *global* approximation by KL-minimization.

We have a symmetric, positive (semi-)definite kernel matrix Θ and wish to compute the *Cholesky factorization* of Θ into a lower triangular factor L such that $\Theta = LL^\top$. justify importance/applications of Cholesky factorization. This can be done in $\mathcal{O}(N^3)$ with standard algorithms, which is often prohibitive. Instead, we want to do so in a *sparse* manner. As such, we can only get an approximate L, $\hat{L}$ belonging to a pre-specified sparsity pattern S (a set of (row, column) indices that are allowed to be nonzero). In order to measure the performance of the estimator, we imagine both kernel matrices as the covariance matrices of multivariate Gaussians with mean $\mathbf{0}$. In order to compare the resulting two distributions, we use the *KL divergence* according to [2], or the expected difference in log-densities.

$$L := \underset{\hat{L} \in S}{\operatorname{argmin}}\, \mathbb{D}_{\text{KL}}\left(\mathcal{N}(\vec{0}, \Theta) \,\middle\|\, \mathcal{N}(\vec{0}, (\hat{L}\hat{L}^\top)^{-1})\right) \tag{3.1}$$

Note that here we are computing the Cholesky factorization of Θ^{-1}. Surprisingly enough, it is possible to exactly compute L. First, we re-write the KL divergence:

$$2\mathbb{D}_{\text{KL}}\left(\mathcal{N}(\vec{0}, \Theta_1) \,\middle\|\, \mathcal{N}(\vec{0}, \Theta_2)\right) = \operatorname{trace}(\Theta_2^{-1}\Theta_1) + \operatorname{logdet}(\Theta_2) - \operatorname{logdet}(\Theta_1) - N \tag{3.2}$$

THEOREM 3.1. *[2]. The non-zero entries of the ith column of L in (3.1) are:*

$$L_{s_i,i} = \frac{\Theta_{s_i,s_i}^{-1}\vec{e}_1}{\sqrt{\vec{e}_1^\top \Theta_{s_i,s_i}^{-1}\vec{e}_1}} \tag{3.3}$$

Plugging L (3.3) back into the KL divergence (3.2), we obtain:

$$\mathbb{D}_{\text{KL}}\left(\mathcal{N}(\vec{0}, \Theta) \,\middle\|\, \mathcal{N}(\vec{0}, (LL^\top)^{-1})\right) = -\operatorname{logdet}(LL^\top) - \operatorname{logdet}(\Theta)$$

Because L is lower triangular, its determinant is just the product of its diagonal entries:

$$= -2\sum_{i=1}^{N} \log(L_{ii}) - \operatorname{logdet}(\Theta)$$

Plugging (3.3) for its diagonal entry,

$$= -\sum_{i=1}^{N} \log(\boldsymbol{e}_1^\top \Theta_{s_i,s_i}^{-1} \boldsymbol{e}_1) - \operatorname{logdet}(\Theta) \tag{3.4}$$

In order to maximize (3.4), we can maximize over each column independently, since each term in the sum only depends on a single column. We want to maximize $\boldsymbol{e}_1^\top \Theta_{s_i,s_i}^{-1} \boldsymbol{e}_1$, the term corresponding to the diagonal entry in the inverse of the submatrix of Θ corresponding to the entries we've taken.

Given a sparsity pattern, we can construct L. The question is now how to compute a good sparsity pattern such that the resulting L has as small KL divergence as possible. First, we can consider a single column since the KL divergence is independent between columns. Throughout this discussion we'll assume we're selecting at most s nonzero entries from each column. In order to select nonzero rows for a single column, we can do this greedily, by picking the row that locally decreases the KL divergence the most. Once we've selected it, we take the next entry according to the same criteria, and so on.

However, if we were to do this naively, we would iterate over the N possible candidate indices k. For each k, we would take the new sparsity pattern $s_i' = s_i \cup \{k\}$, and compute the inverse of the submatrix Θ indexed at this new sparsity pattern, $\Theta_{s_i',s_i'}^{-1}$. Finally, we would pick the k with the largest top left entry and add it to our sparsity pattern. This would cost s^3 per candidate to invert the resulting matrix, over N candidates and s rounds, with a cost of Ns^4 per column and $N^2 s^4$ over all the columns. Luckily, we can do much better by taking advantage of Schur complements.

We can cleverly block our matrix in a way to take advantage of the redundancy in computing $\Theta_{s_i',s_i'}^{-1}$. Note that we can assume we have Θ_{s_i,s_i}^{-1}, the inverse of the entries we've selected already. Then when we consider k as a candidate, we're just adding a single row and column to this new matrix. If we organize everything, our $\Theta_{s_i',s_i'}$ will be:

$$P\Theta_{s_i',s_i'}P^\top = \left(\begin{array}{cc|c} \boxed{\Theta_{11}} & \Theta_{12} & \Theta_{13} \\ \Theta_{21} & \Theta_{22} & \Theta_{23} \\ \hline \Theta_{31} & \Theta_{32} & \Theta_{33} \end{array}\right) \tag{3.5}$$

Here Θ_{33} is the diagonal entry Θ_{ii}, henceforth known as the *special entry*, Θ_{11} are the entries we've taken already (excluding i), and Θ_{22} is Θ_{kk}, the candidate entry. The off diagonal terms all correspond to the proper indexing of $\Theta_{s_i',s_i'}$, i.e. Θ_{21} is Θ at row k indexed along the sparsity pattern s_i, Θ_{31} is Θ at row i indexed along s_i, and Θ_{32} is the scalar Θ_{ik}. The terms above the diagonal are symmetric.

Thus, computing $(\Theta_{s_i',s_i'}^{-1})_{11}$ is equivalent to computing $((P\Theta_{s_i',s_i'}P^\top)^{-1})_{33}$ since inverting a permuted matrix just permutes its inverse. We can efficiently compute the bottom right entry of the block matrix with Schur complementation: write Schur complements as explicit conditioning

$$(S/S_{22}) = (S_{33} - S_{32}S_{22}^{-1}S_{23})^{-1}$$

$$(\Theta_{s_i',s_i'}^{-1})_{11} = \left(\Theta_{33} - \Theta_{31}\Theta_{11}^{-1}\Theta_{13} - \frac{(\Theta_{32} - \Theta_{31}\Theta_{11}^{-1}\Theta_{12})^2}{\Theta_{22} - \Theta_{21}\Theta_{11}^{-1}\Theta_{12}}\right)^{-1}$$

Recall that we want to maximize this term. So we can minimize

$$\equiv \min_k \Theta_{33} - \Theta_{31}\Theta_{11}^{-1}\Theta_{13} - \frac{(\Theta_{32} - \Theta_{31}\Theta_{11}^{-1}\Theta_{12})^2}{\Theta_{22} - \Theta_{21}\Theta_{11}^{-1}\Theta_{12}}$$

Since $\Theta_{33} - \Theta_{31}\Theta_{11}^{-1}\Theta_{13}$ is constant over our candidates,

$$\equiv \max_k \frac{(\Theta_{32} - \Theta_{31}\Theta_{11}^{-1}\Theta_{12})^2}{\Theta_{22} - \Theta_{21}\Theta_{11}^{-1}\Theta_{12}} \tag{3.6}$$

This is precisely the objective in (2.4) as the numerator is the squared covariance between the candidate and the special entry, conditional on the entries already selected, while the denominator is the conditional variance of the candidate. Hence the sparse Cholesky factorization motivated by KL divergence can be viewed as the sparse Gaussian process regression over each column, where entries are selected to maximize mutual information with the entry on the diagonal of the current column. elaborate on the connection and recycle algorithm

be more explicit that the objectives themselves are identical, not just the greedy objectives (from conditioning being Schur complementation being taking the matrix, inverting, taking a submatrix, inverting)

3.1. Algorithms for Efficient Schur Complementation.

There are two primary ways we can efficiently compute these quadratic forms.

3.1.1. Explicit Inverse.

Maintain Θ_{11}^{-1} explicitly. When an index k is added to the sparsity set, efficiently update Θ_{11}^{-1} with Schur complementation in $\mathcal{O}(s^2)$. Finally, compute the quadratic forms by storing the previous values for each candidate and updating based on the fact that adding a new row and column only adds $\mathcal{O}(s)$ terms to compute. For each of the s rounds, it takes s^2 to update the inverse and for each of the N candidates, it takes s to compute their new quadratic form, costing $Ns^2 + s^3$ per column and $\mathcal{O}(N^2 s^2 + N s^3)$ overall.

3.1.2. Cholesky Factorization.

Maintain a Cholesky factorization of Θ_{11}, $\Theta_{11} = LL^\top$. When an index is added, update the Cholesky factorization with left-looking in $\mathcal{O}(Ns)$. Each quadratic form can be updated in $\mathcal{O}(1)$ given the Cholesky factorization, yielding Ns^2 per column and $\mathcal{O}(N^2 s^2)$ overall.

While both approaches have similar time complexity, the explicit inverse algorithm uses $\mathcal{O}(s^2 + N)$ space to store the inverse and quadratic forms while the Cholesky factorization uses $\mathcal{O}(Ns)$ to store the Cholesky factors ($N > s$). Also the explicit inverse algorithm computes Θ_{11}^{-1}, which can be directly used to generate the columns of L according to (3.3) without extra computational cost.

3.2. Review of KL approximation.

4. Numerical experiments.

Acknowledgments. This research was supported in part through research cyberinfrastructure resources and services provided by the Partnership for an Advanced Computing Environment (PACE) at the Georgia Institute of Technology, Atlanta, Georgia, USA.

REFERENCES

[1] A. Krause, A. Singh, and C. Guestrin, *Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies*, J. Mach. Learn. Res., 9 (2008), p. 235–284.

[2] F. Schäfer, M. Katzfuss, and H. Owhadi, *Sparse Cholesky factorization by Kullback-Leibler minimization*, arXiv preprint arXiv:2004.14455, (2020).

[3] F. Schäfer, T. J. Sullivan, and H. Owhadi, *Compression, inversion, and approximate PCA of dense kernel matrices at near-linear computational complexity*, arXiv preprint arXiv:1706.02205, (2017).

add proofs, if any, in appendix

.1. Iterative conditioning as Schur complementation as Cholesky factorization as Orthogonal Matching Pursuit.