Submodular Kernels for Efficient Rankings

Michelangelo Conserva 1  Marc Peter Deisenroth 2  K S Sesh Kumar 3

Abstract

Many algorithms for ranked data become computationally intractable as the number of objects grows due to complex geometric structure induced by rankings. An additional challenge is posed by partial rankings, i.e. rankings in which the preference is only known for a subset of all objects. For these reasons, state-of-the-art methods cannot scale to real-world applications, such as recommender systems. We address this challenge by exploiting geometric structure of ranked data and additional available information about the objects to derive a submodular kernel for ranking. The submodular kernel combines the efficiency of submodular optimization with the theoretical properties of kernel-based methods. We demonstrate that the submodular kernel drastically reduces the computational cost compared to state-of-the-art kernels and scales well to large datasets while attaining good empirical performance.

1. Introduction

Ranked data arises in many real-world applications, e.g., multi-objects tracking (Hu et al., 2020) and preference learning (Fürnkranz & Hüllermeier, 2003). However, the classical application is recommender systems (Karatzoglou et al., 2013), for example for films (Harper & Konstan, 2015) or jokes (Goldberg et al., 2001). In such scenarios, rankings can take different forms (full, top-k or interleaving) and, typically, contain information only for a subset of the objects. The main challenge of machine learning applications for ranked data is a computational one as rankings present a super exponential growth. Given $n$ objects, there are $n!$ possible full rankings and about $\frac{1}{2}(\frac{1}{\log(2)})^{n+1}n!$ partial rankings (Gross, 1962).

Several approaches have been proposed to model ranked data. Balcan et al. (2008) and Ailon et al. (2008) propose to interpret rankings as a sequence of binary decisions between objects whereas Lebanon & Mao (2008), Helmbold & Warmuth (2009) and Huang et al. (2009) define a distribution over permutations. These approaches, however, do not scale when the number of objects is large since they do not exploit the underlying structure of ranked data.

Ranked data, despite being high dimensional, is structured, and being able to compute similarities suffices in many applications. For example, in recommender systems, the similarity of two users based on their preferences is vital to propose algorithms that provide recommendations to them. For this reason, kernels, which can be viewed as measures of similarities, are particularly well suited to model ranked data.

The application of kernels to ranked data started with the diffusion kernel (Kondor & Lafferty, 2002). However, the diffusion kernel is prohibitively expensive to compute in real-world settings as its time complexity scales exponentially with the number of objects. (Jiao & Vert, 2015) proposed the Kendall and Mallows kernels, which are based on the Kendall distance for rankings (Kendall, 1948), and a tractable algorithm to calculate them. These kernels correspond to the linear kernel and the squared exponential kernel on the symmetric group (Diaconis, 1988), which is a particular space in which rankings are viewed as permutations. For a set of $n$ objects, the time complexity for both kernels is $n \log n$ for full rankings. In the specific case of the Kendall kernel, the algorithm can be extended to partial rankings. However, empirically, it is not computationally appealing for large sample sizes. Jiao & Vert (2018) propose a weighted version of the Kendall kernel to model the different contributions of the different pairs in the rankings but the $n \log n$ time complexity algorithm is only valid for full and top-k rankings. Finally, (Lomeli et al., 2019) propose a Monte Carlo kernel estimator for the Mallows kernel, which is only valid for top-k rankings.

The main challenge of applying kernel methods to ranked data is to efficiently deal with all kinds of rankings: full, top-k and interleaving. Previous kernels for ranked data (Jiao & Vert, 2015; Lomeli et al., 2019) have been tailored for a single kind of rankings. Here, we propose the submodular kernel, which can attain good performance at a low computation budget regardless of the specific kind of ranking. To achieve this challenging objective, we interpret ranked
data in terms of ordered partitions, rather than permutations on the symmetric group. This allows us to use the efficient machinery of submodular functions. Submodular functions have been applied to ratings\(^1\) in the form of Lovász Bregman divergences (Iyer & Bilmes, 2013). In this paper, we propose a novel kernel, the submodular kernel, for rankings.

The main contribution of the paper is the yield by submodular functions and is characterised by an in rankings the objects are ordered according to the preference.

In Section 2, we show how to encode all kinds of rankings as ordered partitions and their relationship with submodular functions. In Section 3, we propose a monotonic feature map using the submodular functions that is consistent with the ranking of the objects. Finally, in Section 4, we support our claims and demonstrate scalability empirically using synthetic and a real datasets.

2. Ordered Partitions and Submodular Functions

In Section 2.1, we introduce the different types of rankings and how to encode them as ordered partitions and, in Section 2.2, we revise submodular functions. Further, in Section 2.3, we introduce the base polytope, a geometrical representation of the space of ordered partitions, which is yielded by submodular functions and is characterised by an exponential number of constraints in the number of objects. Section 2.4 presents the tangent cone, an outer approximation of the base polytope, which is instead characterised by constraints that scale linearly in the number of objects. This cheap but representative approximation for rankings will be used to derive the feature map of the submodular kernel in Section 3. It turns out that the monotonicity of the feature map is consistent with the ranking of the objects.

2.1. Ranked Data and Ordered Partitions

Ranked data is often available in different forms. Table 1 contains the most common kinds of rankings objects, where \(a \prec b\) denotes the preference of \(b\) over \(a\). In full rankings, the pairwise ordering between all objects is known. In top-\(k\) rankings, the information on the pairwise ordering is only available for the \(k\) most favourite objects. This implicitly means that the remaining objects are less favourite. Non-exhaustive rankings only contain information on a subset of objects and, therefore, the ordering of the other objects is completely unknown. For instance, in the example given in Table 1, object 3 may lie between object 2 and object 1 or object 1 and object 4.

In this paper, we encode rankings using ordered partitions to achieve a compact and general representation. Let \(\mathcal{V}\) be the set of objects, then an ordered partition can be represented by \(l\) mutually exclusive sets \(A_1, \ldots, A_l\), such that

\[
A_1 \prec \ldots \prec A_l. \tag{1}
\]

If the subsets cover all objects, i.e. \(\bigcup_{i=1}^l A_i = \mathcal{V}\), then we refer to this as an exhaustive ordered partition, otherwise as a non-exhaustive ordered partitions. Full and top-\(k\) rankings are special cases of exhaustive ordered partition. However, non-exhaustive rankings can only be represented by non-exhaustive partial ordering.

In the following, we introduce submodular functions and a specific polytope related to submodular functions whose faces are characterized by ordered partitions. In Section 3, we then define a monotonic feature map that is consistent with a ranking that is represented by the an ordered partition.

2.2. Submodular Functions

Let \(n\) be the number of objects in set \(\mathcal{V}\). A set function defined on the power set \(F: 2^\mathcal{V} \to \mathbb{R}\) is submodular if, for all subsets \(A_i, A_j \subset \mathcal{V}\)

\[
F(A_i) + F(A_j) \geq F(A_i \cup A_j) + F(A_i \cap A_j). \tag{2}
\]

The power set \(2^\mathcal{V}\) is naturally identified by the vertices \(\{0,1\}^n\) of an \(n\)-dimensional unit hypercube. Therefore, a set function \(F: 2^\mathcal{V} \to \mathbb{R}\) defined on the power set of \(\mathcal{V}\) is the same as defining it on the vertices of the \(n\)-dimensional unit hypercube. Each set function may be extended to the complete hypercube \([0,1]^n\) using the Lovász extension (Lovász, 1982), which we refer to as \(f: [0,1]^n \to \mathbb{R}\). For any vector \(w \in [0,1]^n\), we may calculate the value of the function \(f(w)\) and find the permutation of objects \(j_1, \ldots, j_n\), such that \(w_{j_1} \geq \ldots \geq w_{j_n}\). Then

\[
f(w) = w_{j_n} F(\{j_1, \ldots, j_n\}) + \sum_{k=1}^{n-1} (w_{j_k} - w_{j_{k+1}}) F(\{j_1, \ldots, j_k\}). \tag{3}
\]

For example, let us take the set \(\mathcal{V} = \{a, b\}\) with power set \(2^\mathcal{V} = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}\). Each subset may be represented by an indicator vector of length \(|\mathcal{V}| = 2\). The

| TYPE | RANKING | ORDERED PARTITION |
|------|---------|-------------------|
| Full | 2 \prec 1 \prec 3 \prec 4 \prec 5 | \{2\} \prec \{1\} \prec \{3\} \prec \{4\} \prec \{5\} |
| Top-2 | 2 \prec 3 | \{4, 5\} \prec \{2\} \prec \{3\} |
| Exh. | 1 \prec 2 \prec 3 \cap 1 \prec 2 \prec 4 \cap \{1\} \prec \{2, 3\} \prec \{3, 4\} | \{1\} \prec \{2, 3\} \prec \{3, 4\} |
| Non-exh. | 2 \prec 1 \prec 4 | \{2\} \prec \{1\} \prec \{4\} |

Table 1: Examples of rankings from a set of 5 objects.
This extension is piecewise linear for any set function \( F \). For an ordered partition \( A \), we can define the tangent cone of the base polytope \( B(F) \), a form of outer approximation with constraints linear in the length of \( A \).

**Definition 2.2 Tangent cone.** Let \( B_i := \cup_{j=1}^i A_j \). We define the tangent cone of the base polytope \( B(F) \) characterized by the ordered partition \( A \) as

\[
\tilde{B}^A(F) = \left\{ s(B_i) \leq F(B_i) \cap s(V) = F(V), \forall i \in \{1, \ldots, l-1\} \right\},
\]

(7)

The tangent cone considers only \( l-1 \) constraints of the exponentially many constraints used to define the base polytope, and it forms an outer approximation of the ordered partitions as illustrated in Figure 1c and d for \( \{1, 2, 3\} \) and \( \{1\} \prec \{2\} \prec \{3\} \) respectively.

Since the number of constraints of the tangent cone is linear in the length of the ordered partition, it is possible to leverage efficient submodular optimization techniques to retrieve a cheap approximation for ranked data using their tangent cone representations. The submodular kernel is built on top of this approximation as explained in Section 3.1.

### 3. Kernel for Ordered Data

Using the connections between ordered partitions and submodularity (Section 3.1) we propose a feature map for the submodular kernel. Section 3.2 extends the kernel to non-exhaustive ordered partitions, i.e., non-exhaustive rankings.

#### 3.1. From the Feature Map to the Kernel

We propose to use the min-norm point on the tangent cone of the base polytope \( \tilde{B}^A(F) \) as the feature map

\[
\phi(A) := s^* = \arg\min_{s \in \tilde{B}^A(F)} \frac{1}{2} \|s\|^2 \in \mathbb{R}^n,
\]

(8)
Submodular Kernels for Efficient Rankings

where \( n \) is the number of objects. Kumar & Bach (2017) showed that this is dual to the isotonic regression problem

\[
\min_{v \in \mathbb{R}^l} \sum_{i=1}^{l} v_i [F(B_i) - F(B_{i-1})] + \frac{1}{2} \sum_{i=1}^{l} |A_i| v_i^2
\]

s.t. \( v_1 \geq \cdots \geq v_l \), \( (9) \)

where the optimal solution \( s^* \) of Eq. (8) and the optimal solution \( v^* \) of Eq. (9) are related via

\[
s^* = - \sum_{i=1}^{l} v_i^* 1_{A_i}. \tag{10}
\]

Note that \( v_i^* \) corresponds to the set \( A_i \), while \( s_j^* \) corresponds to the object \( j \) and \( s_j^* = -v_j^* \), if object \( j \) belongs to the subset \( A_i \). The isotonic regression problem may be solved using the pool adjacent violator algorithm (PAVA) (Best & Chakravarti, 1990). If the optimal solution \( v^* \) has the same values for some of the partitions, i.e. \( v_i^* = v_{i+1}^* \) for some \( i \in \{1, \ldots, l-1\} \), this corresponds to a merge of \( A_i \) and \( A_{i+1} \). If these merges are made, we obtain a basic ordered partition\(^2\), such that our optimal \( v^* \) has strictly decreasing values. Because none of the constraints are tight, primal stationarity leads to explicit values of \( v^* \) given by

\[
v_i^* = -(F(B_i) - F(B_{i-1}))/|A_i|, \tag{11}\]

i.e., the exact solution to the isotonic regression problem in Eq. (9) may be obtained in closed form.

We now present an example of a feature map for

\[
A = 5 \approx 8 \approx 3 \approx 4 \approx 1 \approx 2 \approx 9 \approx 10 \approx 7 \approx 6
\]

to support the choice of the feature map. Using \( \phi \) allows us to derive a geometrically meaningful kernel.

Figure 2 reports an example of \( v^* \) associated with \( A \). Higher values correspond to more preferred objects and lower values to less preferred ones, whereas objects in the middle tend to be associated with values closer to zero. This particular structure of \( v^* \) is characteristic of submodular functions \( F \). The values \( v^* \) are related to the feature map in Eq. (10) in the sense that each dimension \( d \) of \( \phi(A) \) corresponds to an object, and \( \phi(A)_d \) is the value in \( v^* \) corresponding to \( d \). From the example in Figure 2, \( \phi(A)_2 = -4.22 \), \( \phi(A)_3 = \phi(A)_8 = -2.36 \) and so on. Therefore, a natural and efficient way to exploit this pattern is to use the linear kernel on the resulting feature maps:

\[
k_s(A, A') = \sum_{d=0}^{n} \phi(A)_d \phi(A')_d = \langle \phi(A), \phi(A') \rangle. \tag{12}\]

Figure 2: Optimal values \( v^* \) (Eq. (11)) using submodular cut functions and the information graph of the sushi dataset for \( A \).

(a) Values for each dimension \( d \) of the 10 dimensional feature maps \( \phi(A) \) and \( \phi(A') \).

(b) Multiplication values for each dimension of \( \phi(A) \) and \( \phi(A') \).

Figure 3: Geometrical interpretation of the linear kernel of feature map \( \phi(A) \) and \( \phi(A') \), where \( A' \) is the inverse of \( A \).

If in two rankings \( A \) and \( A' \), an object \( d \) is ranked similarly then this results in the multiplication of two numbers with the same sign, i.e. \( \phi(A)_d \phi(A')_d > 0 \). On the contrary, if \( d \) is ranked differently then the multiplication would happen between numbers with different signs and thus \( \phi(A)_d \phi(A')_d < 0 \). How much greater or less than zero depends on the position of the \( d \) in the rankings. Figure 3 reports \( \phi(A) \) and \( \phi(A') \) where \( A' \) is the inverse of \( A \) along with the multiplication factor \( \phi(A)_d \phi(A')_d \).

3.2. Kernel for Non-Exhaustive Partial Orderings

To extend the submodular kernel from exhaustive to non-exhaustive ordered partitions, we recur to the convolution kernel (Haussler, 1999) by defining the set of exhaustive order partitions that are coherent with a non-exhaustive ranking, \( R(\cdot) \). The convolution kernel over the submodular kernel is

\[
k_c(A, A') = \frac{1}{|R(A)||R(A')|} \sum_{a \in R(A)} \sum_{a' \in R(A')} k_s(a, a'), \tag{13}\]

where \( A, A' \) are non-exhaustive rankings ordered partitions and \( R(A), R(A') \) are the set of exhaustive ordered partitions that are coherent with \( A, A' \) respectively. Due to the linearity of the inner product, we can reduce \( k_c(\cdot, \cdot) \) to the
inner product between the average of the feature maps over $R(A)$ and $R'(A')$, so that
\[
k_s(A, A') = \frac{1}{|R(A)||R'(A')|} \sum_{a \in R(A)} \sum_{a' \in R'(A')} k_s(a, a') = \frac{1}{|R(A)||R'(A')|} \sum_{a \in R(A)} \sum_{a' \in R'(A')} \langle \phi(a), \phi(a') \rangle = \langle \hat{\phi}(A), \hat{\phi}(A') \rangle, \tag{14}
\]
where $\hat{\phi}(A)$ is the mean of the feature map over the set of exhaustive ordered partitions that are coherent with $A$. Similarly to $k_s$, we can precompute these feature maps.

The main drawback of using the convolution kernel for non-exhaustive rankings is that the time complexity for calculating the feature maps becomes exponential due to the exponential growth of $R(A)$ in the number of objects. Given an interleaving ranking $A$ of length $l$ for $n$ objects, $R(A)$ has cardinality $(l + 1)^{n - 1}$. In this case, we can sample from the set of coherent ordered partitions and calculate the average feature map on the sampled subset of $R(A)$. This approximation allows us to scale well also for large sets of items. In Section 4, we show that we empirically obtain competitive results using this sampling procedure that removes the burden of the computational complexity.

3.3. Submodular kernel algorithm

In summary, the algorithm to calculate the submodular kernel happens in two steps. (1) Given a submodular function, we extract low dimensional features from the data using the Pool Adjacent Violator Algorithm (PAVA) 1, and (2) we use the computationally cheap linear kernel. Assuming an oracle submodular function with time complexity $O(p)$ and an exhaustive ordered partition of length $l$, the time complexity for calculating the Gram matrix using the submodular kernel is $O(smpl + m^2 n)$. Note that these complexities hold only if all the rankings are exhaustive ordered partitions of at most size $l$. The time complexity for Gram matrix for rankings that are non-exhaustive is given by $O(smpl + m^2 n)$, where $s$ is the number of samples when we use sampling as explained in Section 3.2. In the next session, we empirically show that the submodular kernel is, at the same time, able to match or outperform state-of-the-art performance in an established classification task for kernel-based methods.

4. Experiments

The goal of this section is to demonstrate (a) scalability, (b) accuracy and (c) ability for rankings of all types. We use a downstream classification task, similar to what (Jiao & Vert, 2015) and (Lomeli et al., 2019).

Algorithm 1 PAVA(A, F)

Input:
- Ordered partition $A$ of length $l$.
- Submodular function $F$.

1. Define an set to contain the optimal values $k \leftarrow l$. // To retrieve sets $B_{k-1}$ and $B_k$

2. while $|v| < |A|$ do
   3. $B_{k-1} \leftarrow \cup_{i=1}^{k-1} A_i$ // as in Eq. (7)
   4. $B_k \leftarrow \cup_{i=1}^{k} A_i$ // as in Eq. (7)
   5. $v' \leftarrow -\frac{F(B_{k-1}) - F(B_k)}{|B_{k-1}| - |B_k|}$ // optimal solution Eq. (11)

   6. if $|v| = 0$ and $v' < v.first()$ then
      7. MERGE(A[k - 1],A[k]) // Merging items
      8. v.removeFirst() // Removing last value

   9. else
      10. v.insert(v') // Inserting value in the set
      11. $k \leftarrow k - 1$ // One less

12. end if

13. end while

14. // Returning the set of optimal values associated to $A$
15. Return $v$

Synthetic Dataset. The synthetic dataset resembles food preferences of eight dishes partitioned in three groups based on the numerical features, which describe them: Sweetness, Savouriness and Juiciness:

\[
\{\text{Cake, Biscuit, Gelato}\}, \{\text{Pasta, Pizza}\}, \{\text{Steak, Burger, Sausage}\}.
\]

There is a clear distinction between the first group and the others and a mild distinction between the second and the third one. Suppose that there are two types of users with opposite preferences \{Sweet $\gg$ Savouriness $\gg$ Juicy\} and \{Juicy $\gg$ Savouriness $\gg$ Sweet\}, the classification task is to distinguish the types of users by their preferences. A parameter $\sigma$ controls the level of noise in the ranking sampling procedure. More details on the dataset and the sampling procedure are available in the supplementary material.

Real Dataset. The state-of-the-art dataset for rankings data is the sushi dataset (Kamishima, 2003). Japanese users are asked to rank two different sets of sushis. In the first set, there are 10 sushis and the users express full rankings whereas in the second set there are one hundred sushis and the users express non-exhaustive rankings of ten of them. We refer to them as small and large sushi datasets. The preference in sushis varies between the two parts of the country so the origin of the person can meaningfully be the label for a classification task (Kamishima, 2003).
4.1. Classification
To assess the generalization capability of the submodular kernel and to compare it with the state-of-the-art kernels, we train Gaussian process (GP) classifiers. For the synthetic dataset, we choose a sample size of 250 for full and top-k rankings and, due to the computational cost of the Kendall kernel, 100 for exhaustive interleaving rankings. For both the small and the large sushi datasets, we use 2500 rankings.

Experimental Setting. Samples are split into train 80% and test sets 20%. Results are reported in terms of F1-scores (Chinchor, 1992), which provides a reliable estimate of the actual capabilities of the model as it penalizes the classifier for false positives and false negatives. To train the GP classifier we use the python package GPflow (Matthews et al., 2017), which we extended with custom classes. The submodular function used in the experiment is the cut function applied to a graph build from the objects, i.e., the dishes in the synthetic dataset and the sushi in the real dataset. The edges of such graph are calculated using the squared exponential kernel as the similarity score function, i.e.,

\[ e(v, v') = \exp\left(-\frac{1}{2} \frac{\|v - v'\|^2}{\ell^2}\right). \]  

The lengthscale \( \ell \) is selected using the inverse median heuristic (Schölkopf & Smola, 2002). For the large sushi dataset, we use 600 samples from the set of exhaustive ordered partitions \( R \) and we retain the top 10% edges of the graph.

Synthetic Dataset. The synthetic dataset is built to show the ability of the kernels to handle noisy data. Figure 4 reports the F1-scores on the test sets obtained by the GP classifiers for increasing levels of noise. The submodular kernel performs in line with state-of-the-art kernels and, for the specific case of top-6 rankings, Figure 4b, slightly better. We note that the performance of submodular kernel for full and for the Top-6 rankings matches. This shows that, given a mild level of censoring as in the Top-6 scenario, the submodular function on the information graph can successfully reconstruct the underlying preferences. The Mallows kernel exhibits a sharp drop in performance in the top-6 scenario and slightly worse performance in the top-3 one, Figure 4b. For both the exhaustive (Figure 4d) and non-exhaustive interleaving rankings (Figure 4e), the submodular kernel matches the performance of the state-of-the-art Kendall kernel. The last two mentioned scenarios present a higher level of variability in performance. For the exhaustive interleaving rankings scenario, this is caused by the small sample size of 100, whereas for interleaving rankings, the variability is due to the task difficulty.

Real Dataset. The sushi dataset is becoming the benchmark dataset to show the generalization capabilities of kernel methods on rankings, (Kondor & Lafferty, 2002; Lomeli et al., 2019). For this challenging classification task the performance on the test set, regardless of the kernel, is limited to an F1-score of 0.63. Nonetheless, our submodular kernel slightly outperforms the state-of-the-art kernels both in terms of performance and stability over different seeds. More specifically, in the full rankings scenario reported in Figure 5a, the submodular kernel’s F1-scores on the test set are significantly better than random performance. This is not true for the top-6 and interleaving scenarios, Figure 5b and c. However, the submodular kernel still manages to get a better than random performance, unlike the Mallows kernel b and the Kendall kernel c and d.

Using synthetic and a real datasets we demonstrated that the submodular kernel works as well as state-of-the-art kernels used for ranking.

4.2. Empirical Time Complexity
In the following, we compare the empirical time complexity for all kernels and highlight the steep advantage of the submodular kernel. For algorithms meant to be employed in real-world applications, it is essential to analyze the empirical time complexity of computing the Gram matrix, in addition to the theoretical one that was covered in Section 3.
In this section, we provide this empirical analysis using rankings sampled from the synthetic dataset.

**Experimental Setting.** The values for the empirical time complexity are averaged over seven trials on a standard laptop\(^1\) only using CPU. On the contrary of state-of-the-art methods, for the submodular kernel, we can easily use GPU hardware acceleration to calculate the Gram matrix, however, in order to present a fair comparison, we limit the submodular kernel calculation on CPU only. The accessibility of hardware acceleration for the submodular kernel is immediate since it is built on top of the linear kernel for which software as GFlow provide built-in highly optimized implementations. In other words, after obtaining the feature maps for the rankings, the user can directly feed them into the software that optimizes all the computations. To give an idea of the scale of such optimization we report the total time required to calculate the Gram matrix for the entire dataset, i.e. 5000 samples. We found that the time required to calculate the feature maps is 6.31s ± 88.6ms, while the time required to compute the Gram matrix from the feature maps is 576ms ± 19.7ms with hardware acceleration using a single Nvidia GeForce GTX 1060 GPU. Considering that the calculation of the feature maps does not happen when fitting the model, it is clear that the submodular kernel can be confidently employed for large scale datasets. To the best of our knowledge, this is the first time it has been possible to calculate the Gram matrix of a rankings dataset of this scale in less than a second using an ordinary laptop.

**Experiments.** Figure 6 reports the time spent for computing the Gram matrix as a function of the sample size and, for partial rankings, also of the level of censoring. For all kinds of exhaustive interleaving rankings (Figure 6a, b and c) computing the Gram matrix of the submodular kernel is extremely cheap compared to state-of-the-art kernels. For the interleaving case, Figure 6d, due to the exponential factor, the time required is high but still significantly lower than the time required to compute the Gram matrix using the Kendall kernel. In Figure 6c some values for the Kendall kernel are not reported. This is because the Kendall kernel becomes computationally intractable in practice when a high number of ties are present in the rankings as it happens for low values of exhaustive interleaving ranking length. The massive speed advantage of the submodular kernel comes from the fact that the computationally intensive calculations are done once per feature map and that the space complexity of storing the feature maps is linear both in the sample size and in the number of object\(^2\). In practice, the submodular kernel scales almost linearly with the sample size.

---

\(^1\)We use a laptop mounted with an Intel Core(TM) i7-7700HQ CPU @ 2.80GHz and 16GB of RAM.

\(^2\)The time for calculating the feature maps is included in the presented empirical time complexity for the submodular kernel.
5. Discussion

The submodular kernel is a theoretically founded kernel that builds on top of the efficient machinery of submodular functions optimizations. Thanks to this efficiency, we have shown a drastic reduction in the computational cost compared to previous kernel-based methods for ranked data. Additionally, it inherits all the theoretical properties of the linear kernel since it is built on top of it.

Contrary to the state-of-the-art kernels for ranked data, the submodular kernel requires an additional choice of an application-specific submodular function. Such a choice can provide additional flexibility as practitioners can choose the specific submodular function that fits best their application, but it can also provide additional challenges for people with lack of knowledge of submodular functions. Finally, the submodular kernel does not exploit the kernel trick as this is left for future work.

6. Conclusion

In this paper, we presented a theoretically founded methodology to apply submodular function optimization for ranked data of all kinds: full, top-k and interleaving. This allowed us to define the submodular kernel that attains good empirical performance while only incurring fairly small computational costs. Thanks to the submodular kernel, we enabled the use of kernel-based methods for large scale datasets of rankings with the hope of encouraging a more widespread use of kernel methods for rankings. We support our claims by empirically training Gaussian process classifiers using both synthetic and real datasets and reporting the empirical time complexity. The submodular kernel has proved to match or outperform the performance of the state-of-the-art kernels both in terms of accuracy on the classification task and of clock time.

We believe that the submodular kernel will greatly benefit both practitioners, as they now have access to a wider range of tools for real-world applications, and researchers, as we have consolidated the theoretical connection between submodular functions optimization and ranked data. We have thus opened several exciting research directions. For example, extending the presented framework from ranking to ratings or consider specific submodular functions to further reduce the computational cost of the kernel.

Figure 6: Train/test procedure for GP repeated over six seeds for the sushi dataset. The small sushi dataset is employed for full, top-6 and exhaustive rankings, whereas the large sushi dataset is used for non-exhaustive rankings. For each kind of rankings, we report the evidence lower bound (ELBO) during train time (Left) and the F1-scores calculated on the test sets with a uniform dummy classifier baseline (Right). Note that the x-axis for the ELBO plot is in logarithmic scale and mean, first quartile and third quartile are reported in bright colours in the violin plots.
Submodular Kernels for Efficient Rankings

References

Ailon, N., Charikar, M., and Newman, A. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM*, 55(5):1–27, 2008.

Bach, F. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends in Machine Learning*, 6(2-3):145–373, 2013.

Balcan, M.-F., Bansal, N., Beygelzimer, A., Coppersmith, D., Langford, J., and Sorkin, G. B. Robust reductions from ranking to classification. *Machine Learning*, 72(1-2):139–153, 2008.

Best, M. J. and Chakravarti, N. Active set algorithms for isotonic regression: a unifying framework. *Mathematical Programming*, 47(1):425–439, 1990.

Chinchor, N. Muc-4 evaluation metrics. *Proceedings of the Conference on Message Understanding*, pp. 22–29, 1992.

Diaconis, P. Group representations in probability and statistics. *Lecture Notes*, 11:i–192, 1988.

Fujishige, S. *Submodular Functions and Optimization*. Elsevier, 2005.

Furnkranz, J. and Hüllermeier, E. Pairwise preference learning and ranking. In *Proceedings of the European Conference on Machine Learning*, pp. 145–156, 2003.

Goldberg, K., Roeder, T., Gupta, D., and Perkins, C. Eigen-taste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

Gross, O. A. Preferential arrangements. *The American Mathematical Monthly*, 69(1):4–8, 1962. ISSN 00029890, 19300972.

Harper, F. M. and Konstan, J. A. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5(4), December 2015. ISSN 2160-6455. doi: 10.1145/2827872.

Haussler, D. Convolution kernels on discrete structures. Technical report, Technical report, Department of Computer Science, University of California, 1999.

Helmbold, D. P. and Warmuth, M. K. Learning permutations with exponential weights. *Journal of Machine Learning Research*, 10(7):1705–1736, 2009.

Hu, T., Huang, L., and Shen, H. Multi-object tracking via end-to-end tracklet searching and ranking. *arXiv preprint arXiv:2003.02795*, 2020.

Huang, J., Guestrin, C., and Guibas, L. Fourier theoretic probabilistic inference over permutations. *Journal of Machine Learning Research*, 10(5):997–1070, 2009.

Iyer, R. and Bilmes, J. The lovász-bregman divergence and connections to rank aggregation, clustering, and web ranking. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 321–330, 2013.

Jiao, Y. and Vert, J.-P. The Kendall and Mallows kernels for permutations. In *Proceedings of the International Conference on Machine Learning*, pp. 1935–1944, 2015.

Jiao, Y. and Vert, J.-P. The weighted Kendall and high-order kernels for permutations. In *Proceedings of the International Conference on Machine Learning*, pp. 2314–2322, 2018.

Kamishima, T. Nantonac collaborative filtering: recommendation based on order responses. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pp. 583–588, 2003.

Karatzoglou, A., Baltrunas, L., and Shi, Y. Learning to rank for recommender systems. In *Proceedings of the Conference on Recommender Systems*, pp. 493–494, 2013.

Kendall, M. G. *Rank Correlation Methods*. Griffin, 1948.

Kondor, R. and Lafferty, J. Diffusion kernels on graphs and other discrete structures. In *Proceedings of the International Conference on Machine Learning*, pp. 315–22, 2002.

Kumar, K. S. S. and Bach, F. Active-set methods for submodular minimization problems. *Journal of Machine Learning Research*, 18(1):4809–4839, 2017.

Lebanon, G. and Mao, Y. Non-parametric modeling of partially ranked data. *Journal of Machine Learning Research*, 9:2401–2429, 2008.

Lomeli, M., Rowland, M., Gretton, A., and Ghahramani, Z. Antithetic and Monte Carlo kernel estimators for partial rankings. *Statistics and Computing*, 29(5):1127–1147, 2019.

Lovász, L. Submodular functions and convexity. *Mathematical Programming: The State of the Art*, pp. 235–257, 1982.

Matthews, A. G. d. G., van der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. GPlow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, 2017.

Rockafellar, R. T. *Convex Analysis*. Princeton University Press, 1997.

Schölkopf, B. and Smola, A. J. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.
A. Duality between isotonic regression and min-norm-point on the tangent cone

Let us recall that \( F : 2^V \to \mathbb{R}_+ \) is a submodular functions and let \( f : \mathbb{R}^n \to \mathbb{R}_+ \) be its Lovász extension. We represent the base polytope of \( F \) by \( B(F) \). Given an ordered partition \( A = (A_1, \ldots, A_l) \) of the set of objects \( V \), we recall the definition of tangent cone of the base polytope and derive its support function. Note that it is already derived by (Kumar & Bach, 2017) that we re-derive here for completion.

A.1. Tangent cone and its support function.

Let \( B_i := \bigcup_{j=1}^i A_j \). We define the tangent cone of the base polytope \( B(F) \) characterized by the ordered partition \( A \) as

\[
\hat{B}^A(F) = \left\{ s(B_k) \leq F(B_k) \cap s(V) = F(V), \quad \forall i \in \{1, \ldots, l-1\} \right\}.
\]

(16)

We can now proceed to compute the support function of the tangent cone \( \hat{B}^A(F) \), which is an upper bound on \( f(w) \) since this set is an outer approximation of \( B(F) \).

\[
\sup_{s \in \hat{B}^A(F)} w^\top s = \sup_{s \in \mathbb{R}^n} \inf_{\lambda \in \mathbb{R}^{l-1}_+ \times \mathbb{R}} w^\top s - \sum_{i=1}^m \lambda_i s(B_i) - F(B_i) \text{, using Lagrangian duality,}
\]

(17)

\[
= \inf_{\lambda \in \mathbb{R}^{l-1}_+ \times \mathbb{R}} \sup_{s \in \mathbb{R}^n} s^\top \left( w - \sum_{i=1}^l (\lambda_i + \cdots + \lambda_l) 1_{A_i} \right) + \sum_{i=1}^l (\lambda_i + \cdots + \lambda_l) [F(B_i) - F(B_{i-1})],
\]

(18)

\[
= \inf_{\lambda \in \mathbb{R}^{l-1}_+ \times \mathbb{R}} \sum_{i=1}^l (\lambda_i + \cdots + \lambda_l) [F(B_i) - F(B_{i-1})]
\]

such that \( w = \sum_{i=1}^l (\lambda_i + \cdots + \lambda_l) 1_{A_i} \).

(19)

(20)

Thus, by defining \( v_i = \lambda_i + \cdots + \lambda_l \), which are decreasing, the support function is finite for \( w \) having ordered level sets corresponding to the ordered partition \( A \) (we then say that \( w \) is compatible with \( A \)). In other words, if \( w = \sum_{i=1}^l v_i 1_{A_i} \), the support functions is equal to the Lovász extension \( f(w) \). Otherwise, when \( w \) is not compatible with \( A \), the support function is infinite.

Let us now denote \( \mathcal{W}^A \) as a set of all weight vectors \( w \) that are compatible with the ordered partition \( A \). This can be defined as

\[
\mathcal{W}^A = \left\{ w \in \mathbb{R}^n \mid \exists v \in \mathbb{R}^l, w = \sum_{i=1}^l v_i 1_{A_i}, v_1 \geq \cdots \geq v_l \right\}.
\]

(21)

Therefore,

\[
\sup_{s \in \hat{B}^A(F)} w^\top s = \begin{cases} f(w) & \text{if } w \in \mathcal{W}^A, \\ \infty & \text{otherwise}. \end{cases}
\]

(22)

A.2. Duality

For all \( w \in \mathcal{W}^A \), i.e., \( w = \sum_{i=1}^l v_i 1_{A_i}, v_1 \geq \cdots \geq v_l \).

\[
f(w) + \frac{1}{2} \|w\|^2 = \sum_{i=1}^l v_i [F(B_i) - F(B_{i-1})] + \frac{1}{2} \sum_{i=1}^l |A_i| v_i^2.
\]

(23)

Let us now consider the minimization of the above optimization problem. It can be rewritten as a weighted isotonic regression problem of the form

\[
\min_{\alpha \in \mathbb{R}^l, \alpha_1 \leq \cdots \leq \alpha_n} \sum_{i=1}^l \beta_i (\alpha_i - a_i)^2.
\]

(24)
where $a \in \mathbb{R}^l$ and $\beta \in \mathbb{R}^l_+$ are given. Note the change of direction of monotonicity from $v$ to $\alpha$ for a standard isotonic regression setting. We now derive the dual of the above problem.

$$
\min_{w \in W^A} f(w) + \frac{1}{2}\|w\|^2 \quad \text{Eq. (22)}
$$

$$
= \min_{w \in \mathbb{R}^n} \max_{s \in B^A(F)} s^T w + \frac{1}{2}\|w\|^2 = \max_{s \in B^A(F)} \min_{w \in \mathbb{R}^n} s^T w + \frac{1}{2}\|w\|^2 = \max_{s \in B^A(F)} -\frac{1}{2}\|s\|^2;
$$

where $s^* = -w^*$ at optimal and the dual problem, $\max_{s \in B^A(F)} -\frac{1}{2}\|s\|^2$ is to find the min-norm-point on the tangent cone of the base polytope.

The isotonic regression problem of the form Eq. (24) may be optimized using the PAVA algorithm proposed by (Best & Chakravarti, 1990) in time $O(l)$ time complexity. This has also been used in the context of submodular function minimization (Appendix A.3 of (Bach, 2013; Kumar & Bach, 2017)).

Algorithm 2 PAVA($y$) → argmin$_{x \in \mathbb{R}^l} \|x - y\|^2$ where $x_1 \leq \ldots \leq x_l$.

1: Initialization
2: $a'_1 \leftarrow a_1$
3: $S_0 \leftarrow 0$
4: $S_1 \leftarrow 1$
5: $j \leftarrow 1$
6: Procedure
7: for $i = 2$ to $n$ do
8: $a'_j \leftarrow a_i$
9: while $j > 1$ and $a'_j < a'_{j-1}$ do
10: $a'_{j-1} \leftarrow \frac{a'_{j} + a'_{j-1}}{2}$
11: $j \leftarrow j - 1$
12: end while
13: $S_j \leftarrow i$
14: end for
15: $x \leftarrow y$
16: // Returning
B. Synthetic Dataset

In this section, we describe the synthetic dataset used in the experiments.

Recall that the synthetic dataset resembles food preferences given eight dishes partitioned in three major groups based on the numerical features which describe them, *Sweetness*, *Savouriness* and *Juiciness*:

$$\{\text{Cake, Biscuit, Gelato}\}, \{\text{Pasta, Pizza}\} \text{ and } \{\text{Steak, Burger, Sausage}\}.$$ 

The exact values for the features are reported in Table 2. There is a marked distinction between the first group and the others and a mild distinction between the second and the third ones.

| Feature | Cake | Biscuit | Gelato | Steak | Burger | Sausage | Pasta | Pizza |
|---------|------|---------|--------|-------|--------|---------|-------|-------|
| Sweet   | 0.9  | 0.7     | 1.0    | 0.0   | 0.2    | 0.1     | 0.4   | 0.4   |
| Savouriness | 0.0  | 0.1     | 0.0    | 0.8   | 0.8    | 1.0     | 0.7   | 0.9   |
| Juicy   | 0.3  | 0.0     | 0.7    | 0.8   | 0.9    | 1.0     | 0.7   | 0.6   |

Table 2: Numerical features for the objects of the synthetic dataset.

Also kindly recall that we suppose that there are two types of users with opposite preferences \(\{\text{Sweet} \succ \text{Savouriness} \succ \text{Juicy}\}\) and \(\{\text{Sweet} \prec \text{Savouriness} \prec \text{Juicy}\}\) to which we refer as *type one* and *type two*. The classification task is to distinguish the types of users by their preferences.

The main objective of this dataset is to reproduce a synthetic setting in which the features of the objects play an important role in determining the rankings. A probabilistic model that expresses such a setting is not available. We thus propose the following sampling procedure assuming that the two mentioned types of users rank the objects. We assume that both *type one* and *type two* users give different and identical importance to the features of the objects when ranking them. We choose the weights for the most preferred feature to be 1, for the second 0.17 and for the last one 0.09 to reproduce an exponential decay in the preference. For example, *type one* presents a unitary importance weight for *Sweet*, a 0.17 importance weight for *Savouriness* and a 0.09 importance weight for *Juicy*. By summing the numerical features of the objects given in Table 2 using the importance weights of the two types of users we obtained two sets of scores for the objects, which Table 3 reports.

| Type | Cake | Biscuit | Gelato | Steak | Burger | Sausage | Pasta | Pizza |
|------|------|---------|--------|-------|--------|---------|-------|-------|
| One  | 0.93 | 0.72    | 1.06   | 0.21  | 0.42   | 0.36    | 0.58  | 0.6   |
| Two  | 0.38 | 0.08    | 0.79   | 0.93  | 1.05   | 1.18    | 0.85  | 0.79  |

Table 3: Scores given to the food objects by the two different types of users.

From the table, it is easy to see that the scores given to the objects greatly vary for the two types of users and they have a strong connection with the underlying feature preferences. Finally, to sample the rankings we add Gaussian noise with \(\sigma\) standard deviation, which we previously called noise parameter, to the scores and return the vector of indices sorted according to the jittered scores.

Formally,

\[
\text{Ranking}_1 \sim \text{ArgSort} \left( \text{Score}_1 + \mathcal{N}(0, \sigma^2 I_8) \right) \in S_8 \\
\text{Ranking}_2 \sim \text{ArgSort} \left( \text{Score}_2 + \mathcal{N}(0, \sigma^2 I_8) \right) \in S_8,
\]

where \(S_8\) is the permutation space yielded by the eight objects in the dataset, \(\text{ArgSort}\) returns the indices the list in input and \(\text{Score}_{user}\) refers to the scores as in Table 3.

In Table 4 we report the underlying noise-free preferences of the two types of users which can be derived from Table 3. The preferences are expressed in increasing order, i.e. zero is associated with the least favourite object and 7 to the most preferred.

We now present a set of rankings sampled according to the previously described procedure with two different values for the noise parameter for the two types of users. In Table 7, we set \(\sigma = 3\) whereas in Table 8, \(\sigma = 0.5\). From analyzing the
Submodular Kernels for Efficient Rankings

|          | Cake | Biscuit | Gelato | Steak | Burger | Sausage | Pasta | Pizza |
|----------|------|---------|--------|-------|--------|---------|-------|-------|
| Type one | 6    | 5       | 7      | 0     | 2      | 1       | 3     | 4     |
| Type two | 1    | 0       | 2      | 5     | 6      | 7       | 4     | 3     |

Table 4: Noise-free objects preferences for the two types of users. Higher values correspond to more preferred objects.

Presented tables it is clear that, for a high value of $\sigma$, the rankings become hardly distinguishable whereas for a low value of noise it is possible to identify the underlying difference between the users. In the experiments we investigate values of noise between 0.1 and 3.

| cake  | biscuit | gelato | steak | burger | sausage | pasta | pizza |
|-------|---------|--------|-------|--------|---------|-------|-------|
| 6     | 3       | 2      | 4     | 1      | 0       | 7     | 5     |
| 3     | 7       | 1      | 2     | 6      | 0       | 4     | 5     |
| 6     | 2       | 7      | 0     | 5      | 4       | 3     | 1     |
| 5     | 3       | 6      | 2     | 1      | 4       | 0     | 7     |
| 4     | 3       | 1      | 6     | 5      | 2       | 0     | 7     |

Table 5: Type one users.

| cake  | biscuit | gelato | steak | burger | sausage | pasta | pizza |
|-------|---------|--------|-------|--------|---------|-------|-------|
| 1     | 7       | 2      | 5     | 3      | 6       | 4     | 0     |
| 2     | 0       | 7      | 6     | 1      | 5       | 4     | 3     |
| 0     | 7       | 1      | 3     | 2      | 6       | 5     | 4     |
| 2     | 0       | 7      | 1     | 4      | 3       | 6     | 5     |
| 0     | 5       | 7      | 2     | 3      | 4       | 6     | 1     |

Table 6: Type two users.

| cake  | biscuit | gelato | steak | burger | sausage | pasta | pizza |
|-------|---------|--------|-------|--------|---------|-------|-------|
| 5     | 4       | 7      | 0     | 6      | 1       | 3     | 2     |
| 3     | 2       | 5      | 4     | 7      | 0       | 6     | 1     |
| 3     | 7       | 2      | 5     | 0      | 6       | 1     | 4     |
| 2     | 7       | 3      | 6     | 4      | 0       | 5     | 1     |
| 1     | 7       | 3      | 2     | 0      | 6       | 4     | 5     |

(a) Type one users.

| cake  | biscuit | gelato | steak | burger | sausage | pasta | pizza |
|-------|---------|--------|-------|--------|---------|-------|-------|
| 2     | 4       | 1      | 0     | 6      | 5       | 7     | 3     |
| 1     | 5       | 7      | 0     | 3      | 2       | 6     | 4     |
| 4     | 7       | 2      | 6     | 0      | 5       | 1     | 3     |
| 5     | 1       | 3      | 0     | 6      | 2       | 7     | 4     |
| 0     | 7       | 3      | 1     | 6      | 5       | 2     | 4     |

(b) Type two users.

Table 7: Sampled rankings in a low noise scenario, i.e $\sigma = 0.5$. Each row represents a different sampled ranking and each column corresponds to the position in the rankings. Higher values correspond to more preferred objects.

| cake  | biscuit | gelato | steak | burger | sausage | pasta | pizza |
|-------|---------|--------|-------|--------|---------|-------|-------|
| 6     | 3       | 2      | 4     | 1      | 0       | 7     | 5     |
| 3     | 7       | 1      | 2     | 6      | 0       | 4     | 5     |
| 6     | 2       | 7      | 0     | 5      | 4       | 3     | 1     |
| 5     | 3       | 6      | 2     | 1      | 4       | 0     | 7     |
| 4     | 3       | 1      | 6     | 5      | 2       | 0     | 7     |

(a) Type one users.

| cake  | biscuit | gelato | steak | burger | sausage | pasta | pizza |
|-------|---------|--------|-------|--------|---------|-------|-------|
| 1     | 7       | 2      | 5     | 3      | 6       | 4     | 0     |
| 2     | 0       | 7      | 6     | 1      | 5       | 4     | 3     |
| 0     | 7       | 1      | 3     | 2      | 6       | 5     | 4     |
| 2     | 0       | 7      | 1     | 4      | 3       | 6     | 5     |
| 0     | 5       | 7      | 2     | 3      | 4       | 6     | 1     |

(b) Type two users.

Table 8: Sampled rankings in a high noise scenario, i.e $\sigma = 3$. Each row represents a different sampled ranking and each column corresponds to the position in the rankings. Higher values correspond to more preferred objects.