The Graph Cut Kernel for Ranked Data

Anonymous authors
Paper under double-blind review

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 kernel for ranking based on the graph cut function. The graph cut kernel combines the efficiency of submodular optimization with the theoretical properties of kernel-based methods. We demonstrate that our novel kernel drastically reduces the computational cost while maintain the same accuracy of state-of-the-art methods.

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 movies (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 computational 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 the space yielded by interpreting rankings 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 $O(n \log n)$ 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 for the specific case of top-k rankings. The main

	Full	Top-k	Exhaustive	Non-exhaustive
Mallows kernel	✓	✓	✗	✗
Kendall kernel	✓	✓	✓	✗
Graph cut kernel (ours)	✓	✓	✓	✓

Table 1: Kernel for the different kinds of rankings data.

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; see Table 1 for an overview of the state of the art.

In this paper, we propose the graph cut 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, which the graph cut function is an example of, have been applied to *ratings*[1] in the form of Lovász Bregman divergences (Iyer & Bilmes, 2013). In this paper, we propose a novel kernel for *rankings*. The main contribution of the paper is the graph cut kernel for all kinds of rankings (full, top-k, exhaustive interleaving and non-exhaustive interleaving). The graph cut kernel exploits the geometric structure of ranked data to drastically reduce the computational cost while still retaining good empirical performance.

The structure of the article is as follows. In Section 2, we show how to encode all kinds of rankings as ordered partitions and their relationship with submodular functions, in particular with the graph cut function. 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 yield by submodular functions and is characterised by an exponential number of constrains 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 graph cut 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 2 contains the most common kinds of rankings

Table 2: Examples of rankings from a set of 5 objects.

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	$2 \prec 3$	$\{4, 5, 1\} \prec \{2\} \prec \{3\}$
Exh.	$1 \prec 2 \prec 3 \cap 1 \prec 2 \prec 4 \cap$ $1 \prec 5 \prec 3 \cap 1 \prec 5 \prec 4$	$\{1\} \prec \{2, 5\} \prec \{3, 4\}$
Non-exh.	$2 \prec 1 \prec 4$	$\{2\} \prec \{1\} \prec \{4\}$

[1]Ratings differ from rankings in the way the preference is expressed. Ratings are defined using a scale of preference, whereas in rankings the objects are ordered according to the preference.

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 order of the remaining objects is unknown. Non-exhaustive rankings only contain information on a subset of objects and, therefore, the ordering of the other objects is unknown. For instance, in the example of non-exhaustive ranking given in Table 2, 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. $\cup_{i=1}^{l} A_i = \mathcal{V}$, 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

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

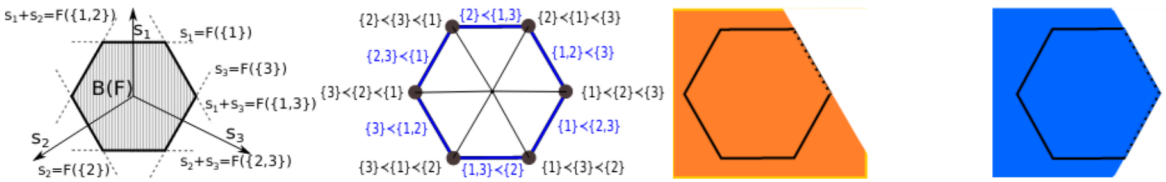
For example, let us take the set $\mathcal{V} = \{a, b\}$ with power set $2^{\mathcal{V}} = \{\varnothing, \{a\}, \{b\}, \{a, b\}\}$. Each subset may be represented by an indicator vector of length $|\mathcal{V}| = 2$. The first index of the indicator vector of the subset represents the presence of the object $\{a\}$ in it while the second index represents the presence of the object $\{b\}$. Therefore, the corresponding indicator vectors $\{[0,0]^T, [1,0]^T, [0,1]^T, [1,1]^T\}$ form the corners of the two-dimensional unit hypercube $[0,1]^2$. If $w = \begin{pmatrix} w_a \\ w_b \end{pmatrix}$, then the Lovász extension f is

$$f(w) = \begin{cases} (w_b - w_a) F(\{b\}) + w_a F(\{a, b\}) & \text{if } w_b \geq w_a, \\ (w_a - w_b) F(\{a\}) + w_b F(\{a, b\}) & \text{if } w_a \geq w_b. \end{cases} \tag{4}$$

This extension is piecewise linear for any set function F, and it is tight at the corners of the hypercube. Furthermore, f is convex if and only if the corresponding set function F is submodular (Lovász, 1982). The Lovász extension may be defined on $\mathbb{R}^n$ using the same notion.

2.3 Base polytope

Any piecewise linear convex function may be represented as the support function $f(w) = \max_{s \in K} \langle w, s \rangle$ of a certain polytope K (Rockafellar, 1997). For the Lovász extension of a submodular function, there is an explicit description of K, known as the *base polytope*, which we now describe.



(a) Geometrical representation from its supporting hyperplanes $\{s(A) = F(A)\}$.

(b) Each face (point or segment) of $B(F)$ is associated with an ordered partition.

(c) Outer approximation $\widehat{B}^{(\{1,2\},\{3\})}(F)$.

(d) Outer approximation $\widehat{B}^{(\{1\},\{2\},\{3\})}(F)$.

Figure 1: Overview of the base polytope and its outer approximations for $n=3$.

Definition 2.1 *Base polytope.* Let $s(B) = \sum_{i \in B} s_i$, where s_i is the ith element of $s \in \mathbb{R}^n$. The base polytope

$$B(F) := \left\{ s(B) \leq F(B) \cap s(\mathcal{V}) = F(\mathcal{V}) \,|\, \forall B \subset \mathcal{V} \right\} \tag{5}$$

is included in the affine hyperplane $s(\mathcal{V}) = F(\mathcal{V})$, and it is traditionally represented by the projection onto that hyperplane (see Figure 1a and b). A key result in submodular analysis (Fujishige, 2005) is that the Lovász extension is the support function of base polytope $B(F)$, i.e., for any $w \in \mathbb{R}^n$, we have

$$f(w) = \sup_{s \in B(F)} \langle w, s \rangle. \tag{6}$$

2.4 Tangent cone

However, the number of constraints for the base polytope, Eq. (5), grows exponentially with the cardinality of $\mathcal{V}$, i.e. $2^n - 1$. 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

$$\widehat{B}^A(F) = \left\{ s(B_i) \leq F(B_i) \cap s(\mathcal{V}) = F(\mathcal{V}), \forall i \in \{1, \ldots, l-1\} \right\}. \tag{7}$$

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 $\left(\{1,2\} \prec \{3\}\right)$ and $\left(\{1\} \prec \{2\} \prec \{3\}\right)$, 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 graph cut 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 graph cut 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 $\widehat{B}^A(F)$ as the feature map

$$\phi(A) := s^* = \operatorname*{arg\,min}_{s \in \widehat{B}^A(F)} \frac{1}{2} \|s\|^2 \in \mathbb{R}^n, \tag{8}$$

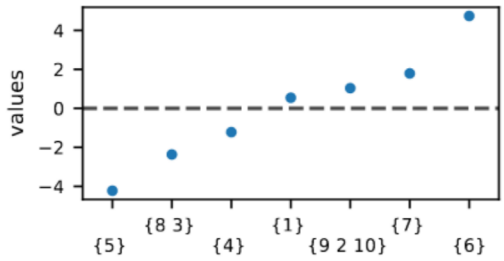


Figure 2: Optimal values v^*, Eq. (11), using the submodular cut function and the information graph of the sushi dataset for A.

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

$$\begin{aligned} &\min_{v \in \mathbb{R}^l} \sum_{i=1}^{l} v_i \big[F(B_i) - F(B_{i-1})\big] + \frac{1}{2} \sum_{i=1}^{l} |A_i| v_i^2 \\ &\text{s.t.} \quad v_1 \geqslant \cdots \geqslant v_l, \end{aligned} \tag{9}$$

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

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

We re-derive the details and show the pseudocode for the PAVA algorithm in Appendix A.

Note that v_i^* corresponds to the set A_i, while s_j^* corresponds to the object j and $s_j^* = -v_i^*$, 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 $i \in \{1, \dots, l-1\}$, 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

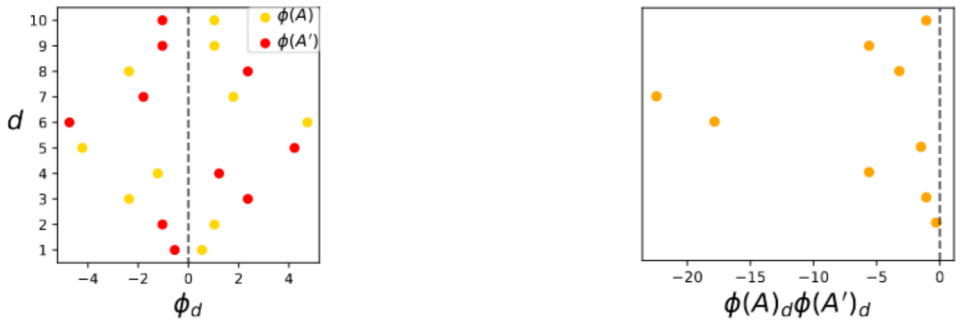
$$A = 5 \prec 8 \prec 3 \prec 4 \prec 1 \prec 2 \prec 9 \prec 10 \prec 7 \prec 6$$

to support the choice of the feature map. Using ϕ 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, $\phi(A)_5 = -4.22$, $\phi(A)_3 = \phi(A)_8 = -2.36$ and so on. Therefore, a natural and efficient way to exploit this pattern is to use the linear kernel on the resulting feature maps:

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

[2]Given a submodular function F and an ordered partition A, when the unique solution problem in Eq. (9) is such that $v_1 > \cdots > v_l$, we say that we A is a *basic ordered partition* for F. Given any ordered partition, isotonic regression allows to compute a coarser partition (obtained by partially merging some sets) which is basic.



(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 graph cut 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 graph cut 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

$$\begin{aligned} k_c(A, A') &= \frac{1}{|R(A))||R(A'))|} \sum_{a \in R(A)} \sum_{a' \in R(A')} k_s(a, a') \\ &= \frac{1}{|R(A))||R(A'))|} \sum_{a \in R(A)} \sum_{a' \in R(A')} \langle \phi(a), \phi(a') \rangle \\ &= \Big\langle \frac{1}{|R(A)|} \sum_{a \in R(A)} \phi(a), \frac{1}{|R(A')|} \sum_{a' \in R(A')} \phi(a') \Big\rangle \\ &= \langle \tilde{\phi}(A), \tilde{\phi}(A') \rangle, \end{aligned} \tag{14}$$

where $\tilde{\phi}(A)$ 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 $(l+1)^{n-l}$. In this case, we 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 The graph cut kernel

In summary, the algorithm to calculate the graph cut kernel happens in two sequential steps, (i) given a submodular function, we extract low-dimensional features from the data using PAVA, see Alg. 1, and (ii) 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 $O(mpl + m^2n)$. 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 $O(mspl + m^2n)$, 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 graph cut 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.

Algorithm 1 PAVA($\mathcal{A}$, F)

Input:
- Ordered partition $\mathcal{A}$ of length l.
- Submodular function F.

$v \leftarrow \{\}$. // Define an set to contain the optimal values
$k \leftarrow l$. // To retrieve sets B_{k-1} and B_k
while $|v| < |A|$ **do**
  $B_{k-1} \leftarrow \cup_{i=1}^{k-1} A_i$ // as in Eq. (7)
  $B_k \leftarrow \cup_{i=1}^{k} A_i$ // as in Eq. (7)
  $v' \leftarrow -\frac{F(B_{k-1})-F(B_k)}{|B_{k-1}|-|B_k|}$ // optimal solution Eq. (11)
  // Constraint violation
  if $|v| \,!= 0$ `and` $v' < v$.`first`() **then**
    `MERGE`($A[k-1]$,$A[k]$) // Merging items
    v.`removeFirst`() // Removing last value
  // No violation
  else
    v.`insert`(v') // Inserting value in the set
    $k \leftarrow k - 1$ // One less
  end
end while
Return v // Returning the set of optimal values associated to $\mathcal{A}$

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 which is a traditional testbed for kernel-based methods for rankings (Jiao & Vert, 2015; Lomeli et al., 2019). The code is available in anonymized form during the reviewing phase at `https://anonymous.4open.science/r/CutFunctionKernel-1745/README.md`.

4.1 Datasets

The downstream classification task is performed on a synthetic dataset that we propose and on a real-world dataset.

Synthetic dataset. The synthetic dataset resembles food preferences of eight dishes

$$\{\textit{Cake}, \textit{Biscuit}, \textit{Gelato}\}, \quad \{\textit{Pasta}, \textit{Pizza}\}, \quad \{\textit{Steak}, \textit{Burger}, \textit{Sausage}\}.$$

partitioned in three groups based on the numerical features, which respectively describe them: *Sweetness*, *Savouriness* and *Juiciness*. The exact values for the features are reported in Table 3.

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 $\{\textit{Sweet} \succ \textit{Savouriness} \succ \textit{Juicy}\}$ (*type one* users) and $\{\textit{Juicy} \prec \textit{Savouriness} \prec \textit{Sweet}\}$ (*type two* users), the classification task is to distinguish the types of users by their preferences.

	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 3: Numerical features for the objects of the synthetic dataset.

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 3 using the importance weights of the two types of users we obtained two sets of scores for the objects, which Table 4 reports. From the table, it is easy to see that the scores given to the objects greatly vary for the two

	Cake	Biscuit	Gelato	Steak	Burger	Sausage	Pasta	Pizza
Type one	0.93	0.72	1.06	0.21	0.42	0.36	0.58	0.6
Type two	0.38	0.08	0.79	0.93	1.05	1.18	0.85	0.79

Table 4: Scores given to the food objects by the two different types of users.

types of users and they have a strong connection with the underlying feature preferences.
In order to create a more challenging classification task, it is possible to inject Gaussian noise with σ standard deviation to the scores and return the vector of indices sorted according to the jittered scores.
Formally, the sample procedure is,

$$\begin{aligned}\text{Ranking}_{\texttt{User1}} &\sim \texttt{ArgSort}\big(\text{Score}_{\texttt{User1}} + \mathcal{N}(0, \sigma^2 I_8)\big) \in \mathbb{S}_8 \\ \text{Ranking}_{\texttt{User2}} &\sim \texttt{ArgSort}\big(\text{Score}_{\texttt{User2}} + \mathcal{N}(0, \sigma^2 I_8)\big) \in \mathbb{S}_8,\end{aligned}$$

where $\mathbb{S}_8$ is the permutation space yielded by the eight objects in the dataset, `ArgSort` returns the indices the list in input and $\text{Score}_{\texttt{User}}$ refers to the scores as in Table 4.
In Table 5 we report the underlying noise-free preferences of the two types of users which can be derived from Table 4. The preferences are expressed in increasing order, i.e. zero is associated with the least favourite object and 7 to the most preferred.

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

Real-world dataset. The state-of-the-art dataset for rankings data is the sushi dataset (Kamishima, 2003). Japanese users were 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).

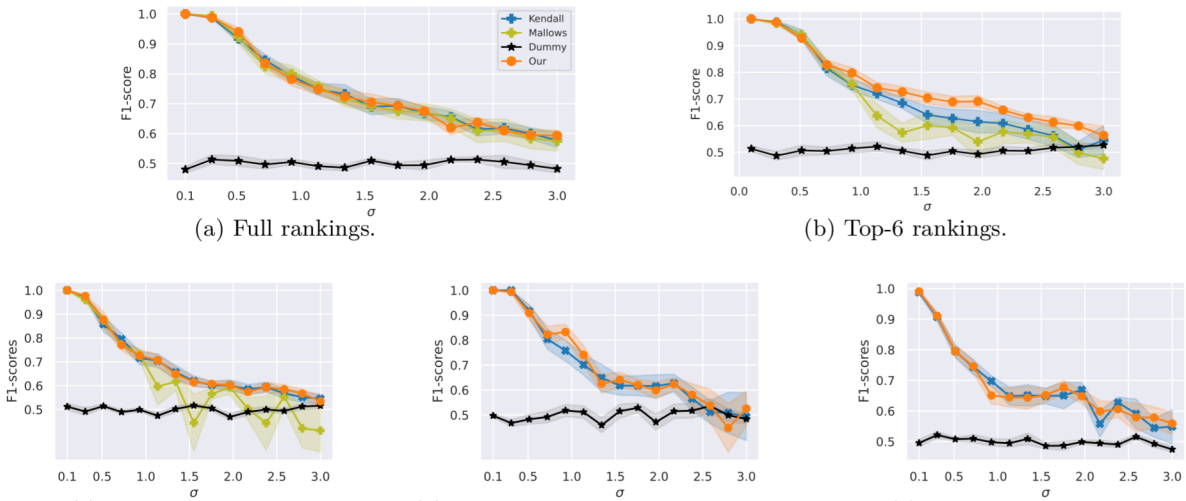
4.2 Classification

To assess the generalization capability of the graph cut kernel and to compare it with the state-of-the-art kernels, we fit 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. Note that for a perfectly balanced dataset, i.e. equal number of samples for each category, the F1-score is equal to the accuracy score. To train the GP classifier we use the python package GPflow (Matthews et al., 2017), which we extended with custom classes. The cut function is applied to a graph build from the information on the objects, i.e., the dishes in the synthetic dataset and the sushis in the real dataset. The edges of such graph are calculated using the squared exponential kernel as the similarity score function, i.e.,

$$e(v, v') = \exp\Big(-\frac{1}{2} \frac{\|v - v'\|_2^2}{\ell^2} \Big). \tag{15}$$

The lengthscale ℓ 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.



(a) Full rankings. (b) Top-6 rankings. (c) Top-3 rankings. (d) Exhaustive interleaving rankings. (e) Interleaving rankings.

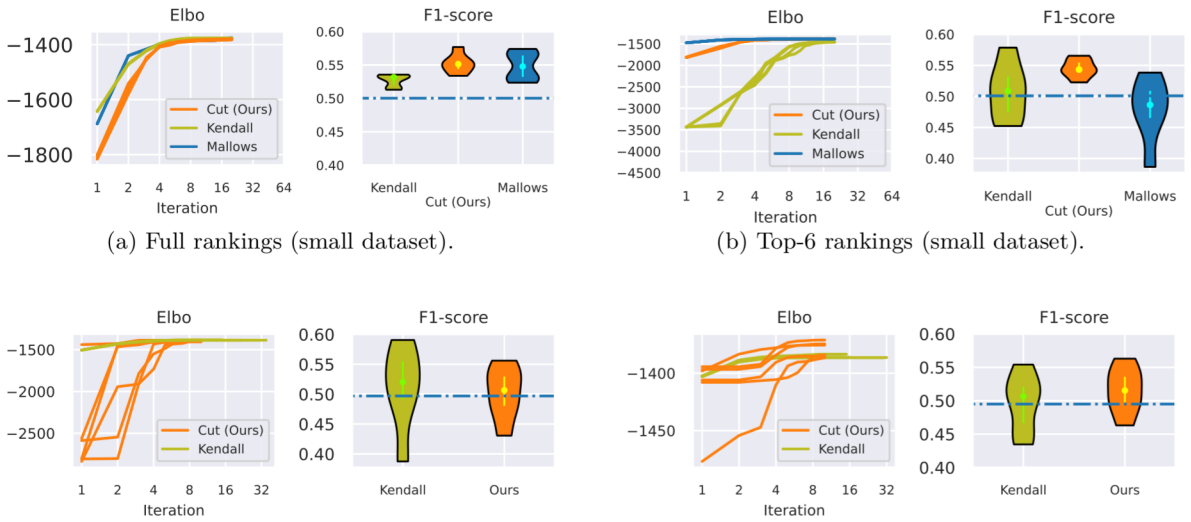
Figure 4: We perform a train/test procedure on the synthetic dataset using 250 samples for each increasing level of noise and for all the different kind of rankings. We report the F1-scores calculated on test data. generated from the synthetic dataset. We average the results over six seeds.

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 the different kernels on

increasing levels of noise. The graph cut 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 graph cut 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 graph cut 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 significantly worse performance in the top-3 one, Figure 4b. For both the exhaustive (Figure 4d) and non-exhaustive interleaving rankings (Figure 4e), the graph cut 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-world 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 0.6. Nonetheless, our graph cut kernel slightly outperforms the state-of-the-art kernels. More specifically, in the full rankings scenario, Figure 5a, the F1-scores achieved by the graph cut kernel is slightly better compared to the Mallows kernel and better than the Kendall kernel. For the top-6 rankings case, Figure 5b, the graph cut kernel is the only kernel to achieve better than random performance. The case of interleaving rankings present a challenge both for the Kendall and the graph cut kernel, Figure 5c and Figure 5d. We note that the performance of the graph cut kernel can still be considered superior to the Kendall kernel as it is less variable across different seed and slightly better in the large dataset, Figure 5d.

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



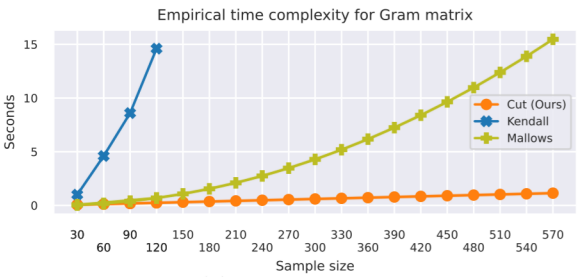
(a) Full rankings (small dataset).

(b) Top-6 rankings (small dataset).

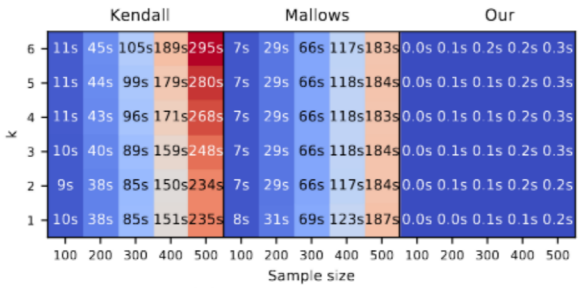
(c) Interleaving rankings (small dataset).

(d) Interleaving rankings (large dataset).

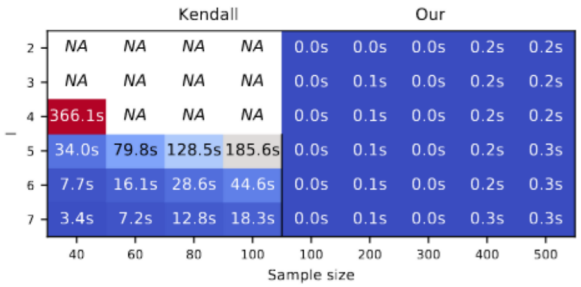
Figure 5: Train/test procedure for GP repeated over six seeds for the sushi dataset. The small sushi dataset is employed for full (a), top-6 (b) and exhaustive rankings (c), whereas the large sushi dataset is used for non-exhaustive rankings (d). For each kind of rankings, we report the evidence lower bound (ELBO) during training (*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.



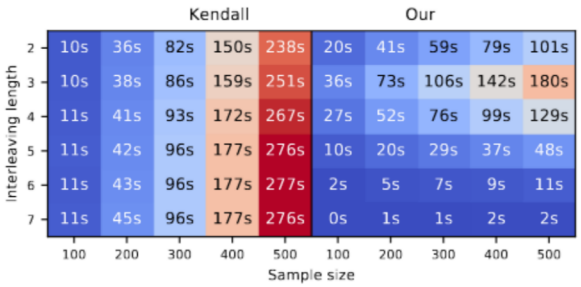
(a) Full rankings.



(b) Top-k rankings.



(c) Generic exhaustive rankings.



(d) Non-exhaustive rankings.

Figure 6: Comparison of empirical time complexity for calculating the Gram matrix using the submodular kernel and state-of-the-art kernels. Each figure reports the results for a different kind of rankings. Note that the symbol *NA* is reported when the empirical time complexity exceeds the maximum allowed time per simulation (seven minutes).

4.3 Empirical time complexity

In the following, we compare the empirical time complexity for all kernels and highlight the steep advantage of the graph cut kernel. For algorithms meant to be employed in real-world applications, it is essential to analyse 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[3] only using the CPU. On the contrary of state-of-the-art methods, for the graph cut kernel, we can easily use GPU hardware acceleration to calculate the Gram matrix, however, in order to present a fair comparison, we limit the graph cut kernel calculation on CPU only. The accessibility of hardware acceleration for the graph cut kernel is immediate since it is built on top of the linear kernel for which software as GPflow 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 $\pm$ 88.6ms, while the time required to compute the Gram matrix from the feature maps is 576ms $\pm$ 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 graph cut 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.

[3]We employ an Intel Core i7-7700HQ CPU @ 2.80GHz.

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 graph cut 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 graph cut 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[4]. In practice, the graph cut kernel scales almost linearly with the sample size.

5 Discussion

The graph cut kernel is a theoretically founded kernel that builds on top of the efficient machinery of graph cut 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.

Permutohedrons are discrete objects best suitable for encoding ranked data. However, they exponentially many vertices where each vertex corresponds to a permutation. This property makes it computationally complex us to compute to feature representation. In our approach, we use graph cut functions that shows a principled way to represent the permutohedron through its base polytope. These functions are designed using additional application specific information. This provides us a principled approach to traverse through the base polytope to provide an efficient feature representation. We have shown this empirically through several experiments.

Graph cuts are submodular functions and it is only this property that we have used to design the kernels. We believe general submodular functions specific to an application (e.g. entropy, mutual information etc.) may be used to design such kernels. This does not require any changes from our approach apart from designing submodular functions using application specific data.

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 graph cut kernel that attains good empirical performance while only incurring fairly small computational costs. Thanks to the graph cut 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 graph cut 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 graph cut kernel will greatly benefit both practitioners, as they now have access to a wider range of tools for real-world applications, and theoreticians, as we have consolidated the theoretical connection between the graph cut function optimization and ranked data. We have thus opened several exciting research directions. We leave as an open problem the extension of the graph cut kernel to general submodular functions, which will require a different way to extract information from the rankings. Another possible future research direction is to adapt the graph cut kernel from ranking to ratings.

[4]The time for calculating the feature maps is included in the presented empirical time complexity for the graph cut kernel.

References

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

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

Maria-Florina Balcan, Nikhil Bansal, Alina Beygelzimer, Don Coppersmith, John Langford, and Gregory B Sorkin. Robust reductions from ranking to classification. *Machine Learning*, 72(1-2):139–153, 2008.

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

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

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

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

Johannes Fürnkranz and Eyke Hüllermeier. Pairwise preference learning and ranking. In *Proceedings of the European Conference on Machine Learning*, pp. 145–156, 2003.

Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

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

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

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

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

Tao Hu, Lichao Huang, and Han Shen. Multi-object tracking via end-to-end tracklet searching and ranking. *arXiv preprint arXiv:2003.02795*, 2020.

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

Rishabh Iyer and Jeff Bilmes. 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.

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

Yunlong Jiao and Jean-Philippe Vert. The weighted Kendall and high-order kernels for permutations. In *Proceedings of he International Conference on Machine Learning*, pp. 2314–2322, 2018.

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

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

Maurice George Kendall. *Rank Correlation Methods*. Griffin, 1948.

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

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

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

Maria Lomeli, Mark Rowland, Arthur Gretton, and Zoubin Ghahramani. Antithetic and Monte Carlo kernel estimators for partial rankings. *Statistics and Computing*, 29(5):1127–1147, 2019.

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

Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, 2017.

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

Bernhard Schölkopf and Alexander J Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.