Shapley Values for Explanation in Two-sided Matching Applications

Suraj Shetiya
University of Texas at Arlington
suraj.shetiya@mavs.uta.edu

Ian P. Swift
University of Illinois Chicago
iswift2@uic.edu

Abolfazl Asudeh
University of Illinois Chicago
asudeh@uic.edu

Gautam Das
University of Texas at Arlington
gdas@cse.uta.edu

ABSTRACT

In this paper, we initiate research in explaining matchings. In particular, we consider the large-scale two-sided matching applications where preferences of the users are specified as (ranking) functions over a set of attributes and matching recommendations are derived as top-k. We consider multiple natural explanation questions, concerning the users of these systems. Observing the competitive nature of these environments, we propose multiple Shapley-based approaches for explanation. Besides exact algorithms, we propose a sampling-based approximation algorithm with provable guarantees to overcome the combinatorial complexity of the exact Shapley computation. Our extensive experiments on real-world and synthetic data sets validate the usefulness of our proposal and confirm the efficiency and accuracy of our algorithms.

1 INTRODUCTION

Beyond its traditional use [1–3], matching has been a core functionality of many of the modern two-sided online platforms [4–6], including dating applications such as `Tinder`, `OkCupid`, and `Bumble`[1], employment-oriented platforms such as `Linkedin`, `Indeed`, and `Zip-Recruiter`[2], and many more. The two-sided matching platforms provide matching recommendations between two types of stakeholders (users). To better explain the matchings, let us consider Example 1.1 as a running example across the paper.

Example 1.1. (Part 1) Consider a two-sided employment-matching application with two types of users, namely *job candidates* and *human resource (HR) users.* The application provides matching recommendations to both users, candidates and HRs. For example, a HR who looks at potential candidates for interview (either directly or indirectly) specifies a set of criteria and their preferences. Then the application returns a set of potential job candidates to the HR. It similarly finds matching job opportunities for the candidates.

Matching in two-sided platforms can be modeled as a bipartite graph[3] where users on one side are matched to the users on the other side. For instance, Figure 1 illustrates Example 1.1 as a bipartite graph, where job candidates and HRs are specified as red and blue nodes, respectively, while an edge $t_i \rightarrow t_j$ means that t_i has been recommended as a match for t_j. Application usually identifies the list of potential matches for each user (called *match list* in this paper) based on their *"individual preferences"*. While classic matching problems assume that each party *explicitly* specifies their preference as a ranking over the entire set on the other side, this assumption is not feasible for modern matching applications, simply due to their numerous number of users, the short attention span of users, and in some cases privacy considerations. As a result, the preferences are instead *implicitly* specified. That is, every user is associated with a set of *attributes*, and the preference of each user is defined as a function over the attributes of the other-side parties. Preference functions are either learned or specified by the users. Matching application uses the user's preference function to shortlist a limited list of candidates (the top-k).

Lack of adequate *explanations* in these systems is a major issue where the users who are impacted by the decision may be interested to know more insights about the matching and why they (or others) *do* or *do-not* appear in certain match lists. To further elaborate on this, let us continue with Example 1.1.

Example 1.1. *(Part 2) Looking back at Example 1.1-Part 1, suppose four attributes are considered for matching: Python, R, PHP, and Javascript. Each candidate has a skill level for each of these attributes, as does each HR, describing the nature of the job. Additionally, each Candidate and HR has an importance weight associated with each attribute, forming their preference as a linear function, while* $k = 2$. *Suppose candidate* t_3 *wants an explanation based on their matchings , which were either disappointing or pleasing. Currently the system provides no explanation for any of the four scenarios below, for which* t_3 *may be interested in transparency:*

1. t_3 *was disappointed to not make it to the top-2 of HR* t_{20}*. An explanation would provide value to* t_3*.*
2. t_3 *was happy they made it into the top-2 of HR* t_{19}*.* t_3 *is eager to know as to what sets them apart from the rest.*
3. *Top-2 of* t_3 *consisted of* $\{t_{19}, t_{20}\}$*. An explanation for why these HRs are good recommendations would be useful.*
4. *Candidate* t_3 *made it into the top-2 of HRs* t_{14}, t_{19}*.* t_3 *wants to know why they made it into these specific top-2s.*

The lack of answers for these questions prevents TRANSPARENCY *for Candidate* t_3*.*

In this work, we create a framework to provide explanations to various queries which are commonly encountered by the users of two-sided matching applications. To the best of our knowledge, this is *the first paper in explaining matchings.* In particular, we observe that top-k (ranking) problem is inherently competitive. As a result, the outcome (score) of a preference function is not enough to realize if a user appears in a match list or not. What

[1]tinder.com; okcupid.com; bumble.com
[2]linkedin.com; indeed.com; ziprecruiter.com
[3]As we shall explain in § 5, matching has many different formulations, properties, and applications. In this paper, our scope is limited only to bipartite many-many matching for two-sided online platforms.

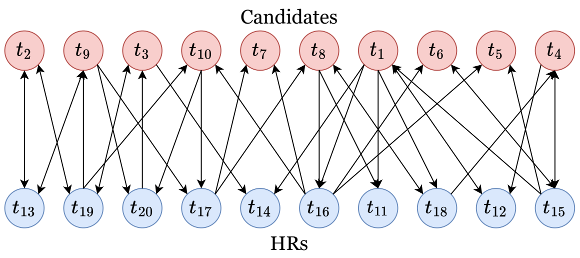


Figure 1: Sample bipartite graph for Example 1.1

	Python	R	PHP	JS
Candidate Attribute Values	2	2	1	1
HR Attribute Values	2	2	0	3
HR Ranking Weights	0.11	0.11	0.67	0.11
Shapley Values	0	-0.16	0.83	0.33

Table 1: The generated explanation for why Candidate t_3 is not in the Top-K of HR t_{20}

	Python	R	PHP	JS
Candidate Attribute Values	2	2	1	1
HR Attribute Values	3	2	0	2
HR Ranking Weights	0.11	0.44	0.33	0.11
Shapley Values	0.25	0.91	-0.08	-0.08

Table 2: The generated explanation for why Candidate t_3 is in the Top-K of HR t_{19}

	Python	R	PHP	JS
Candidate Attribute Values	2	2	1	1
Candidate Ranking Weights	0.84	0.02	0.06	0.08
Shapley Values	0.61	0.28	0.0	0.11

Table 3: The generated explanation for why Candidate t_3's Top-K looks the way it does.

	Python	R	PHP	JS
Candidate Attribute Values	2	2	1	1
Shapley Values	0.48	0.23	0.33	0.15

Table 4: The generated explanation for why Candidate t_3 appears in the Top-Ks of the specific HRs

.

matters in these settings is the relative position (rank) of a user with respect to other users who "compete" for the top-k positions. Such competitive environments are naturally explainable by *Shapley values* [7] – a game theory concept that identifies the contribution of each player (each attribute in our context) for deriving an outcome (e.g., a top-k match list). Shapley values have been proven repeatedly to solve explainability problems across different contexts [8–10].

Based on this observation, we consider Shapley values as the core of our system for our explanations. We consider a set of possible explanation queries, and provide Shapely-based approaches to answer them. Exact computation of Shapley values is a combinatorially hard problem, requiring algorithms that are exponential to the number of players. On the other hand, users might find accurate approximation of the values appropriate for explanation. Therefore, we propose a sampling-based approximation algorithm with provable guarantees on run-time and approximation error. Our system enables explanations as demonstrated in Example 1.1 (Part 3).

Example 1.1. *(Part 3) Using Shapley-based methods, we generate an explanation for each of the previous queries. These explanations take the form of a value for each of the skills and other features on which the matching is generated. A high value indicates that the feature was largely responsible for each of the four cases. The individual can then be provided with a general explanation as to what about them resulted in the various outcomes.*

1. *(From Table 1) t_3 is informed that they failed to be matched to HR t_{20} because they were not a good match with their PHP skills. However, based on their R skills alone, they would have been a good match for the job.*
2. *(From Table 2) t_3 can be told that the reason they were in the top-2 of HR t_{19} is because they were an excellent match on R skills. They can also be informed that their Python skills were less but still beneficial towards the matching as well.*
3. *(From Table 3) Top-2 of t_3 consisted of $\{t_{19}, t_{20}\}$. They are informed that this is largely because of the Python requirements of the HRs, and less so because of the R and JavaScript requirements.*
4. *(From Table 4) Candidate t_3 is provided the information that they made it into the Top-2 of HRs t_{14}, t_{19} because of their Python skills first, then their PHP skills, then their R skills, and finally their JavaScript skills, with each contributing slightly less than the previous.*

Summary of contributions. In summary, our contributions are as follows:

- In this paper, we initiate a study of a novel problem - that of providing explanations for matching and top-k recommendation systems. To the best of our knowledge, this paper is *the first to study explanation for matching.*
- We propose four explainability problems on the top-*k* matching model which help in providing transparency to the system and *white boxing* the blackbox function.
- Considering the competitive nature of our matching problem, we propose a Shapley-based approach to explain the queries and provide run time analysis for each of the problems. We show the need for alternate methods, as the run time is bound exponentially by the number of dimensions.
- We propose a sampling based approach to compute approximate Shapley values and prove guarantees on the trade-offs between number of samples and error rate. We adapt KernelSHAP as a practical heuristic to our problem.
- Extensive experimental analysis are provided for the various query settings and error guarantees, and our methods are evaluated in the real world via a user study.

2 PRELIMINARIES

Data model: We consider a dataset $\mathbb{D}$ with a Boolean attribute for matching (*blue* and *red*), *d* numeric or categorical attributes $\mathbb{A} = \{A_1 \cdots A_d\}$. The dataset $\mathbb{D}$ consists of *n* entities t_1 to t_n, with a sizeable number of *blue*s and *red*s. We use the notation $t_i[j]$ to refer to the value of the attribute A_j for the entity t_i. Similarly, we use $t_i[m]$ to refer to the type of t_i, i.e. the value of the Boolean matching attribute on t_i. The values in the dataset $\mathbb{D}$ (numeric or categorical represent the scores of each entity for various attributes, which are used in the matching process.

Note on distinct attributes for blues and reds: Our data model uses the same set of attributes for both the *blue* and *red* entities.

All the concepts and techniques defined later in the paper can be easily modified to work with a data-model where the blue entities have a different set of attributes (d_1) compared to the red (d_2 attributes). For the sake of simplicity, our data model is defined with symmetric attributes for both the *blue* and *red* entities.

Ranking functions: Each entity $t_i \in \mathbb{D}$ is associated with a ranking (aka preference or scoring) function that maps any given entity to a real valued score $f : \mathbb{R}^d \rightarrow \mathbb{R}^+$. The ranking function is used to express the preference of an entity during the matching process. As a hard criteria for matching, an entity with blue matching attribute only wants to match with an entity with red matching attribute and vice-versa. For instance, job recruiters (*blue*) and job seekers (*red*) are trying to match in a job matching scenario. The list of ranking functions F consists of n ranking functions corresponding to each of the entities. An entity t_i's ranking function is referred to as f_i throughout the paper. Some widely used types of ranking functions are linear, nearest-neighbor, and monotonic [11]. The techniques proposed in this paper are agnostic to the choice of ranking function. In this paper, the time taken to compute a score by the ranking function is referred to as C. An important requirement for the ranking functions we consider in this paper is the masking property. That is, given a ranking function, one can tune the function not to consider masked attributes when computing the scores. Masking is usually possible (including in linear, nearest-neighbor, and monotonic functions) by setting the values of the masked attributes as zero or null across all entities. That is, given an attribute A_j to mask, one can set $t_i[j] = 0, \forall t_i \in \mathbb{D}$. The set of attributes M which are set to 0s (or *nulls*) are known as masked attributes. The purpose of the masking function is to generate the outcome of a subset of non-masked attributes. Please note that in case of categorical attributes, the ranking function must be able to handle null values. In later sections, we explain the importance of the masking property for explanations.

Match list: As there are a large number of entities, any entity would like to see a small relevant set of entities as a potential match. Ranking functions are used to rank all the entities in the dataset belonging to the opposite matching attribute. In this work, top-k entries are shown for each entity using the rank of the entities as potential match. That is, given the ranking function f_i for an entity t_i, scores are assigned to all the entities $\{t_j \in \mathbb{D} \mid t_j[m] \neq t_i[m]\}$ using f_i. Those entities are then ranked and the top-k are chosen to be shown to t_i.

These top-k entries, known as a match list, are used to express the recommendations for matching. We denote a match list by l_i, such that each list consists of k entities. Given an entity t_i, let f_i^k represent the score for the k^{th} ranked entity using the scoring function f_i. Given an entity t_i,the match list can be mathematically expressed as,

$$l_i = \{t_j \mid f_i(t_j) \geq f_i^k \text{ and } t_j[m] \neq t_i[m]\} \quad s.t. \quad |l_i| = k$$

When a mask M is applied to obtain the top-k, the match list is represented as $l_i(M)$. Note that the value of k depends on the application and is not restricted to a fixed value for any individual. Without loss of generality, in this paper we assume a consistent k across all entities' ranking functions.

2.1 Problem definition

Our objective in this paper is to *increase responsibility in matching systems* by providing individuals with explanations about the matching and further information regarding why the matches occurred the way they did. The matching model for which explanations are being provided is called the "Top-k matching model", which is formally defined as:

Definition 2.1 (Top-k matching model). Given a dataset of entities $\mathbb{D}$, an integer k and ranking function for each of the entities, determine the match lists of each of the entities using the top-k from the ranked list.

When match list l_i is provided for a problem instance, it may not be clear how the various attributes contributed to the outcome.To solve this problem, an *explanation* is provided which identifies the role of different attributes in producing the outcome.

Definition 2.2 (Explanation). Given an output of a function $y = f(A_1, A_2, ...A_d)$, determine the impact on producing the overall value y of each attribute A_i when A_i is included as a parameter.

We now transition into explainability problems which help in making the system transparent.

2.1.1 Point Queries. The first two types of explanations are when an entity queries about their presence or absence from another entity's match list. This type of explanation is simply called a "point query".

In Example 1.1, Candidate t_3 finds that they are not present in t_{20}'s match list. In such a scenario, t_3 would want an explanation for: *why t_3 is not present in t_{20}'s match list?* Such a problem/scenario where Candidate t_3 would like an explanation for why they were not present in t_{20}'s list, can be formally defined as follows:

PQ-NotMatch : *Given a dataset $\mathbb{D}$ and entities t_i and t_j determine the contribution of attributes $A_1, \ldots, A_d$ in t_j* ***not appearing*** *in match list of t_i.*

Conversely, again using Example 1.1, Candidate t_3 may also be interested in what factors were responsible for them to be present in the match list of HR t_{19}. In this scenario, the Candidate, seeking to understand the scenario, might request for an explanation as to: *why t_3 is present in t_{19}'s match list?* Formally,

PQ-Match : *Given a dataset $\mathbb{D}$ and entities t_i and t_j determine the contribution of attributes $A_1, \ldots, A_d$ to t_j* ***appearing*** *in match list of t_i.*

The next two queries deal with set based properties and hence, we term these as *Set queries.*

2.1.2 Set queries. A different type of query that Candidate t_3 from Example 1.1 might ask pertains to the match list. The candidate t_3 might also be curious, based on their ranking function f_3, *why the match list l_3 was generated some way*, either because they are happy or disappointed with the list that was provided to them. Formally, the query, *why does an entity's match list look a certain way*, can be formulated as,

SQ-Single : *Given a dataset $\mathbb{D}$ and an entity t_i with a match list of l_i, determine the contribution of attributes $A_1, \ldots, A_d$ to t_i's match list looking like l_i.*

Finally, in addition to point queries and understanding their own ranking, Candidate t_3 from Example 1.1 may be concerned with the outcome of being ranked by others. Particularly, they may be interested in what attributes about them influence the set of match lists in which they appear.

Such an explanation can help Candidate t_3 understand the factors responsible for their current matches, for both cases where they are either pleased or displeased with the results. If Candidate t_3 is overall displeased with the HRs who match with them, knowing what attributes are responsible for this outcome can be informative. Equally, if they are consistently pleased with the HRs with whom they match, knowing the attributes that contribute to this outcome can provide insight into what went right. Formally, the problem to determine factors that influence an entity appearing in the match list of other entities is defined as:

SQ-Multiple : *Given a dataset $\mathbb{D}$ and an entity t_i from the match list of a set of T entities, determine the contribution of attributes $A_1, \ldots, A_d$ for t_i appearing in the match lists of T entities.*

3 SHAPLEY VALUE BASED SOLUTION

We start this section with discussing why Shapley values are suitable for explaining top-k and bipartite matching problems, followed by an overview of Shapley technique. Next, we show the transformation of our problem to Shapley, and discuss its scalability issue for higher dimensions. Then, we propose a sampling-based approximation technique with provable guarantees. We conclude the section by adapting KernelSHAP as a practical heuristic to solve our problem.

3.1 Why Shapley?

Scoring and ranking functions have been well-studied topics, while a major focus has been on explaining scoring functions. Even though there are similarities between scoring and ranking, their underlying requirements bring out drastic differences [12]. The score of an entity only depends on the (attributes) of the entity itself. On the other hand, the rank of an entity depends not only on its attribute values, but also on the attribute values of the other entities in the dataset. Consider an entity t in a dataset $\mathbb{D}$. $f(t)$ assigns a score to t. However, to find out the rank of t, one first needs to compute $f(t')$ for every entity in $\mathbb{D}$, and then find the position of t in the sorted list of entities based on f.

Because our problem is based on ranking, methods for scoring are no longer applicable for many queries. For example, in a (score-based) classification task, the attribute with the highest weight in the scoring function would be the most impactful. However, in a ranking problem, the score values are important only in comparison with the score of other entities. As a result, it is not enough if the score of an entity is high; what matters is that it is higher than other entities in the dataset. In this situation, an attribute A_i with a *low weight* in the scoring function can become important if dataset entities have a *high variance* on it. This is because the ranking is determined by *competition* between the entities, which is not captured simply by the score on a single entity. Let us illustrate this with Example 3.1.

Example 3.1. (Part 1) Consider a sample dataset $\mathbb{D}$ with 3 attributes $\mathbb{A} = \{A_1, A_2, A_3\}$ and 5 entities, shown in Table 5. The entities belong to the same matching group. Let the scoring function for an entity belonging to the opposite group be the linear function $f(t_i) = 5t_i[1] + 4t_i[2] + t_i[3]$. For this example, let the

	t_1	t_2	t_3	t_4	t_5	weights
A_1	10	1.2	9	9.5	10.1	5
A_2	1	1.1	1.2	1.5	1.8	4
A_3	8	15	32	1	24	1
$f(t_i)$	62	25.4	81.8	54.5	81.7	

Table 5: (Example 3.1) A sample dataset $\mathbb{D}$ with three attributes A_1, A_2 and A_3, and 5 entities.

value of k be equal to 2. The linear ranking function scores the entities t_1, t_2, t_3, t_4 and t_5 with scores 62 , 25.4, 81.8, 54.5, and 81.7, respectively. The entities ranked by their scores are t_3, t_5, t_1, t_4 and t_2. Hence, the match list looks like $\{t_3, t_5\}$ after the ranking function is applied. By looking at the entities, the entity whose ranking function is used for ranking might question *why is t_1 not in the top-k?*

An explanation model based on scoring functions would emphasize on weights of the given query entity to obtain the contribution of the different attributes. The weights for $A_1(5)$ is the highest, followed by $A_2(4)$ and $A_3(1)$. Looking at these weights, a scoring based explanation approach would conclude that either of A_1 or A_2 might be responsible for the query result. But upon masking attributes $\{A_1\}$, $\{A_2\}$ or $\{A_1, A_2\}$ and re-ranking one can observe that t_3 and t_5 are always scored higher than t_1. On the other hand upon masking attribute $\{A_3\}$, one can see that t_1 enters the match list. This illustrates the combinatorial feature importance in explaining ranking.

Note on normalization: A common technique used to handle disproportionate scores is normalization. For the example above, we illustrate that attributes with *high-weights* post normalization does not answer our query correctly. For our scenario, one could normalize each attribute A_i using $t[i]/max_i$, where max_i represents the maximum value for attribute A_i. Attributes A_1, A_2 and A_3 can be normalized by dividing each of the tuples values for the corresponding attributes by 10.1, 1.8 and 32. As we have scaled the attributes using normalization, the weights for the scoring functions also need to be adjusted by multiplying by 10.1, 1.8 and 32 respectively. Thus, the scaled weights from the normalization process are $< 50.5, 7.2, 24 >$. As the contribution of each attribute in the Shapley function have remained same, the normalization process did not change the Shapley scores. Yet, attribute A_1 attribute has a higher weight than A_3. The high weight on A_1 still produces an improper explanation for our scenario.

Since ranking is based on the competition between entities, it is natural to map our problem using *coalitional game theory.* In a coalitional game, different players of a coalition in a competitive game compete for utility. Accordingly, we utilize coalitional game theory to capture the importance of different attributes.

Shapley value [7] is a concept in coalitional game theory which allows one to compute the importance of each of the players in the game to the outcome. Each game contains a set of n players and a utility function $v : 2^n \rightarrow \mathbb{R}$, which determines the worth of the subset of players. Note that the utility of an empty set of players is 0, $v(\emptyset) = 0$ as the value represents the worth of an empty set ($\emptyset$) of players. The average contribution by each of the players to the outcome of the game can be defined for each player i as the Shapley value Sh_i. Shapley value for the game is given by,

$$Sh_i = \frac{1}{n!} \sum_{X \in Sym(N)} (v(Pre_i^X \cup i) - v(Pre_i^X)) \qquad (1)$$

where $Sym(N)$ represents the Symmetric group of set N, and Pre_i^X represents the players preceding i in permutation X. Equation 1 captures the marginal increase in contribution of a player i to its predecessors in permutation X.

An alternate form of Equation 1 can be used to compute the Shapley values. For a given subset S of players, there exist $|S|!(n-|S|-1)!$ permutations each of which have the same utility value whose re-computation is avoided when using the alternate form given below,

$$Sh_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n-|S|-1)!}{(n!)} (v(S \cup \{i\}) - v(S)) \tag{2}$$

Even though the formula is helpful in reducing the complexity from $n!$ to 2^n, the computation is still exponential. Accordingly, in practice, as the number of attributes grow this process tends to become infeasible.

3.2 Mapping Shapley value to matching

In Example 3.1, we argued that ideas such as using attribute-weights in the scoring function may not be effective for explaining top-k matchings. Alternatively, we propose to map attributes $\{A_1, A_2, \cdots, A_d\}$ as players of a coalitional game, compute the contribution of each attribute using the concept of Shapley values, and use these values to explain matchings. Indeed, an attribute's contribution is query dependent. That is, for different explainability queries, its contribution may vary from one query to the other. For instance, in Example 1.1, the contribution of the attribute (skill) *Java* for the query *"why is t_3 in the match list of t_{19}"* is different from *"why is t_3 not in the match list of t_{20}"*.

We design a utility function v based on the query, which takes in a set of attributes $S \subseteq \mathbb{A}$ and maps it to a utility score which is a real value $\mathbb{R}$ (i.e. $v : 2^d \rightarrow \mathbb{R}$). For a given subset S, the masked attributes are the attributes that are absent in S (i.e. $M = A \setminus S$). Due to this operation, our technique expects the ranking function to support *masking*. We provide design details of the utility function for different queries in § 3.3.

3.3 Shapley value in matching

One of our contributions is to transform each of our explanation problems to Shapley value computation problem and define a utility function over the set of attributes. This section provides the details for the Shapley value based approach applied to the four types of explainability queries defined in § 2.1.

3.3.1 PQ-NotMatch. Consider the scenario when a dataset of entities $\mathbb{D}$ and two entities t_i and t_j are provided for a query type of PQ-NotMatch. The attributes (players) responsible for t_j not appearing in t_i's match list would be valued high.

To reflect this we define a value function that returns a 1 (0 resp.) based on if an the entity t_j's absence (presence resp.) in t_i's match list l_i for a subset of attributes S. In this case, the set of attributes has contributed to a failed match and hence a value (reward) of 1. On the flip side, the utility function returns 0 if t_j is present in l_i for a given subset of attributes S. To explain further with Example 1.1 (1), the utility function would return 1 if t_3 was not present in l^*_{20} and 0 otherwise. Assuming l^*_i is the match list computed based on strictly the attributes in set S, the utility function v can thus be defined as,

$$v(S) = \begin{cases} 0, & \text{if } S = \emptyset \textbf{ or } t_j \in l^*_i \\ 1, & \text{if } t_j \notin l^*_i \end{cases} \tag{3}$$

For Example 1.1 question (1), and the query PQ-NotMatch, the results are shown in Table 1. The exact algorithm generated the Shapley values 0, -0.16, 0.83, and 0.33 for the attributes Python, R, PHP, and JS respectively. As discussed earlier, this shows that PHP is the largest contributor to why Candidate t_3 was not in HR t_{20}'s match list. Additionally, since Python has a negative value, the candidate's Python skill would cause the individual to appear in the match list.

3.3.2 PQ-Match. The next explainability query which can be calculated using Shapley value is the second query PQ-Match. A similar approach that we have seen so far can be extended to this scenario.

In the PQ-Match utility function, a value of 1 (reward) is returned if t_j is present in t_i's match list l_i for a given subset of attributes S. Alternatively, v returns 0 if t_j is absent from t_i's match list l_i for a subset of attributes S, as in this case S has not contributed to a successful match. Again, l^*_i is the match list computed based on strictly the attributes in set S. To illustrate this with Example 1.1 (2), the utility function v would return 1 if t_3 was present in l^*_{19} and 0 otherwise. Utility function v for PQ-Match can thus be defined as,

$$v(S) = \begin{cases} 0, & \text{if } S = \emptyset \textbf{ or } t_j \notin l^*_i \\ 1, & \text{if } t_j \in l^*_i \end{cases} \tag{4}$$

Continuing with Example 1.1 question (2), the results for PQ-Match can be seen in Table 2. The exact algorithm generated the Shapley values 0.25, 0.91, -0.08, and -0.08 for the attributes Python, R, PHP, and JS respectively. By far the largest contributor to Candidate t_3 being in HR t_{19}'s match list is the candidate's R skills. Additionally, both PHP and JS can be seen as having a negative value, negatively impacting Candidate t_3 being in the match list.

3.3.3 SQ-Single. Utility functions for queries based on match lists explain more complicated problems than the previous two queries. This is due to these queries use *differences in the match list*, instead of *presence or absence from the match list.*

Consider the scenario where a dataset of entities $\mathbb{D}$ and an entity t_i with a match list of l_i are provided for the query type of SQ-Single. The utility function must capture the similarity of the computed top-k match list for the set of attributes S, l^*_i, with match lists for the full set of attributes $\mathbb{A}$, l_i. As the explanation relies on the top-k items, l_i, and its similarity with l^*_i, we can use the Jaccard similarity between these two sets as the utility function. The Jaccard similarity is a value between 0 and 1. The value 0 indicates that the sets share no common elements and 1 indicates that the sets l_i and l^*_i are identical. To further explain with Example 1.1 (3), v is the Jaccard similarity of the match list of entity t_3 ($l_3\{t_{19}, t_{20}\}$) with l^*_3. The utility function v can be expressed as,

$$v(S) = \begin{cases} 0, & \text{if } S = \emptyset \\ \frac{|l_i \cap l^*_i|}{|l_i \cup l^*_i|}, & otherwise \end{cases} \tag{5}$$

Again, with Example 1.1 question (3), the results for SQ-Single are shown in Table 3. The exact algorithm generated the Shapley values 0.61, 0.28, 0.0, and 0.11 for the attributes Python, R, PHP, and JS respectively. The largest contributor to Candidate t_3's *match list being the way it was*, is the Python requirement of the HRs, however to a lesser degree the R requirements contributed, and even lesser, the JS requirements contributed.

It is worth noting that our Shapley function does not distinguish the order/rank of the entities within the (top-k) match-list, i.e., it treats them as a set. Therefore, it uses Jaccard similarity, the metric for computing set similarities. But if the order of the items within the top-k is important, value function needs to account for any change within the ranking of the match-list.

3.3.4 SQ-Multiple. A similar approach of using Jaccard similarity can be used to find the contribution of each attribute for the SQ-Multiple problem. Consider the scenario when a dataset $\mathbb{D}$ and an entity t_i is provided. The entity t_i is present in the match list of $T \subseteq O$ entities. For a subset $S \subseteq \mathbb{A}$ of attributes, let T^* be the set of entities in whose match list t_i is present for the attributes S. The utility function represents the similarity between the set T^* and T.

A similarity of 0 indicates there are no shared elements between the two match lists, where a similarity of 1 indicates that the sets are identical. A smaller value indicates a smaller intersection or a higher union, and a larger value indicates a larger union or a smaller intersection. From the running Example 1.1 (4), the set of entities whose match list consist of t_3 is $T = \{t_{14}, t_{19}\}$. Hence, the function v for this case would compute the Jaccard similarity between $\{t_{14}, t_{19}\}$ and T^* for a subset of attributes $S \subseteq \mathbb{A}$. The value function v can be expressed as,

$$v(S) = \begin{cases} 0, & \text{if } S = \emptyset \\ 1, & \text{if } S \neq \emptyset \ \& \ |T \cup T^*| = 0 \\ \frac{|T \cap T^*|}{|T \cup T^*|}, & otherwise \end{cases} \tag{6}$$

Finally, for Example 1.1 question (4), the results for SQ-Multiple are shown in Table 4. The exact algorithm generated the Shapley values $0.48\bar{3}$, $0.23\bar{3}$, $0.33\bar{3}$, and 0.15 for the attributes Python, R, PHP, and JS respectively. All of the candidates skills contributed somewhat, but Python, PHP, R, and JS contributed in descending order of importance.

Having mapped Shapley values to the top-k matchings, we can now explain matchings using the attribute contributions, as shown in Example 3.1 (Part 2).

Example 3.1. *(Part 2) Looking at Table 5, the Shapley values of* $\{A_1, A_2, A_3\}$ *for the query* "why is t_1 not in the top-k?" *are computes as* $\{-0.1\bar{6}, 0.3\bar{3}, 0.8\bar{3}\}$, *using Equation 5. The high Shapley value of attribute* A_3 *indicates that* A_3 *can explain the query the most. The impact of attribute* A_3 *is partially seen empirically when we remove the attribute. Removal of* A_3*'s impact on the linear ranking function is reflected in the new scores the entities get,* t_1, t_2, t_3, t_4 *and* t_5 *get a score of* 54 , 10.4, 49.8, 53.5, 57.7 *respectively. These scores show that the entity* t_1 *is present in the match list when* $k = 2$ *when* A_3 *is removed, thus confirming its relative contribution.*

3.3.5 Time complexity analysis. Theorem 3.2 shows the time taken to compute the exact Shapley value is *exponential* to the number of attributes for all the four queries, making Exact Shapley value computation impractical when d is not small.

Theorem 3.2. *Given a dataset* $\mathbb{D}$ *with ranking functions* F *with the other parameters for PQ-NotMatch, PQ-Match, SQ-Single and SQ-Multiple, computing the exact Shapley value takes exponential time to number of the players (attributes).*

Proof. Computation of exact Shapley values using Equation 2 relies on computing the utility function efficiently over all subsets of $\mathbb{A}$. We analyse the running time of each of the value functions of the 4 problems and prove that the exponential nature arises solely from Shapley value computation from Equation 2. As noted in § 2, we denote the amount of time taken by the ranking function as C.

PQ-NotMatch: Consider a dataset $\mathbb{D}$ and entities (t_i and t_j). The value function used for PQ-NotMatch is given in Equation 3. Given a subset of attributes $S \subseteq \mathbb{A}$, the time taken to recompute the ranking function for an entity is C. As there are n entities, the function v takes nC time to obtain the scores for all entities. Obtaining the top-k (match list l_i^*) can be efficiently performed using the selection algorithm which takes a total of $O(n)$ time. Checking if the entity t_j is present in the match list takes k time. Hence, total time taken by function v is $O(nC)$ for a given subset S.

PQ-Match: Computation of the value function is similar to the computation of PQ-NotMatchand hence consumes $O(nC)$ time.

SQ-Single: Consider a dataset $\mathbb{D}$ and entity t_i with match list l_i when using all attributes $\mathbb{A}$. The Shapley value function for SQ-Single is given in Equation 5. The set based value function relies on *Jaccard similarity* between sets l_i and l_i^* to obtain a value. Hence, the first step is similar to PQ-NotMatch and PQ-Match problems, i.e. computation of l_i^*.

The Jaccard similarity computation can be efficiently performed by sorting the match lists l_i and l_i^*, followed by performing simultaneous linear scans on l_i and l_i^* to obtain both the intersection and union. This step consumes a total of $O(k \log(k))$ time. Hence, the total time consumed is $O(nC + k \log(k))$

SQ-Multiple: Consider a dataset $\mathbb{D}$ and entity t_i which is present in the match list of entities T when using all attributes $\mathbb{A}$. The Shapley value function for SQ-Single is given in Equation 6. Given a subset of attributes S, T^* can be obtained by first computing match list l_j^* for all n entities and checking which ones contain t_i.

The computation of l_j^* and checking if t_i is present in l_j^* ($t_i \in l_j^*$) for a single entity consumes $O(nC)$ as seen above. As there are n entities, obtaining T^* consumes a total of $O(n^2C)$ time. Additionally, to compute the *Jaccard* similarity, (i) need to sort both T and T^*, (ii) perform simultaneous scans on T and T^* to obtain both intersection and union, and (iii) obtain the ratio. Steps (i) and (ii) consume $O(n \log(n))$ time and (iii) consumes $O(n \log(n))$ time. Hence, the overall time consumed is $O(n^2C)$.

The Shapley value computation for each of the four cases relies on generating all subsets of the set of attributes $\mathbb{A}$. As there are d attributes, generating the sets consumes a total of $d\ 2^d$ time. Hence, the Shapley computation for all the four queries is exponential in d, the number of attributes $\mathbb{A}$. □

Note on limitations of Shapley based method: Kumar et al. [13] have shown certain limitations of Shapley-based methods while explaining machine learning models. During masking, these methods are shown to sample out-of-distribution data points which can affect the Shapley explanation model and create undesirable output. Matching-based applications are not subject to similar problems like machine learning models, as the whole process of top-k and bipartite matching is based on competition. These processes are less subject to changes in scores and instead rely on ranking between items. Nevertheless, we would like to note our dependence on the scoring functions to handle the NULL values generated during masking. Additionally, we would like to note that explanations generated from Shapley-based methods are not actionable and must be used as a means to *whitebox* the blackbox function.

Designing Shapley value function for other queries: Intuitively, the design of the value function needs to capture the nature of the query. We illustrate this with modifications to existing value functions when the queries are modified.

Let us consider the scenario of simultaneous matching. Consider the explainability query, *find the attributes responsible for t_i being present in t_j's match list and t_j being present in t_i's match list simultaneously.* While this query is similar to PQ-Match, value function needs to reward only when both the PQ-Match events occur simulataneously i.e. $t_i \in l_j^*$ **and** $t_j \in l_i^*$. Based on this, the value function can be defined as follows.

$$v(S) = \begin{cases} 0, & \text{if } S = \emptyset \textbf{ or } (t_i \notin l_j^* \textbf{ or } t_j \notin l_i^*) \\ 1, & \text{otherwise} \end{cases}$$

Let us now consider a scenario when the order within the top-k match list is important to the query. Consider the scenario when the entity t_i wants to be *near* the top of t_j's match list. Let us assume that the ranking process is modified to provide a ordered list of k entities instead of a set. With a little abuse of notation, let l_j^* be the ordered match list of t_j and p_i^* be the location of entity t_i in l_j^*. If t_i was located at position 1 in l_j^*, the value function needs to reward with a score of 1, otherwise the location p_i^* would define the reward. Based on the position p_i^* the value function is,

$$v(S) = \begin{cases} 0, & \text{if } S = \emptyset \textbf{ or } t_i \notin l_j^* \\ 1 - \frac{p_i^* - 1}{k}, & \text{otherwise} \end{cases}$$

3.4 Approximate sampling based approach

The prohibitive nature of the exact Shapley value algorithm has spurred research into approximate methods. Sampling based methods have been proposed to obtain approximate Shapley values. In this paper, we use the sampling based on permutations of attributes. Based on [14], the mean of the marginal contributions for each attribute A_i over the entire symmetric group $Sym(A)$ is equivalent to the corresponding Shapley value Sh_i. To approximate the value Sh_i, instead of calculating all members of the symmetric group, q members of the $Sym(A)$ are sampled. The estimated Shapley value for attribute A_i resulting from the samples is referred as $\hat{Sh_i}$ throughout the paper. The expected value of a random sample from the uniform distribution is equivalent to the mean of that distribution such that $Ex[\hat{Sh_i}] = Sh_i$.

A randomized approximation algorithm based on the sampling of permutations for computing the explanation queries is defined as follows. Initially, random sampling is performed to select q permutations from $Sym(A)$. With $\hat{Sh_i}$ set to 0, $\hat{Sh_i}$ is increased by $\frac{1}{q} \times (v(Pre_i^X \cup A_i) - v(Pre_i^X))$ for each sampled permutation X. The value $\hat{Sh_i}$, is the approximate Shapley value for attribute A_i.

Sample size, q: The randomized approximate algorithm can be demonstrated to show that by varying q and specifying an approximation bound α, an error rate of ϵ can be satisfied such that $Pr(|\hat{Sh_i} - Sh_i| < \alpha) > 1 - \epsilon$. We prove in Theorem 3.3 that a for a given value of approximation bound α and error rate ϵ, there exists a fixed samples size q which satisfies $Pr(|\hat{Sh_i} - Sh_i| < \alpha) > 1 - \epsilon$ for our queries.

Theorem 3.3. *Given an approximation bound α and error rate ϵ, sampling $q \geq \frac{2\log(2/\epsilon)}{\alpha^2}$ random permutation members of the symmetric group, ensures that the inequality $Pr(|\hat{Sh_i} - Sh_i| < \alpha) > 1 - \epsilon$ is satisfied for all four problems.*

Proof. We prove this theorem using Hoeffding's inequality [15]. First, we prove for the four queries that each random variable Z_i varies within the range of $Z_i \in [-1, 1]$. The random variable Z_i in each of the four problems refers to the Shapley value function for the sampled permutation. We obtain the inequality by applying Hoeffding's inequality.

For the queries PQ-NotMatchand PQ-Match, the random variable is the difference between the Shapley value function when attribute A_i is added to permutation X i.e. $Z_i = (v(Pre_i^X \cup A_i) - v(Pre_i^X))$. Hence, the random variable Z_i can take values -1, 0 or 1 based on how the Shapley value function changed.

For the set-similarity based problems SQ-Singleand SQ-Multiple Z_i is the difference between the Jaccard similarity for the random permutation $v(Pre_i^X \cup A_i)$ and $v(Pre_i^X)$. Hence, Z_i for these two problems is a continuous variable between -1 and 1. As we have seen, in all four problems the random variable Z_i varies between -1 and 1. Applying Hoeffding's inequality with $b = 1$ and $a = -1$ we get, $\mathbf{P}\left(|\hat{Sh_i} - Sh_i| \geq \alpha\right) \leq 2e^{\frac{-q\alpha^2}{2}} \leq \delta$. Restructuring the above equation and representing q in terms of α and ϵ we get,

$$q \geq \frac{2\log(2/\epsilon)}{\alpha^2}$$

Hence, proved. □

3.5 KernelSHAP

Another technique to approximate Shapley values is SHAP [10]. SHAP proposes a linear model to explain black box machine learning model-prediction for a given input data point. KernelSHAP is a generalised SHAP approximation technique proposed by Lundeberg et. al. [10]. The technique is built on top of LIME [16] to approximate Shapley values. KernelSHAP provides the parameters to be set in the optimisation function of LIME such that the linear model finds the Shapley values. To the best of our knowledge, KernelSHAP does not inherently provide any guarantees on sample size unlike the sampling approach, but empirically performs better.

4 EXPERIMENTS

We conduct extensive experiments on real-world and synthetic data to validate our proposal and to evaluate the performance of our algorithms. In the following, after discussing the experiment set up, we provide proof of concept experiments that focus on validating our results. Finally, the empirical evaluation of the different techniques is presented.

4.1 Experiments setup

4.1.1 Datasets. Matching datasets involving real-world entities are often not publicly available due to privacy concerns. Hence, in our experiments we considered two publicly-available real-world datasets, along with 12 configurations of synthetic datasets.

- Job candidates dataset (*real world dataset*)[4]: 390 candidates were parsed from 22 columns, including 18 numerical columns, 3 categorical columns, and 1 set based column. Values for numerical columns were normalized to be a value between 0 and 1. To generate HR entities, uniform random values were selected for each column from the set of possible values for that column in the candidate dataset.
- Graduate admissions dataset[17](*real world dataset*): The dataset consists of application details of 500 students to the graduate

[4]https://www.kaggle.com/datasets/saikrishna20/candidates-list

program. There are a total of 6 numerical features and a categorical feature. Each data point in the dataset also a dependant variable, *chance of admit*, which is a score between 0 and 1.

- Synthetic dataset: There are many parameters that can be varied when generating the datasets. We have used 3 main factors, probability distribution, correlation between the attributes, and the number of attributes. Since our experimental study aims to assess our method across different settings, we generate multiple datasets for each setting and aggregate results for each setting. For the probability distribution, we consider *Uniform* and *Zipfian* distributions. Linear and non-linear ranking functions; correlated, anti-correlated, and independent attributes were used to ensure variety of data in our experiments. For each of these 12 settings, 10 datasets were generated bringing the total to 120 datasets. Each dataset consists of 1000 items each with 9 dimensions.

4.1.2 Hardware and Platform. All our experiments were performed on a work station with a Core i9 Intel X-series 3.5 GHz machine running Linux Ubuntu with 128 GB of DDR4 RAM. The algorithms were implemented in Python 3[5].

4.1.3 Ranking function. As the ranking / preference functions were not available for the real world datasets, we generated n linear ranking functions. The ranking functions are based on proximity to the candidate's skills/ HR's requirements. For synthetic dataset, we have used both linear and non-linear ranking functions. For non-linear ranking functions, weighted squares of the attributes have been used to rank the entities. In the user study, the ranking function used for contrasting LIME and Shapley values explanations is learned using machine learning models based on the other features using *AutoML* which is then used to predict the *chance of admit*.

4.1.4 Algorithms Evaluated. We have implemented the brute force Shapley value algorithm and the approximate Shapley value algorithms. We use the KernelSHAP library from github by Lundberg [6].

As baselines to compare against, we use the weight based algorithm described in § 3.1 and an attribute based baseline. The attribute based baseline ranks each of the attributes individually for a given query. More specifically, for a query we mask the set of attributes such that only one attribute is unmasked. We measure the effect of the individual attributes and then rank these attributes. For the PQ-NotMatch (PQ-Match resp.) query, we use the highest (least resp.) rank achieved by the entity when only using a single attribute as a measure to compare all attributes and rank them. As SQ-Single and SQ-Multiple queries are set based queries, we use the Jaccard similarity measure instead to rank the attributes. The comparison for these baselines is provided in § 4.2.2.

4.1.5 Dataset details for Example 1.1. In this subsection, we provide details for the dataset in Example 1.1. The Example 1.1 dataset consists of 20 entities with 4 attributes, *Python, R, PHP* and *JS*. These 20 entities are a part of the Candidates real world dataset discussed in § 4.1 of the paper.

The Candidates entities are $t_1 \cdots t_{10}$ and the HRs are $t_{11} \cdots t_{20}$. As any HR entity would like to find candidates whose skills are similar to that of the *job requirement*, a weighted k-nearest neighbor function with ℓ_1 distance is used. The distance function can be mathematically written as, $f_i(t) = \sum_{j=1}^{4} w_j \, |t_i[j] - t[j]|$ The equation above obtains the *distance* between the requirement and hence smaller distance is preferred.

[5]Code can be accessed at https://github.com/UIC-InDeXLab/ExplainMatchTopK
[6]https://github.com/slundberg/shap

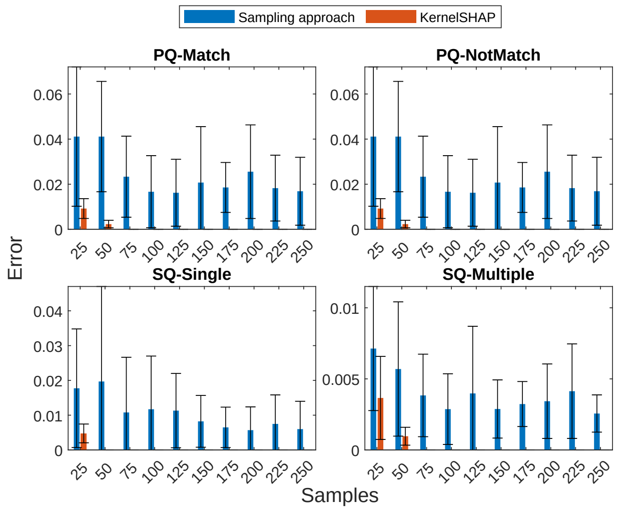


Figure 2: Error variations in sampling-based approach and KernelSHAP

Note that we have normalized each attribute between the values $[0, 1]$. The normalization for an attribute A_j is performed for any entity t_i using, $t_i[j] = \frac{t_i[j] - min(A_j)}{max(A_j)}$ where *min* and *max* find the minimum and maximum values for attribute A_j in the dataset $\mathbb{D}$.

4.2 Proof of Concept

As our first set of experiments, we provide results to validate our proposal for explaining match lists using Shapley values. In particular, we first present a case study, discussing the explanations for specific cases in detail. Then, we provide an experiment to demonstrate the effectiveness of Shapley values for explanation.

4.2.1 Case Study. We begin our proof of concept experiments by a case study to illustrate the usefulness of our explanations. Aligned with our running example (Example 1.1), we select a user from our experiments on job-candidates dataset, and discuss the generated explanation in detail. Approximate Shapley values produced using the sampling algorithm for PQ-NotMatch are shown in Table 6. Among their programming skills, the largest contributors to this result were `R, React.js` and `CSS` with Shapley values of 0.09, 0.09, and 0.085. In each of these cases, the candidate performed poorly on these skills while HR ranked these skills fairly highly relative to other programming skills. The lowest Shapley among programming skills value was `Python`, where the individual had the maximum score, but the weight was also the lowest relative to all skills. Overall, the highest Shapley value was for the Candidate's `Master of Science` degree, indicating that this degree (as opposed to another one) was the main reason the candidate was not placed on the match list. Finally, the overall lowest weights were their performances in grade 12, 10, and post grad, with scores of -0.05, -0.05 and -0.02. Negative Shapley values indicate that these skills worked against the candidate not being in the match list, and can be seen as skills that if evaluated solely on, the individual would appear in the match list. An explanation may appear as:

`"You were not matched with this HR largely due to your degree in Master of Science. Among your programming skills, the`

	Candiate Values	HR Function	Shapley Values
Python	1.0	0.002	0.0
R	0.0	0.005	0.09
Deep Learning	0.333	0.005	0.025
PHP	0.667	0.007	0.05
MySQL	0.667	0.007	0.075
HTML	0.667	0.007	0.035
CSS	0.0	0.005	0.085
JavaScript	0.667	0.005	0.065
AJAX	0.0	0.005	0.06
Bootstrap	0.0	0.006	0.07
MongoDB	0.0	0.005	0.045
Node.js	0.0	0.003	0.045
Reactjs	0.0	0.005	0.09
Performance_PG	0.791	0.06	-0.02
Performance_UG	0.7	0.06	0.015
Performance_12	1.0	0.06	-0.05
Performance_10	1.0	0.120	-0.05
Other Skills	['Algorithms', 'Data Structures', ...]	0.0769	0.065
Degree	Master of Science	0.0769	0.21
Stream	Computer Science	0.0769	0.05
Grad Year	2018	0.307	0.04
Current City	Bangalore	0.0769	0.005

Table 6: Candidate values, HR rankings, and PQ-NotMatch Shapley values.

```
company would like to see more skill in R, CSS, and React.js
specifically. The company was overall pleased with your
Grade 10, Grade 12, and Post-grad Performance, but
considering all factors, they did not want to match with you
at this time."
```

Please note that the Shapley value explanation does not provide a direction for improvement. An intuitive way of thinking about this is that Shapley values does not perturb the numeric or categorical values to measure its impact, instead it looks at the impact that an attribute has as a whole. Hence, our model is not aware about which direction is preferred or how an attribute needs to be perturbed.

4.2.2 *Effectiveness of Shapley values for explainability.*

For our first experiment, we empirically measure the effectiveness of the approximate algorithm in capturing the Shapley values. Given an explainability query, the brute force algorithm produces the exact Shapley values. Since our goal is to capture the exact Shapley values as accurately as possible, we compare these values with the results of various methods. To do this, we considered the *top ranked* attribute for each algorithm.

In this experiment, we measure the effectiveness of three methods in explaining the query. These methods are the Shapley values by sampling, the attribute with the highest weight (Section 3.1), and the attribute evaluated on the query function independently.

The datasets for this experiment consists of twelve settings on synthetic data. The settings are each combination of distribution, feature correlation types and scoring function type, i.e. {*uniform, Zipfian distribution*} × { *independent, correlated or anti-correlated features* } × { *linear, non-linear scoring functions*}. For each of these settings, ten trials were run with $k = 5$ and the results were recorded. For each experiment, highest rated attribute was removed and the output compared for its impact on the query. For the sampling based approximate Shapley value sample size (q) of 900 was used. KernelSHAP obtains the same result as sampling based technique with lesser number of samples.

The results are tabulated in Table 7. The results show that approximate method produced the same output as brute-force in 119 out of the 120 trials. The other methods performed consistently at best equal but generally worse across all queries. Measuring by weight performed almost always better than attribute based baseline. However, measuring by weight still often failed. It had a particularly low accuracy for PQ-NotMatch. Additionally, since weight based approach is not possible for SQ-Multiple, it was considered to be an insufficient method for computing the highest Shapley value.

4.2.3 *User study.*

In this experiment, we conduct a user study to validate our methods. The participants for this user study included working professionals - data analysts, software engineers, and graduate students. A total of 35 people took the user study. The first step of the user study was a quality control step to understand the participants knowledge and confidence in the answers. Among the 35 participants 28 participants showed sufficient knowledge in the control step. Hence, the answers of the remaining 7 participants were disregarded from the user study.

The goal of the user study is to assess the participant's preference between LIME and Shapley. The user study consisted of two scenarios and a question related to each scenario.

The first scenario was based on Example 3.1. Participants were provided with the scoring function $5A_1 + 4A_2 + A_3$ and told that t_5 was not selected during the selection process. Participants were provided the explanations from LIME and Shapley value and were asked to select among the two. For the explanation for LIME, we perturb the scores of tuple t_5 and create a new Boolean variable which measures if t_5 is present in the top-k or not ($k = 1$). Based on the new dataset, LIME produces an explanation by weighing each of the attributes. In order to explain the meaning of LIME to the user we provide the following statement. "*Increasing/Decreasing [FEATURE] by [X] unit contributes [Y] units to* t_5 *not being in the top-k, when all other feature values remain fixed*". The corresponding explanation for Shapley values is, "*The score of [FEATURE] contributed [X] to* t_5 *not being in the top-k compared to the average prediction for the dataset.*" Among the 28 participants 18 chose Shapley values and the remaining 10 chose LIME.

For the second scenario, we use the graduate admissions dataset. The scoring function for the graduate admissions dataset was learned using *auto-sklearn* which generated a ensemble model. We chose a candidate who did not qualify among the top-k but was very close to the k^{th} candidate. Similar to the previous scenario, particpants were given a choice between two explanations - LIME and Shapley value. Among the 28 participants, 16 voted in favor of Shapley and 12 in favor of LIME.

While the votes for the second scenario is close between LIME and Shapley to conclude a clear preference among the participants, for the first scenario the participants preferred the explanation generated by Shapley to be more favorable than LIME.

4.3 Performance Evaluation

Next, we evaluate the approximation error and runtime of sampling-based and KernelSHAP v.s. exact Shapley values.

4.3.1 *Impact of sampling size on error.*

Approximate sampling-based approach and KernelSHAP provide us with a trade-off between error and time-consumed. As the number of samples increases, the error in Shapley-value decreases, but time increases. We measure the impact of sample size on error and time consumed.

In this experiment, the number of samples that we use for the approximate Shapley value algorithm and KernelSHAP are varied and the runtime and error are measured. Match list size, k,

Distribution	Correlation	Function	APX - Q1	APX - Q2	APX - Q3	APX - Q4	WT - Q1	WT - Q2	WT - Q3	SCR - Q1	SCR - Q2	SCR - Q3	SCR - Q4
Uniform	Correlated	Linear	1.0	1.0	1.0	1.0	0.9	0.7	1.0	0.2	0.6	0.1	0.1
Uniform	Correlated	Non-Linear	1.0	1.0	1.0	1.0	1.0	0.5	0.9	0.5	.5	0.3	0.1
Uniform	Anti-Correlated	Linear	1.0	1.0	1.0	1.0	.8	0.7	1.0	0.5	0.7	0.1	0.2
Uniform	Anti-Correlated	Non-Linear	1.0	1.0	1.0	1.0	1.0	0.4	1.0	0.1	0.4	0.3	0.1
Uniform	Independent	Linear	1.0	1.0	1.0	1.0	1.0	0.6	1.0	0.4	0.7	0.3	0.4
Uniform	Independent	Non-Linear	1.0	1.0	1.0	1.0	1.0	0.5	1.0	0.2	0.5	0.5	0.2
Zipfian	Correlated	Linear	1.0	1.0	1.0	1.0	0.7	0.8	0.9	0.5	0.8	0.1	0.4
Zipfian	Correlated	Non-Linear	1.0	1.0	1.0	1.0	1.0	0.5	1.0	0.2	0.6	0.2	0.2
Zipfian	Anti-Correlated	Linear	1.0	1.0	1.0	1.0	0.6	0.9	0.8	0.3	0.5	0.8	0.3
Zipfian	Anti-Correlated	Non-Linear	1.0	1.0	1.0	1.0	0.9	0.5	0.7	0.8	0.2	0.0	0.8
Zipfian	Independent	Linear	1.0	1.0	1.0	0.9	0.8	0.8	0.9	0.2	0.8	0.1	0.1
Zipfian	Independent	Non-Linear	1.0	1.0	1.0	1.0	1.0	0.6	1.0	0.3	0.6	0.2	0.3

Table 7: The success measure of four methods in computing the same top value as Brute Force; APX=Approximate, WT=Weight, SCR=Attribute Score; and the four queries, Q1-Q4. For Q4, WT could not be used.

	Functions weights				$\mathbb{A}$			
	Python	R	PHP	JS	Python	R	PHP	JS
t_1	0.83	0.03	0.1	0.03	0	0.67	0.67	0.67
t_2	0.33	0.11	0.22	0.33	1	0.33	0	0
t_3	0.84	0.02	0.06	0.08	0.67	0.67	0.33	0.33
t_4	0.09	0.09	0.64	0.18	0.67	0.67	0.67	0
t_5	0.29	0.14	0.14	0.43	0.67	0	0	0
t_6	0	0	0	0.99	0.67	0	0.67	0
t_7	0	0.97	0.01	0.02	0.67	0	0.67	0.67
t_8	0.68	0.05	0.05	0.23	0.33	0	0.67	0.67
t_9	0.17	0.17	0.17	0.50	0.67	0	0	0.67
t_{10}	0.12	0.04	0.12	0.71	0.67	0	0	0.33
t_{11}	0	0.23	0.18	0.59	0	1	1	1
t_{12}	0	0	1	0.00	0	0.67	1	1
t_{13}	0.2	0.3	0.4	0.10	1	0.33	0	1
t_{14}	0.29	0.14	0.43	0.14	0	1	0.33	1
t_{15}	0.01	0.01	0.09	0.89	0	1	0.67	0
t_{16}	0.7	0.1	0.1	0.10	0	0	0	0
t_{17}	0	0	0.98	0.01	0	0	0	1
t_{18}	0	0	1	0.00	0.33	1	0.67	1
t_{19}	0.11	0.44	0.33	0.11	1	0.67	0	0.67
t_{20}	0.11	0.11	0.67	0.11	0.67	0.67	0	1

Table 8: Sample dataset used in Example 1.1. The function weights for the 20 entities are also present as columns within the table.

is set to 5 for these experiments. Samples for the sampling based algorithm are chosen uniformly, with the sample size varying from 25 to 250 in increments of 25. For this experiment, we consider both, the synthetic datasets and the real world dataset. In each experiment setting, we also run the brute force algorithm to obtain exact Shapley values to measure against. For each of the configurations, we aggregate (average) the error across the d attributes and compute the standard deviation of the errors. The measured average error in Shapley values and standard deviation of the average errors are plotted in Figure 2 for the Candidates dataset. The time consumed by KernelSHAP and sampling based approach are comparable and far better than the brute force.

Among the two approximate techniques, KernelSHAP outperformed the sampling based approach with a similar number of samples as can be seen from the figure. Even though sampling based technique provides us with guarantees on error rate, practically KernelSHAP has a faster convergence rate which can be seen in the smaller variations in the error bars compared to the sampling based approach. Similar results can also be seen for the synthetic dataset with linear (Figure 4) and non-linear functions (Figure 3) .

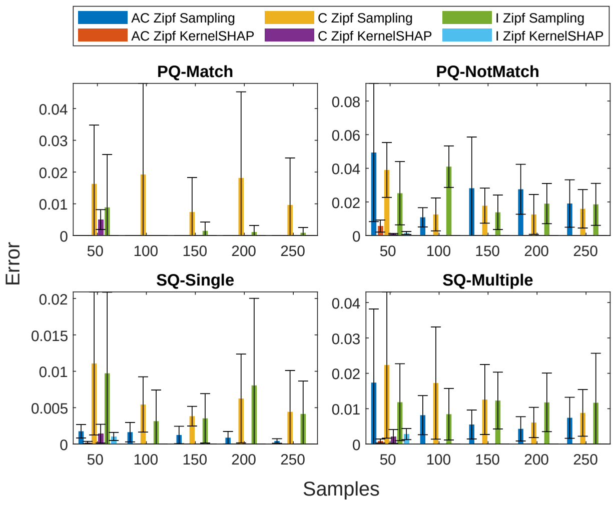


Figure 3: Error variations in sampling-based approach and KernelSHAP when varying number of samples for synthetic dataset with non-linear ranking functions

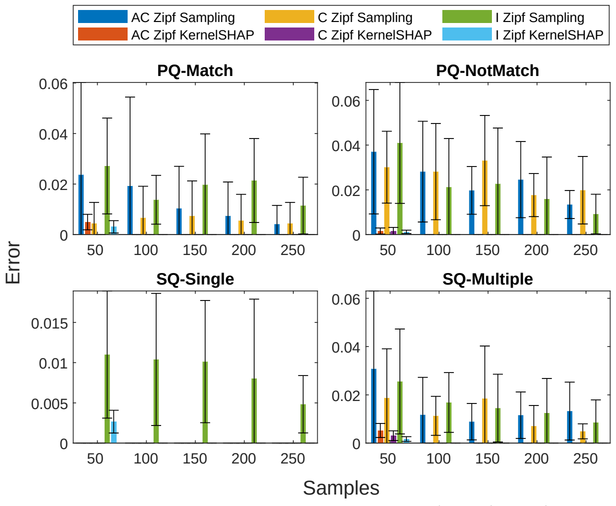
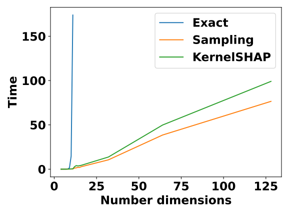
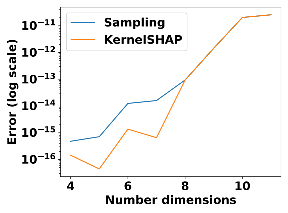


Figure 4: Error variations in sampling-based approach and KernelSHAP when varying number of samples for synthetic dataset with linear ranking functions

4.3.2 Impact of dimensions on time taken and error. In this experiment, we generate synthetic datasets with varying the number of dimensions in the dataset. For each of the datasets we run the exact Shapley approach, sampling based approach



(a) Time taken by the different Shapley approaches when varying the number of dimensions



(b) Measuring error for approximate methods when varying the number of dimensions

and KernelSHAP. Sampling based approach and KernelSHAP both use 100 samples in this experiment. For this experiment, we vary the number of dimensions from 4 to 128. We measure the amount of time taken and error of the Shapley values by the Exact, Sampling-based Approximate and KernelSHAP. As the Shapley values is a vector, the error is measured as the ℓ_2 norm of difference between the Shapley values of Exact and approximate methods. We limit the time of each configuration to 10 minutes. As the Exact approach consumes exponential amount of time with respect to d, computing the Exact Shapley values for larger dimensions is expensive.

Figure 5a shows the time consumed by Exact, Sampling and KernelSHAP as we vary the number of dimensions. Due to the 10 minute time limit on each configuration of this experiment, the expensive Exact approach exceeds the time limits for $d > 11$. For similar number of samples, KernelSHAP and Sampling consumes similar amount of time, with Sampling consuming slightly lesser amount of time. The error incurred during the computation of Sampling and KernelSHAP is shown in Figure 5b. As can be seen, KernelSHAP outperforms Sampling based technique in terms of error. While the plots for KernelSHAP and Sampling seem to overlap as the number of dimensions increases, the values for larger dimensions also show that KernelSHAP slightly performs better than Sampling based technique.

5 RELATED WORK

Matching and applications. Matching plays a critical role in the allocation of resources based on supply and demand like matching a region to medical needs [18], ecosystem services are matched based on changing land use [1], public health needs are matched by health insurance plans [2]. Matching systems rely on different mechanisms to capture preferences the users. Traditionally, these preferences are explicitly specified [19–22].

Explicitly specified matching relies on capturing preference lists from both the parties and then create a stable matching under specific conditions. Based on game theory, Gale and Shapley [23] designed a mathematical framework to attain stable matching and applied it to stable marriage and college admission. Such matching where one party is matched only to one other party is known as stable marriage [24–26]. Similarly, many-to-one stable matching exists with applications such as: hospitals provide employment to many doctors [27] or student–project allocations [28] which was formalized by Roth and Sotomayor [29]. Many-to-many stable matching relies on matching many supply parties to many demand parties, which has applications in D2D-enabled cellular networks [20, 30], collaborator recommendations [31]. While there has been extensive work on explicit matching, to the best of our knowledge there has not been any explain-ability work in these fields.

Often, specifying the complete preference list is prohibitive in big data applications [32]. In such cases, the preference can be obtained via different means. In case of a search engine, users interact with the search engine to express their query until they reach their desired result (web-page). Traditionally TF-IDF [33, 34] and Latent Semantic Analysis [35, 36] based approaches were proposed to solve this problem. Recently, vector representation based approaches [37, 38] convert words into vector form and then use this embedded space representation to process the query. Another type of application where implicit preference is seen is user-item recommendation systems like Amazon[7], Netflix[8]. In user-item recommendation systems, items are recommended with prior interactions of the user with their systems. Numerous techniques like collaborative filtering [39], matrix factorisation [40], auto encoder based representations [41, 42] have been proposed to solve the problem. While some of these works have explanations built within the system [43, 44], to the best of our knowledge we are the first to work on explanations in bipartite matching scenarios.

Explainability in top-k and ranking. Ranking functions are popular for solving multi-criteria optimization. While ranking functions have been studied for many years, study of explainability in ranking has been a recent trend. Verma et al. [45] explain queries based on a sampling approach in the neighbourhood of the query. Gale and Marian explore the topic of explaining ranking in multiple papers. First, in their 2019 paper [46] they assign scores to various attributes based on whether the items in the top-k for all methods are in the top-k for a particular attribute. They also demonstrate that the explanation for a ranking can be used to adjust a ranking function such that attributes are contributing to the amount required. The main difference between their work and ours is that they use this as an explanation for ranking. Top-k is similar but independent from ranking, and additionally our matching are bipartite. The difference is further shown by the fact that these algorithms cannot be easily modified to work for our queries. Next in 2020 [47], Gale and Marian expand upon their initial observations by also considering the weight of different parameters, and considering multiple metrics for the type of ranking produced by the function, namely disparity and diversity. Diversity and disparity in ranking has also been studied by works like [48]. Additionally, some work has been done on responsible ranking function design [48, 49], where the objective is to minimally change the weights in a ranking function to make the generated rankings (top-k) fair and stable.

Why not questions over database queries answer why a certain tuple was not present in the database query output. They were first studied by Chapman and Jagadish [50] followed by a rich field of work known as *why* or *why-not* provenance [51–55]. While *why-not* query tries to explain why a tuple is not present in the query result, our problem looks at a black-box matching system. The notion of *why not* was extended to top-k queries by He et. al. [56] where the problem is to find why a certain tuple is not present in the given top-k query. They also propose an approach to modify the query slightly such that the query data-point is present in the top-k query. Gao et. al. [57] propose the problem of *why not* for reverse top-k queries and also propose modifications to accommodate the point into the top-k. The notion of *why-not* was further extended by Chen et. al. [58] to *why not yet* where the potential solution included modifying the query weights, or

[7]https://www.amazon.com/
[8]https://www.netflix.com/

k slightly to include the query tuple in the top-k. However, there are two fundamental ways in which our work differs from this body of work. All these papers assume (a) linear ranking functions, and (b) the functions themselves are "white boxes", i..e, the weights are known. In contrast, our paper focuses on black box ranking functions, and moreover, these functions do not have to be linear. All we assume is that the attributes in our functions can be masked to observe how their behavior changes. Islam et. al [59] propose the problem of answering *why-not* queries over reverse skyline and dynamic skyline queries. A key difference between our work and [59] is that skyline can be used to find the top-1 over the set of monotone functions. While for a tuple to appear in the skyline it suffices to be in the top-1 of one (any) of the functions in the set, we are interested in a specific (query) function. Additionally, the notion of match list in matching deals with top-k.

Shapley values for explanations. Shapley values were introduced by Gale Shapley [7] to determine the contribution of each player to the success of the overall coalition. Shapley values have also been applied to the problem of explanations with a great degree of success. Here, the contribution of the various features to the overall prediction are calculated as Shapley values, and the Shapley values are used to explain the task [60]. Several notable contributions have been made to this. Štrumbelj et al. [9] devise a Monte Carlo sampling technique for explaining models, in order to avoid the exponential nature of exact Shapley value computation. Lundberg et al. [10] mapped the notion of Shapley values to the problem of model interpretability, introducing SHAP and specifically KernelSHAP which allows for regression-based, model agnostic computation of SHAP values. Lundberg et al. expand upon this notion in 2018 [61] with the TreeSHAP method, which is capable of computing SHAP values for tree based models in polynomial time. While these methods have been studied extensively in machine learning, the problem of explainability in bipartite matching is novel. Kumar et al. [13] have shown certain limitations of Shapley-based methods while explaining machine learning models. The limitations highlighted during the process include (i) out-of-distribution points generated during the masking process, (ii) explanations generated using Shapley are not actionable. We emphasise that bipartite matching is not subject to similar problems due to competition. Nevertheless, we have added a note about improper handling of NULL values during masking process.

6 FINAL REMARKS

The concept of match list is modelled on real world matching websites and applications. Some more practical models may include more complex scenarios which can extend to more data types like non-binary matching values, matching preferences that extend to multiple matching values, probabilistic modelling of preference functions. While we present four different explainability queries that may be encountered in real life, there may be many more of these queries which may be of interest. We consider both these extensions important, and an avenue for future work.

While the model we propose is based on the real world, one might also want to consider other theoretical models of matching, like a complete ranked list by each individual instead of ranking functions. These lists could then be used in a stable marriage algorithm to produce a matching. Such alternate models which may be of theoretical interest can be considered as alternate models and present an opportunity for a thorough theoretical analysis.

7 ACKNOWLEDGEMENTS

The work of Gautam Das was supported in part by grants NSF 2107296 and NSF 2008602 from the National Science Foundation. The work of Abolfazl Asudeh was supported in part by grant NSF 2107290 from the National Science Foundation.

REFERENCES

[1] Sibyl Hanna Brunner, Robert Huber, and Adrienne Grêt-Regamey. A backcasting approach for matching regional ecosystem services supply and demand. *Environmental Modelling & Software*, 75:439–458, 2016.

[2] Robert D. Lieberthal. *Matching Supply and Demand*, pages 145–171. Springer International Publishing, Cham, 2016.

[3] Klaasjan Visscher, Peter Stegmaier, Andrea Damm, Robin Hamaker-Taylor, Atte Harjanne, and Raffaele Giordano. Matching supply and demand: A typology of climate services. *Climate Services*, 17:100136, 2020.

[4] Anjan Goswami, Fares Hedayati, and Prasant Mohapatra. Recommendation systems for markets with two sided preferences. In *2014 13th International Conference on Machine Learning and Applications*, pages 282–287. IEEE, 2014.

[5] Gourab K Patro, Arpita Biswas, Niloy Ganguly, Krishna P Gummadi, and Abhijnan Chakraborty. Fairrec: Two-sided fairness for personalized recommendations in two-sided platforms. In *Proceedings of the web conference 2020*, pages 1194–1204, 2020.

[6] Laurent Muzellec, Sébastien Ronteau, and Mary Lambkin. Two-sided internet platforms: A business model lifecycle perspective. *Industrial Marketing Management*, 45:139–150, 2015.

[7] Lloyd S. Shapley. *A value for n-person games*, page 31–40. Cambridge University Press, 1988.

[8] Dominik Janzing, Lenon Minorics, and Patrick Blöbaum. Feature relevance quantification in explainable ai: A causal problem. In *International Conference on artificial intelligence and statistics*, pages 2907–2916. PMLR, 2020.

[9] Erik Štrumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and information systems*, 41(3):647–665, 2014.

[10] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[11] Abolfazl Asudeh, Nan Zhang, and Gautam Das. Query reranking as a service. *Proceedings of the VLDB Endowment*, 9(11):888–899, 2016.

[12] Abolfazl Asudeh and HV Jagadish. Fairly evaluating and scoring items in a data set. *Proceedings of the VLDB Endowment*, 13(12):3445–3448, 2020.

[13] I Elizabeth Kumar, Suresh Venkatasubramanian, Carlos Scheidegger, and Sorelle Friedler. Problems with shapley-value-based explanations as feature importance measures. In *International Conference on Machine Learning*, pages 5491–5500. PMLR, 2020.

[14] Shaheen S Fatima, Michael Wooldridge, and Nicholas R Jennings. A linear approximation method for the shapley value. *Artificial Intelligence*, 172(14):1673–1699, 2008.

[15] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The collected works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

[16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[17] Mohan S Acharya, Asfia Armaan, and Aneeta S Antony. A comparison of regression models for prediction of graduate admissions. In *2019 international conference on computational intelligence in data science (ICCIDS)*, pages 1–5. IEEE, 2019.

[18] Haiyan Shao, Cheng Jin, Jing Xu, Yexi Zhong, and Bing Xu. Supply-demand matching of medical services at a city level under the background of hierarchical diagnosis and treatment-based on didi chuxing data in haikou, china. *BMC Health Services Research*, 22(1):1–12, 2022.

[19] Yeon-Koo Che, Jinwoo Kim, and Fuhito Kojima. Stable matching in large economies. *Econometrica*, 87(1):65–110, 2019.

[20] Tom Hößler, Philipp Schulz, Eduard A Jorswieck, Meryem Simsek, and Gerhard P Fettweis. Stable matching for wireless urllc in multi-cellular, multi-user systems. *IEEE Transactions on Communications*, 68(8):5228–5241, 2020.

[21] Atila Abdulkadiroglu and Tayfun Sönmez. Matching markets: Theory and practice. *Advances in Economics and Econometrics*, 1:3–47, 2013.

[22] Rustamdjan Hakimov and Dorothea Kübler. Experiments on matching markets: A survey. Technical report, WZB Discussion Paper, 2019.

[23] David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[24] Alvin E Roth and Marilda Sotomayor. Two-sided matching. *Handbook of game theory with economic applications*, 1:485–541, 1992.

[25] Gary S Becker. A theory of marriage: Part i. *Journal of Political economy*, 81(4):813–846, 1973.

[26] Theodore C Bergstrom and Mark Bagnoli. Courtship as a waiting game. *Journal of political economy*, 101(1):185–202, 1993.

[27] Natsumi Shimada, Natsuki Yamazaki, and Yuichi Takano. Multi-objective optimization models for many-to-one matching problems. *Journal of Information Processing*, 28:406–412, 2020.

[28] David J Abraham, Robert W Irving, and David F Manlove. Two algorithms for the student-project allocation problem. *Journal of discrete algorithms*, 5(1):73–90, 2007.

[29] Alvin E Roth and Marilda Sotomayor. The college admissions problem revisited. *Econometrica: Journal of the Econometric Society*, pages 559–570, 1989.

[30] Shenshen Qian, Bowen Wang, Song Li, Yanjing Sun, Yi Yu, and Jingjing Wang. Many-to-many matching for social-aware minimized redundancy caching in d2d-enabled cellular networks. *Computer Networks*, 175:107249, 2020.

[31] Xiangjie Kong, Linyan Wen, Jing Ren, Mingliang Hou, Minghao Zhang, Kang Liu, and Feng Xia. Many-to-many collaborator recommendation based on matching markets theory. In *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 109–114. IEEE, 2019.

[32] Jing Ren, Feng Xia, Xiangtai Chen, Jiaying Liu, Mingliang Hou, Ahsan Shehzad, Nargiz Sultanova, and Xiangjie Kong. Matching algorithms: fundamentals, applications and challenges. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(3):332–350, 2021.

[33] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 29–48. Citeseer, 2003.

[34] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[35] Susan T Dumais et al. Latent semantic analysis. *Annu. Rev. Inf. Sci. Technol.*, 38(1):188–230, 2004.

[36] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57, 1999.

[37] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2333–2338, 2013.

[38] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM international conference on conference on information and knowledge management*, pages 101–110, 2014.

[39] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009, 2009.

[40] Thanh Tran, Kyumin Lee, Yiming Liao, and Dongwon Lee. Regularizing matrix factorization with user and item embeddings for recommendation. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pages 687–696, 2018.

[41] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th international conference on World Wide Web*, pages 111–112, 2015.

[42] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the ninth ACM international conference on web search and data mining*, pages 153–162, 2016.

[43] David Alvarez Melis and Tommi Jaakkola. Towards robust interpretability with self-explaining neural networks. *Advances in neural information processing systems*, 31, 2018.

[44] Stefano Teso. Toward faithful explanatory active learning with self-explainable neural nets. In *Proceedings of the Workshop on Interactive Adaptive Learning (IAL 2019)*, pages 4–16. CEUR Workshop Proceedings, 2019.

[45] Manisha Verma and Debasis Ganguly. Lirme: locally interpretable ranking model explanation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1281–1284, 2019.

[46] Abraham Gale and Amélie Marian. Metrics for explainable ranking functions. In *Proceedings of the 2nd International Workshop on ExplainAble Recommendation and Search (EARS 2019)*, 2019.

[47] Abraham Gale and Amélie Marian. Explaining monotonic ranking functions. *Proceedings of the VLDB Endowment*, 14(4):640–652, 2020.

[48] Abolfazl Asudeh, HV Jagadish, Julia Stoyanovich, and Gautam Das. Designing fair ranking schemes. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1259–1276, 2019.

[49] Abolfazl Asudeh, HV Jagadish, Gerome Miklau, and Julia Stoyanovich. On obtaining stable rankings. *Proceedings of the VLDB Endowment*, 12(3), 2018.

[50] Adriane Chapman and HV Jagadish. Why not? In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 523–534, 2009.

[51] Seokki Lee, Bertram Ludäscher, and Boris Glavic. Approximate summaries for why and why-not provenance (extended version). *arXiv preprint arXiv:2002.00084*, 2020.

[52] Kareem El Gebaly, Parag Agrawal, Lukasz Golab, Flip Korn, and Divesh Srivastava. Interpretable and informative explanations of outcomes. *Proceedings of the VLDB Endowment*, 8(1):61–72, 2014.

[53] Kareem El Gebaly, Guoyao Feng, Lukasz Golab, Flip Korn, and Divesh Srivastava. Explanation tables. *Sat*, 5:14, 2018.

[54] Sudeepa Roy and Dan Suciu. A formal approach to finding explanations for database queries. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1579–1590, 2014.

[55] Eugene Wu and Samuel Madden. Scorpion: Explaining away outliers in aggregate queries. 2013.

[56] Zhian He and Eric Lo. Answering why-not questions on top-k queries. *IEEE Transactions on Knowledge and Data Engineering*, 26(6):1300–1315, 2012.

[57] Yunjun Gao, Qing Liu, Gang Chen, Baihua Zheng, and Linlin Zhou. Answering why-not questions on reverse top-k queries. 2015.

[58] Zixuan Chen, Panagiotis Manolios, and Mirek Riedewald. Why not yet: Fixing a top-k ranking that is not fair to individuals. *Proceedings of the VLDB Endowment*, 16(9):2377–2390, 2023.

[59] Md Saiful Islam, Rui Zhou, and Chengfei Liu. On answering why-not questions in reverse skyline queries. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 973–984. IEEE, 2013.

[60] C. Molnar. *Interpretable Machine Learning*. Lulu.com, 2020.

[61] Scott M Lundberg, Gabriel G Erion, and Su-In Lee. Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888*, 2018.