Intelligently Recommending Key Bindings on Physical Keyboards with Demonstrations in Emacs

Shudan Zhong
University of California, Berkeley
Berkeley, California, USA
sdz@berkeley.edu

Hong Xu[*]
University of Southern California
Los Angeles, California, USA
hongx@usc.edu

ABSTRACT

Physical keyboards have been peripheral input devices to electronic computers since early 1970s and become ubiquitous during the past few decades, especially in professional areas such as software programming, professional game playing, and document processing. In these real-world applications, key bindings, a fundamental vehicle for human to interact with software systems using physical keyboards, play a critical role in users' productivity. However, as essential applications of artificial intelligence research, research on intelligent user interfaces and recommender systems barely relates to key bindings on physical keyboards. In this paper, we develop a recommender system (referred to as EKBRS) for intelligently recommending key bindings with demonstration in Emacs, which we use as a base user interface. This is a brand new direction of intelligent user interface research and also a novel application of recommender systems. To the best of our knowledge, this is the world's first intelligent user interface that heavily exploits key bindings of physical keyboards and the world's first recommender system for recommending key bindings. We empirically show the effectiveness of our recommender system and briefly discuss the applicability of this recommender system to other software systems.

CCS CONCEPTS

• **Information systems → Personalization**; **Recommender systems**; • **Human-centered computing** → *Text input*; • **Software and its engineering** → Software maintenance tools.

KEYWORDS

Intelligent User Interface, Recommender System, Key Binding, Physical Keyboard

ACM Reference Format:
Shudan Zhong and Hong Xu. 2019. Intelligently Recommending Key Bindings on Physical Keyboards with Demonstrations in Emacs. In *24th International Conference on Intelligent User Interfaces (IUI '19), March 17–20, 2019, Marina del Rey, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3301275.3302272

[*]Corresponding author

1 INTRODUCTION

Physical keyboards have been peripheral input devices to electronic computers since early 1970s [2] and become ubiquitous during the past few decades. Despite the dramatic advancements of touch-screen keyboards during the past two decades [20, 21], physical keyboards remain and continue to be essential for productivity in many professional areas, such as software programming, professional game playing, and document processing. Even among academics, physical keyboards have long been and continue to be active research topics [7, 14, 15, 19, 22]. However, to the best of our knowledge, research on applying recommender systems to tasks related to physical keyboards is still missing.

One fundamental way to interact with a computer using physical keyboards is via key bindings. A key binding is one or many key strokes that are bound to trigger one particular functionality in a software system. Key bindings in modern computer systems can be as short as one key stroke, such as Ctrl+c (copy to clipboard) and Ctrl+v (paste from clipboard) in Microsoft Windows, but can also be as long as tens of key strokes, such as cheat codes in many computer games. Key bindings play essential roles in many user interfaces. For computer programmers and article writers, properly set key bindings can boost their productivity. For professional gamers, key bindings directly affect their enjoyment of games and competitiveness in electronic sports. Therefore, it is imperative to create key bindings intelligently.

One famous software system with a user interface that exploits the use of key bindings to an extreme extent, if not the most, is *Emacs*. It was first developed in 1970s and has become a classical family of computer programs for writing programming code since then [5]. It is famous among software developers for its ultra-high extensibility and customizability—i.e., personalization in the terminology of recommender systems. Indeed, our scanning of source code shows that, a default installation of version 26.1 of *GNU Emacs* [18], the most popular member of the Emacs family today, has over 9,000 customization options. (For simplicity, we will refer to "GNU Emacs" simply as "Emacs" throughout the rest of the paper.) This high customizability of Emacs makes it a desirable base for developing an intelligent user interface that focuses on recommending key bindings.

A huge fraction of Emacs' customizability comes from its fully customizable mapping from *Lisp functions*—Emacs' internal representations of its functionalities—to key bindings. In Emacs, users can define their own Lisp functions just like in an usual programming setting: They have control flow and access to APIs provided by Emacs. Due to this high customizability of user-defined Lisp functions, powerful users and extension developers of Emacs generally customize key bindings frequently, especially for Lisp functions that they defined. Furthermore, Emacs also allows much more sophisticated individual key bindings than most other software systems and perhaps all text/programming code editor programs. Therefore, algorithms for recommending key bindings on Emacs' key binding system are also likely to be applicable to most other software systems with user interfaces. For these reasons, we choose Emacs as the base user interface to develop and demonstrate a recommender system for intelligently recommending key bindings.

In this paper, we develop a recommender system for intelligently recommending key bindings with demonstration in Emacs, which we use as a base user interface. We refer to our recommender system as the *Emacs Key Binding Recommender System* (EKBRS). This is a brand new direction of intelligent user interface research and also a novel application of recommender systems. To the best of our knowledge, this is the world's first intelligent user interface that heavily exploits key bindings of physical keyboards and the world's first recommender system for recommending key bindings. The paper is organized as follows. In Section 2, we summarize the structure of key bindings and common approaches for (manually) creating key bindings in Emacs. In Section 3, we describe the details of EKBRS. In Section 4, we empirically evaluate EKBRS. Finally, in Section 6, we conclude our work and briefly discuss the applicability of this recommender system to other software systems.

2 STRUCTURE OF KEY BINDINGS IN EMACS

While Emacs is well known as a text/programming code editor program, its internal is designed as a Lisp machine in which all functionalities are formulated by Lisp code. A *Lisp function* consists of a group of Lisp code that accomplishes a defined goal and its name is by convention a sequence of English words separated by dashes. In Emacs, a *key binding* is a sequence of *key strokes* that invokes a specific Lisp function. For example, a user can use the key binding Ctrl+s to interactively invoke `isearch-forward`, a Lisp function that searches the current buffer (contents in the currently active window).

Emacs has a more sophisticated key binding system than those of most (and perhaps all) other editor programs. Unlike most other text editors, Emacs allows key bindings that consist of multiple key strokes. A key stroke is the action of pushing a *normal key* with or without *modifier keys* and then releasing them. The *prefix* and the *suffix* of a key binding with n key strokes is the sequence of its first $n-1$ key strokes and its last key stroke, respectively. For example, Ctrl+x s is a key binding meaning "pushing Ctrl+x first and releasing them, then pushing s and releasing it." It has two key strokes, in which Ctrl+x is the first key stroke and s is the second key stroke. Its prefix is Ctrl+x and its suffix is s. Ctrl is a modifier key and x and s are normal keys.

To better understand key bindings in Emacs for developing a recommender system, here, we observe and summarize common approaches in Emacs (and its third-party extensions) for manually creating key bindings for a given Lisp function. Combinations of the approaches listed below are also commonly used. All examples are taken from default key bindings.

(1) Use a key binding that consists of the first letter of an English word in the name of the Lisp function. For example, `next-line`/`previous-line` is bound to Ctrl+n/Ctrl+p, where n/p is the first letter of the English word "next"/"previous."
(2) Use a key binding that consists of a key stroke directly related to the meaning of the Lisp function. For example, `split-window-below` is bound to Ctrl+x 2, where 2 relates to the meaning of "splitting the window into two halves."
(3) Use a key binding with a prefix that is conventionally used for a certain category of Lisp functions. For example, Ctrl+h is a common prefix for help seeking Lisp functions: `describe-variable`, a function that displays the description of a variable, is bound to Ctrl+h v; `describe-key`, a function that displays the description of a key binding, is bound to Ctrl+h k.
(4) Use a key binding that locates closely (on the keyboard) to the key binding of a related Lisp function in terms of functionality. This approach is usually applied on non-alphabetical normal keys. For example, `split-window-right` is bound to Ctrl+x 3. This is geometrically close to Ctrl+x 2, which binds `split-window-below`, a Lisp function that relates `split-window-right` in terms of functionality.
(5) To avoid conflicting with existing key bindings, in addition to using one or combinations of the approaches listed above, also add/change modifier keys. For example, `save-buffer` is bound to Ctrl+x Ctrl+s, since Ctrl+x s already binds `same-some-buffers`.

3 RECOMMEND KEY BINDINGS IN EMACS

In this section, we propose a recommender system for recommending key bindings in Emacs. This recommender system recommends a list of key bindings for a given Lisp function based on the existing *key binding database*, which contains a set of Lisp functions and their associated key bindings. These existing key bindings can consist of both default (i.e., set by vanilla Emacs and its extensions) and user-customized key bindings. Therefore, how personalized the recommended key bindings are depends on the amount of key bindings that the Emacs user has customized.

EKBRS consists of two modules: The *word-suffix* (WS) and *function-function* (FF) modules. The WS module scores normal keys in suffices based on the relationship between English words in Lisp functions' names and normal keys in suffices. This is due to Observations (1) and (2) listed in Section 2. Based on similarities between Lisp functions, the FF module scores prefixes (due to Observations (3) listed in Section 2) and non-alphabetical normal keys in suffices (due to Observations (4) listed in Section 2).

We now discuss the two modules of EKBRS in details.

3.1 The word-suffix Module

This module scores normal keys in suffices based on the relationship between English words in Lisp function names and normal keys in suffices. This relationship is important because these normal keys are usually chosen based on the English words in Lisp functions' names, as per Observation (1) in Section 2. This module consists of two submodules: The *prior-word-suffix* and *posterior-word-suffix* submodules. The prior-word-suffix submodule is based on priorly known relationship and the posterior-word-suffix submodule is based on this relationship in the existing key binding database.

3.1.1 *The prior-word-suffix Submodule.*

This submodule is based on priorly known relationship between normal keys and English words in Lisp functions. There are two types of such relationship: Spelling and meaning. A normal key has a spelling relationship to an English word if the normal key is the first letter of the English word (e.g., the example in Observation (1)). A normal key has a meaning relationship to an English word if the English word is the spelling name of the normal key (e.g., the example in Observation (2)). For a Lisp function, this submodule recommends a normal key to be used in the suffix iff the normal key has a spelling or meaning relationship with at least one English word in that Lisp function. All recommended normal keys in suffices have the same score 1.

3.1.2 *The posterior-word-suffix Submodule.*

This submodule is based on the relationship between English words in Lisp functions' names and suffices in the existing key binding database. In this submodule, we first discover this relationship using the existing key binding database and then recommend normal keys in suffices according to this relationship.

We first describe how this submodule discovers the relationship between English words and normal keys in suffices. This relationship is discovered via the correlation between the presence of each English word in Lisp functions and the presence of each normal key in suffices. Formally, for each existing Lisp function ℓ and its key binding b in the existing key binding database $\mathcal{D}$, we let $\pi_w^{\ell,b}$ and $\pi_k^{\ell,b}$ denote the presences of the English word w in ℓ (denoted by $w \in \ell$) and the normal key k in the suffix of b (denoted by $k \in b$), respectively. $\pi_w^{\ell,b} = 1$ if $w \in \ell$ and otherwise $\pi_w^{\ell,b} = 0$; $\pi_k^{\ell,b} = 1$ if $k \in b$ and otherwise $\pi_k^{\ell,b} = 0$. This relationship is defined as the correlation $r(w, k)$ between the two vectors $\boldsymbol{\pi}_w = \left\langle \pi_w^{\ell,b} \mid (\ell, b) \in \mathcal{D} \right\rangle$ and $\boldsymbol{\pi}_k = \left\langle \pi_k^{\ell,b} \mid (\ell, b) \in \mathcal{D} \right\rangle$. Here, we characterize the correlation using Pearson's correlation coefficient [12], a statistical parameter that is commonly used in collaborative filtering systems [8].

From the correlation between every pair of an English word and a normal key, we construct the *word-key graph*, a bipartite undirected edge-weighted graph that characterizes the relationship between English words and normal keys in suffices. Each node in the first partition of this graph represents an English word and each node in the second partition represents a normal key. We refer to these two types of nodes as *word nodes* and *key nodes*, respectively. A word node w is connected to a key node k iff there exists at least one $(\ell, b) \in \mathcal{D}$ such that $w \in \ell$, $k \in b$ and w is the most correlated English word to k among all English words in ℓ, i.e., $w = \arg\max_{w' \in \ell} r(w', k)$. The weight of this edge is the number of such $(\ell, b) \in \mathcal{D}$. An English word w and a normal key k are related if their corresponding nodes in the word-key graph are connected by an edge, and the larger the weight of this edge is, the more strongly w and k are related.

For a given Lisp function ℓ, this submodule considers the subgraph G_ℓ of the word-key graph induced by all English words in the name of this Lisp function and their neighboring normal keys. It scores these normal keys using the sum of weights of all edges incident to them in G_ℓ.

3.2 The function-function Module

Based on similarities between Lisp functions, this module scores prefixes and non-alphabetical normal keys in suffices. For a given Lisp function ℓ, we compute its similarity $s(\ell, \ell')$ between ℓ and each Lisp function ℓ' in the existing key binding database. We then score all ℓ' using its similarity to ℓ, and prefixes and non-alphabetical normal keys are thereafter scored based on their presences in key bindings of all ℓ'.

3.2.1 *Similarity.*

We use two different similarity measures. The first similarity measure is the *Sørensen-Dice coefficient*, which was first proposed in [3] and [17] and is now also commonly used in content-based filtering [6]. It measures the density of overlapped English words in two Lisp function names, i.e., $s(\ell, \ell') = \frac{2 \cdot |\ell \cap \ell'|}{|\ell| + |\ell'|}$.

This similarity measure, unfortunately, by its nature cannot capture the semantics of English words. For this reason, we introduce a second similarity measure that makes use of *word embedding*, which we refer to as the *word embedding measure*. Word embedding is a language model that represents the semantic of each word using a Euclidean vector and is a well established concept in the field of natural language processing [9, 10, 13]. In recent years, word embeddings have also started being applied in recommender systems [11, 16, 23].

For the word embedding measure, we compute the similarity between two Lisp functions as follows. We represent each Lisp function using a vector equal to the sum of all vectors corresponding to the English words in the name, and optionally the documented summary and description, of the Lisp function. The similarity is the cosine of the angle between the vectors representing these two Lisp functions, commonly known as *cosine similarity*.

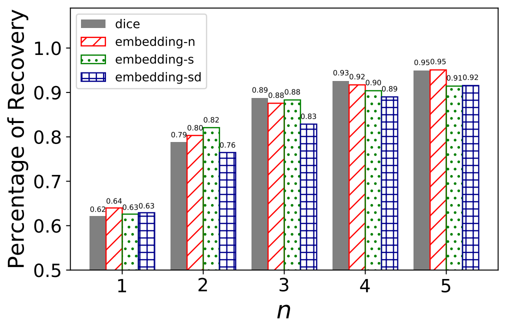
3.2.2 *Scoring.*

The score of each prefix p (or non-alphabetical normal key k in suffix) is $\max_{\ell' \in L} s(\ell, \ell')$, where L is the set of all Lisp functions that have p (or k) in their key bindings. Non-alphabetical normal keys that locate next to k have the same score as k (due to Observation (4) in Section 2).

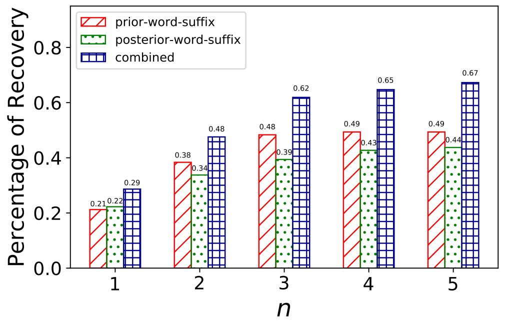
3.3 Scoring Key Bindings

After the two modules score prefixes and normal keys in suffices, we combine these scores to generate scores for key bindings. To combine the scores from the two modules, we first score all normal keys in suffices as the weighted sum of their scores in the prior-word-suffix and posterior-word-suffix submodules and the FF module, and then score key bindings by combining scores of prefixes and normal keys in suffices.

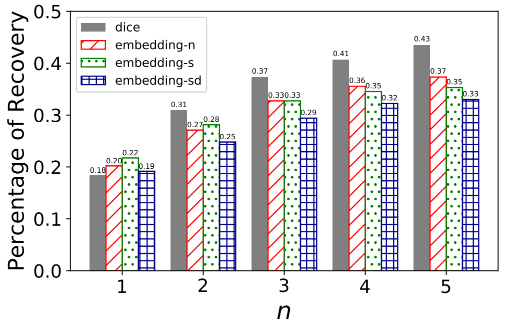
To combine the prefixes and normal keys in suffices scored by the PW and FF modules, we first (a) rescale the score of each prefix and normal key in suffix by dividing it by the sum of the scores of all prefixes and normal keys in suffix, respectively, then (b) score key bindings as the sum of the scores of their prefixes and normal keys in their suffices, and finally (c) recommend key bindings with the highest scores. In step (c), we always recommend a key binding with no modifier keys in its suffix unless this key binding conflicts with an existing key binding (due to Observation (5) in Section 2). In this case, after sorting combinations of modifier keys descendingly with respect to their frequencies in the existing key binding database, we try adding each of them to its suffix until the conflict is resolved.



(a) Prefixes



(b) Normal keys in suffices



(c) Key bindings

Figure 1: Experimental results on recommended (a) prefixes, (b) normal keys in suffices, and (c) key bindings. The x-axes indicate the number of generated recommendations, and the y-axes indicate the percentage of recovery of the recommended items.

4 EMPIRICAL EVALUATION

We empirically evaluated EKBRS on the default global key binding database in vanilla Emacs version 26.1. We removed all key bindings consisting normal keys outside the main area of a qwerty keyboard (except for arrow keys). After this preprocessing, this key binding database consists of 391 key bindings in total. We used a leave-one-out strategy to evaluate EKBRS. In each round, one key binding b and its corresponding Lisp function ℓ were temporarily removed from the key binding database. Then we recommended n key bindings/prefixes/normal keys in suffices for ℓ based on the rest of the key binding database. If b's prefix/normal key in b's suffix is among these n recommendations, then we say that b's prefix/normal key in b's suffix is *recovered*. Since modifier keys in suffices are mostly merely used for avoiding conflicts with existing key bindings (and therefore in general do not affect the quality of a key binding), we say that b is recovered so long as both b's prefix and normal key in its suffix are recovered. The higher percentage of recovery, the more effective EKBRS roughly is (under the assumption that default key bindings are optimally designed).

In our experiments, we set the coefficients to be 2, 1, and 1 for weighted summing the scores of normal keys in suffices generated by the prior-word-suffix submodule, the posterior-word-suffix submodule, and the function-funcion module, respectively. We ranged n from 1 to 5. For the measure of similarity in the FF module, we experimented on both measures (coded `dice` and `embedding`) described in Section 3.2.1. For the word embedding measure, we used the 300-dimensional word vectors pre-trained by GloVe on Wikipedia 2014 and Gigaword 5 [1, 13]. We represented each Lisp function using the sum of all vectors corresponding to the English words in its name (`embedding-n`), its documented summary (`embedding-s`), as well as its documented summary and description (`embedding-sd`) after removing all stopwords in them.

We first evaluate recommended prefixes. Figure 1a shows our experimental results. As n increases to 5, the percentage of recovery becomes greater than 0.9. This means that, for approximately 90% of Lisp functions in the existing key binding database, the key binding configured in the vanilla Emacs was among the first 5 recommendations by EKBRS. In addition, none of the four similarity measures were dominantly advantageous, with `embedding-sd` being slightly weaker than the other three similarity measures. The reason might be that English words in a Lisp function's name and summary are more focused on its core functionality, while its description is more diluted by less relevant English words, such as references to other Lisp functions.

We now evaluate recommended normal keys in suffices. We compared our approach (`combined`) with those that apply either the prior-word-suffix or posterior-word-suffix submodule as baselines. Figure 1b shows our experimental results: A weighted sum of scores of normal keys in suffices generated by the prior-word-suffix and posterior-word-suffix submodules was more effective than each of these individual scores. We do not list the scores of normal keys in suffices generated by the FF module, since non-alphabetical normal keys are rare in the existing key binding database.

We now evaluate recommended key bindings. Figure 1c shows our experimental results. It shows that the `dice` measure was more effective than the other three measures when $n \geq 2$, while the `embedding-s` measure was most effective when $n = 1$, i.e., when only one recommendation was requested.

To demonstrate the effectiveness of EKBRS, we also compared it with a baseline algorithm, which always recommends one key

Table 1: Comparison between EKBRS (with $n = 1$ and the `dice` measure) and the baseline algorithm.

Algorithm	Ours	Baseline with m =						
		1	2	3	4	5	6	7
Percentage of Recovery	0.184	0.128	0.128	0.110	0.115	0.105	0.115	0.107

binding. The baseline algorithm uses an *m-nearest neighbors*[1] strategy. It works as follows. For a given Lisp function ℓ, it finds its m-nearest neighbors L_m using the `dice` measure. It then counts the number of key strokes in the key bindings of each Lisp function in L_m and chooses the number of key strokes in the key binding of ℓ as the one with the largest count. For the key stroke in each position of the to-be-determined key binding, it chooses the one that appears most in the same position in the key bindings of Lisp functions in L_m. Table 1 compares EKBRS to the baseline algorithm with different m's. It shows that our algorithm was more effective than this baseline algorithm in terms of percentage of recovery.

5 DISCUSSION

While we have developed and demonstrated our recommender system, EKBRS, for recommending key bindings in Emacs, the vision of this paper goes far beyond it: We intend to build a framework of recommender system that can intelligently recommend key bindings for a variety of systems with user interfaces. As stated in Section 1, we chose Emacs as the base system to develop and demonstrate such a recommender system simply due to its sophistication: A recommender system for recommending key bindings that works on Emacs is likely adaptable to other common software systems with user interfaces that are less sophisticated in terms of key bindings. In this section, we discuss the applicability of EKBRS to a few classes of common software systems.

5.1 Common Text/Programming Code Editors

The text/programming code editors used the most by software engineers usually have key binding systems that are much simpler than that of Emacs. For example, although popular programming code editors such as Atom[2], Sublime Text[3], and Notepad++[4] are also powerful and customizable, the functionalities that are accessible to users via key binding customization are far more limited than those in Emacs. However, due to their similarities to Emacs in terms of their purpose (editing text and/or programming code), many techniques used in EKBRS are still applicable after proper adaptation.

For example, similar to Emacs, Atom and Sublime Text also map key bindings to functions of their respective internal programming languages, while they have much fewer available pre-defined key bindings and functions than Emacs, and their key binding are also much less disciplined. For these systems, we can adapt EKBRS by weighting down prefixes when scoring due to their simpler sequences of key strokes in key bindings.

However, users on these systems are much less likely to customize key bindings and the usefulness of this adapted recommender system is not as valuable as the one for Emacs. Therefore, instead of focusing on recommending key bindings directly to users, we can use this adapted system to recommend default key bindings in editors and their plugins to their respective developers. In this case, we can even further adapt our system to recommend key bindings for multiple functions all at once by adding a conflict detection component: The same sequence of key strokes should not be recommended to map two different functions. This may be done by integrating a recommender system method such as constraint-based filtering [4].

[1]This is more commonly known as "k-nearest neighbors." Here, we use m instead of k to avoid notational confusion.
[2]https://atom.io
[3]https://www.sublimetext.com/
[4]https://notepad-plus-plus.org/

5.2 Computer Games

Computer games such as WarCraft[5] and StarCraft[6] often require professional gamers to operate fast. Therefore, these games usually provide many default key bindings and professional gamers also need to frequently (re-)customize key bindings.

The structure of key bindings and users' preferences thereto in these computer games are substantially different from those in text/programming code editors, but we can still adapt EKBRS for them. First, professional gamers prefer shorter sequences of key strokes and disfavor modifier keys much more than computer programmers. We can adapt EKBRS by weighting down key bindings with long sequences of key strokes and key strokes with modifier keys.

Secondly, professional gamers prefer sequences of key strokes with related functionalities to be close to each other. For example, keys that control the moving directions of a game character are preferred to be close to each other, since they often need to be triggered chronically closely to each other. We can adapt EKBRS by altering the function-function module to measure similarities between functionalities that are accessible from key bindings, and replace prefixes of sequences of key strokes with a metric of locations on the physical keyboard.

Thirdly, professional gamers also use keys outside their physical keyboards, such as on their mice—for instance, an average professional gaming mouse can have as many as 12 additional physical keys[7]. We can trivially adapt EKBRS to consider additional physical keys. However, in some systems, these additional keys are made accessible by being configured to be equivalent to some key strokes on the physical keyboard. In this case, we can add a conflict detection component to EKBRS and adapt it to also recommend proper configuration to minimize the harm caused by conflict avoidance.

6 CONCLUSION

In this paper, we developed a recommender system, named EKBRS, that is applied on tasks related to physical keyboards. It recommends key bindings for Lisp functions in Emacs, a novel application of recommender systems. To the best of our knowledge, this is the first publicly documented recommender system for tasks related to physical keyboards (and more specifically key bindings). This is also the first intelligent user interface that heavily exploits key bindings of physical keyboards. Based on our observations on the structure of default key bindings in Emacs, our recommender system divides the task of recommending key bindings into scoring prefixes and normal keys in suffices individually, and then combines them. We empirically evaluated the effectiveness of EKBRS and showed that it is more effective than some of its simpler variants as well as a baseline algorithm based on m-nearest neighbors.

[5]https://worldofwarcraft.com
[6]https://starcraft.com
[7]https://www.corsair.com/us/en/Color/scimitar-pro-config/p/CH-9304011-NA

REFERENCES

[1] 2018. https://nlp.stanford.edu/projects/glove/. Accessed: 2018-09-22.

[2] Datapoint Corporation 1970. *Datapoint 3300 / Maintanance*. Datapoint Corporation.

[3] Lee R. Dice. 1945. Measures of the Amount of Ecologic Association Between Species. *Ecology* 26, 3 (1945), 297–302. https://doi.org/10.2307/1932409

[4] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. 2015. Constraint-Based Recommender Systems. In *Recommender Systems Handbook*. Springer, 161–190. https://doi.org/10.1007/978-1-4899-7637-6_5

[5] Bernard S. Greenberg. 1996. *Multics Emacs: The History, Design and Implementation.* Technical Report. http://www.multicians.org/mepap.html

[6] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. 2010. *Recommender Systems: An Introduction.* Cambridge University Press.

[7] Hwan Kim, Yea-kyung Row, and Geehyuk Lee. 2012. Back Keyboard: A Physical Keyboard on Backside of Mobile Phone Using Qwerty. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* 1583–1588. https://doi.org/10.1145/2212776.2223676

[8] Yehuda Koren and Robert Bell. 2015. Advances in Collaborative Filtering. In *Recommender Systems Handbook*. Springer, 77–118. https://doi.org/10.1007/978-1-4899-7637-6_3

[9] Yang Li and Tao Yang. 2018. Word Embedding for Understanding Natural Language: A Survey. In *Guide to Big Data Applications.* Springer, 83–104. https://doi.org/10.1007/978-3-319-53817-4_4

[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *Proceedings of the International Conference on Neural Information Processing Systems.* 3111–3119.

[11] Cataldo Musto, Giovanni Semeraro, Marco de Gemmis, and Pasquale Lops. 2016. Learning Word Embeddings from Wikipedia for Content-Based Recommender Systems. In *Proceedings of the European Conference on Information Retrieval.* 729–734. https://doi.org/10.1007/978-3-319-30671-1_60

[12] Karl Pearson. 1895. Notes on Regression and Inheritance in the Case of Two Parents. *Proceedings of the Royal Society of London* LVIII (1895), 240–242.

[13] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing.* 1532–1543.

[14] Jose Miguel R. Salvo, Christian Jay B. Raagas, Maria Tatjana Claudeene M. Medina, and Alyssa Jean A. Portus. 2016. Ergonomic Keyboard Layout Designed for the Filipino Language. In *Proceedings of the AHFE International Conference on Physical Ergonomics and Human Factors.* Springer, 407–416. https://doi.org/10.1007/978-3-319-41694-6_41

[15] Sidney J. Segalowitz and Roger E. Graves. 1990. Suitability of the IBM XT, AT, and PS/2 Keyboard, Mouse, and Game Port as Response Devices in Reaction Time Paradigms. *Behavior Research Methods, Instruments, & Computers* 22, 3 (1990), 283–289. https://doi.org/10.3758/BF03209817

[16] Donghyuk Shin, Suleyman Cetintas, Kuang-Chih Lee, and Inderjit S. Dhillon. 2015. Tumblr Blog Recommendation with Boosted Inductive Matrix Completion. In *Proceedings of the ACM Conference on Information and Knowledge Management.* https://doi.org/10.1145/2806416.2806578

[17] Thorvald Julius Sørensen. 1948. A Method of Establishing Groups of Equal Amplitude in Plant Sociology Based on Similarity of Species and Its Application to Analyses of the Vegetation on Danish Commons. *Biologiske skrifter* 5 (1948), 1–34.

[18] Richard Stallman et al. 2017. *GNU Emacs Manual* (17th ed.). Free Software Foundation.

[19] James Walker, Bochao Li, Keith Vertanen, and Scott Kuhl. 2017. Efficient Typing on a Visually Occluded Physical Keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* 5457–5461. https://doi.org/10.1145/3025453.3025783

[20] Shumin Zhai. 2017. Modern Touchscreen Keyboards As Intelligent User Interfaces: A Research Review. In *Proceedings of the International Conference on Intelligent User Interfaces.* 1–2. https://doi.org/10.1145/3025171.3026367

[21] Shumin Zhai, Per Ola Kristensson, Caroline Appert, Tue Haste Andersen, and Xiang Cao. 2012. *Foundational Issues in Touch-Surface Stroke Gesture Design: An Integrative Review.* Now Foundations and Trends. https://doi.org/10.1561/1100000012

[22] Haimo Zhang and Yang Li. 2014. GestKeyboard: Enabling Gesture-based Interaction on Ordinary Physical Keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* 1675–1684. https://doi.org/10.1145/2556288.2557362

[23] Lei Zheng, Bokai Cao, Vahid Noroozi, Philip S. Yu, and Nianzu Ma. 2017. Hierarchical Collaborative Embedding for Context-Aware Recommendations. In *Proceedings of the IEEE International Conference on Big Data.* 867–876. https://doi.org/10.1109/BigData.2017.8258002