ACHIEVING INTELLIGENT TASK-BASED MOBILE WIDGET ORGANIZATION AND CUSTOMIZATION THROUGH MACHINE LEARNING TECHNIQUES AND USER MODEL GENERALIZATION

A Thesis
submitted to the Faculty of the
Graduate School of Arts and Sciences
of Georgetown University
in partial fulfillment of the requirements for the
degree of
Master of Science
in Computer Science

By

Benjamin R. Hood, B.S.

Washington, DC
November 15, 2010
ACHIEVING INTELLIGENT TASK-BASED MOBILE WIDGET ORGANIZATION AND CUSTOMIZATION THROUGH MACHINE LEARNING TECHNIQUES AND USER MODEL GENERALIZATION

Benjamin R. Hood, B.S.

Thesis Advisor: Marcus A. Maloof, Ph.D.

ABSTRACT

In a paper by Boström et al., an interface was presented providing rapid access to widget applications on mobile devices. Icons representing widgets were added and removed by users to a “canvas” that enabled them customize the interface to suit their primary objective. Though this implementation was sound and well received, we believed that it could be improved through the combination of two methods grounded in machine learning. These are the generalization of a data set for modeling a default user and a new algorithm named KAWS (K-Based Algorithm for Widget Selection). By accurately predicting potentially desirable widgets and automatically populating the widget canvas, there was potential to mitigate the amount of necessary interaction between the user and device resulting in a diminished physical and cognitive burden. To evaluate the ability of our collected data to form a generalized user model, we ran it against four machine learning algorithms IBK, KStar, Naïve Bayes and J48 using 10-fold cross validation. We found that we were able to achieve an average of 56.9 percent correct class predictions while maintaining a relatively low variance and strong kappa statistic. When compared to a purchasing recommendation system and a personal assistant scheduling system that both use collaborative filtering and machine learning techniques to predict user preferences, our data generalization model was consistent with the two who maintained accuracies of around 50 percent. When this data was subsequently run against our new
implementation, KAWS, we were able to reduce the average amount of requisite interaction by 11 percent when compared to the implementation by Boström et al.
Thank you to my advisor and mentor Mark Maloof for his invaluable guidance and to my thesis committee for all of their time and effort.

BENJAMIN R. HOOD
# TABLE OF CONTENTS

INTRODUCTION ......................................................................................................................... 1  
RELATED WORK ......................................................................................................................... 4  
EXAMPLE SCENARIO ................................................................................................................... 7  
METHOD .................................................................................................................................... 10  
USER MODEL GENERALIZATION .............................................................................................. 10  
KAWS (K-Based Algorithm for Widget Selection) ................................................................. 11  
EXPERIMENTAL STUDY ........................................................................................................... 13  
METHOD 1: MODEL GeneralIZATION ...................................................................................... 13  
ALGORITHMS .......................................................................................................................... 13  
METRICS .................................................................................................................................. 15  
DATA SET ................................................................................................................................. 16  
EXPERIMENTAL PROCEDURE ................................................................................................. 19  
RESULTS .................................................................................................................................. 21  
ANALYSIS ................................................................................................................................. 24  
Comparison of Algorithms ....................................................................................................... 26  
Average Performance ................................................................................................................. 27  
Discussion ................................................................................................................................ 28  
METHOD 2: KAWS (K-Based Algorithm for Widget Selection) ................................................ 30  
COMPARISON OF TECHNIQUES ............................................................................................. 31  
METRICS .................................................................................................................................. 31  
DATA SET ................................................................................................................................ 34  
EXPERIMENTAL PROCEDURE ................................................................................................. 35
LIST OF FIGURES

Figure 1: Capricorn Canvas After One Manual Widget Addition ................................. 8
Figure 2: Intelligent Interface Canvas After One Manual Widget Addition ................. 9
Figure 3: WEKA .arff format .................................................................................. 18
Figure 4: Performance and kappa for varying K values when using the IBK algorithm.. 22
Figure 5: Comparison of algorithms on the data set .................................................. 26
Figure 6: Average Performance and Data Set Growth ............................................... 27
Figure 7: Algorithm input sample .......................................................................... 35
Figure 8: Percent Efficiency for all K values ............................................................ 38
Figure 9: Percent Efficiency for all K values ............................................................ 38
Figure 10: Percent Efficiency for all K values for all manual additions ...................... 39
Figure 11: Percent Efficiency for K=7 across for all manual addition steps ............... 42
LIST OF TABLES

Table 1: Average Performance of the IBK algorithm ....................................................... 22
Table 2: Average Performance of the KStar algorithm .................................................... 23
Table 3: Average Performance of the Naïve Bayes algorithm ......................................... 23
Table 4: Average Performance of the J48 algorithm ........................................................ 23
Table 5: Average Performance of the combined algorithms ............................................ 23
Table 6: IBK Performance when K = 1 ............................................................................ 47
Table 7: IBK Performance when K = 2 ............................................................................ 47
Table 8: IBK Performance when K = 3 ............................................................................ 47
Table 9: IBK Performance when K = 4 ............................................................................ 47
Table 10: IBK Performance when K = 5 ......................................................................... 47
Table 11: IBK Performance when K = 6 ......................................................................... 47
Table 12: IBK Performance when K = 7 ......................................................................... 47
Table 13: IBK Performance when K = 8 ......................................................................... 47
Table 14: IBK Performance when K = 9 ......................................................................... 48
Table 15: IBK Performance when K = 10 ....................................................................... 48
Table 16: KAWS performance for K = 1 ....................................................................... 49
Table 17: KAWS performance for K = 2 ....................................................................... 49
Table 18: KAWS performance for K = 3 ....................................................................... 49
Table 19: KAWS performance for K = 4 ....................................................................... 50
Table 20: KAWS performance for K = 5 ....................................................................... 50
Table 21: KAWS performance for K = 6................................................................. 50
Table 22: KAWS performance for K = 7................................................................. 51
Table 23: KAWS performance for K = 8................................................................. 51
Table 24: KAWS performance for K = 9................................................................. 51
Table 25: KAWS performance for K = 10............................................................... 52
LIST OF ALGORITHMS

Algorithm 1: K-Based Algorithm for Widget Selection (KAWS) ........................................... 12

Algorithm 2: Calculation for Total Work ............................................................................... 33
INTRODUCTION

“...The world is getting more complex, and problems are getting more urgent. These must be dealt with collectively. However, human abilities to deal collectively with complex/urgent problems are not increasing as fast as these problems”

-Doug Engelbart

Over the years developers have primarily concerned themselves with providing functionality to users, even if it comes at the cost of usability. All too often they are driven solely by the functionality of their software and not the means by which it is actually utilized, sometimes leaving users with a heavy cognitive burden consistent with the use of that system [23, 24, 25]. This shortcoming continues to manifest itself as mobile computing becomes increasingly prevalent. As these devices have grown smaller and more powerful their complexity has grown due simply to the fact that they are much more capable than ever before. Devices that were once clumsy and single functioned have now become capable of running full scale applications. This alone has imposed many new design considerations that must be made for each piece of software. Additionally, given the fact that these applications can also be run concurrently, developers must now design an efficient and intuitive method by which to navigate between them. Currently, there are a number of interface variations for mobile devices offering the ability to do so, but one of the best received at this point is the use of icons and widgets [1, 2].

The widget itself is not a new concept as it is, in many regards, a mobile manifestation of the desktop version designed to “encapsulate a specific information
source and provide easy, ubiquitous access to the information provided by that source” [3, p. 327]. The actual implementation, though, has been modified slightly to fit within the mobile paradigm and to make use of the physical technology (i.e. touch screen, smaller display area, etc.). This, however, has generated an entirely new set of problems that have yet to be successfully or satisfactorily answered. One of these newly introduced troubles includes the user’s ability to *quickly find and utilize a particular widget or grouping of widgets necessary for the completion of a primary task*. Given that single widgets do not always maintain all of the required functionality to complete a particular task, users may rely on a number of varying widgets to do so. Current design paradigms do not facilitate this particular situation as, in most cases, they require users to search for each application widget individually costing them valuable time and requiring additional physical interactions. Though there have been some studies that try to mitigate these burdens in which users add suggested widgets in the form of icons to a blank “canvas” [3], it is our opinion that they have still fallen short of what we believe to be acceptable from a usability standpoint.

In our research we proposed two methods that mitigate the burden imposed by widget seeking tasks, especially during initial device usage. Each method utilized machine learning techniques to accomplish these goals.

For our first method, we aggregated a large collection of user data to generate a preliminary default user model. It was our belief that this generalization of user data and subsequent model will mitigate much of the time and overall system utilization typically required for a machine learning algorithm to build a useful model of the new user. To test the ability of the data to form a generalized model we ran it against four established
machine learning algorithms including IBK, Naïve Bayes, KStar and J48. Our results show that the algorithms were able to achieve an average 56.85 percent of correct predictions, well above the minimum 50 percent required to avoid any negative impact on the usability of the interface [4]. Additionally, when compared to other studies in which preferences were predicted through tailored user models, our generalized method performed consistently with them in both cases [26, 27]. This implies that the model can be generalized well to a single user.

It is this implication that acts as the foundation for our second method. By demonstrating the good generalizability of a user model based on historical, aggregated data, we were able to then apply that understanding to the design and execution of our newly developed algorithm KAWS (K-based Algorithm for Widget Selection). This is an important fact to note, as our new algorithm also needed to be able to generalize the user model from a discrete set of data at the outset of device usage to afford the same advantages that our first method highlighted.

The algorithm itself is based on the K-Nearest Neighbor algorithm, a implementation of which (IBK) was used in our user model generalization study. Utilizing the aggregated default user data from the model generalization research, KAWS selects like widgets based on relationships between them regardless of task. Once a probabilistic threshold has been met, the predicted widgets are returned to the user. Good performance of this algorithm results in much less work for the user as they no longer have to search for and manipulate every single widget or widget icon individually given that they are preemptively returned for them.
Our results suggest that the algorithm, and application of user model generalization techniques, reduce the amount of physical interactions by 11.13 percent when compared to the baseline scenario. Thus, by generalizing user data and using the KAWS algorithm to predict useful widgets, we have been able to mitigate a number of obstacles currently associated with accomplishing a generic task on widget-based mobile devices. Ultimately, it is our hope that this will afford an improved overall user experience when utilized in widget-based mobile devices.

In the following sections we will provide context for our study in the related works section, provide a detailed description of our two methods, convey the experimental study for each method including the design, procedure, results and analysis, provide possible avenues for additional research in the future work section and lastly evaluate and reiterate our study as a whole in the conclusion.

**RELATED WORK**

It is important that we have an understanding of the context in which this system is being proposed and, additionally, the underlying methodologies and paradigms that have been employed over time to build this context.

It begins at the highest level with a cross-disciplinary field known as HCI, or Human Computer Interaction. This is generically defined as being “concerned with the design, evaluation and implementation of interactive computing systems for human use…” [20, p. 5]. As this definition suggests, HCI includes a large subset of focus areas across a variety of disciplines including computer graphics, operating systems, human factors, industrial engineering, cognitive psychology and ergonomics [20]. It is the
intersection of these fields that researchers use to develop increasingly efficient and intuitive computer systems and interfaces. These advances have become especially significant as human computer interaction has shifted from “the command line to graphical interfaces to off-the-desktop ubiquitous computing paradigms” [11, p. 31].

Pervasive computing has become increasingly abundant with advancements in mobile technologies. Devices such as cellular phones have become much more than just a means by which to make a phone call. They are now capable of running fully functional applications once reserved for desktop computers. This is evidenced by the significant number of applications available for download (and the millions that have, in fact, been downloaded) [2].

In an editorial for *Personal Ubiquitous Computing*, Luca Chittaro describes the importance of focusing on the interface design for these mobile devices stating that “Users (especially novice ones) will not enthusiastically adopt mobile computing devices if we are not able to prevent the pains and complexities of interacting through very limited input and output facilities” [21, p. 69]. Broken down further, the complexities of device interactions described above, can be identified as the impact that mobility has on the device capabilities and, more importantly, the impact on the cognitive resources of users [22]. Thus, it is imperative that these are not only well understood, but that this understanding is subsequently applied to new designs such that they will fit within the constraints of the mobile device as well as mitigate any cognitive burdens by somehow offsetting difficult interactions [23].

One method that has been particularly successful in these endeavors is the utilization of icons and widgets for mobile device interfaces. In fact, the most popular and
sought after mobile devices employ these design paradigms [1]. Even with these implementations there is room for improvement as they do not adequately address the organization or more specifically, the ability to effectively obtain a desired widget or grouping of widgets. This is a function of the fact that users are forced to search linearly for widgets or applications within the interface and that few, if any, search capabilities are provided. However, research is being conducted to alleviate these issues. Such research includes a specific study published by Boström et al. [3].

In their study we see the implementation of Capricorn, an intelligent widget interface for mobile devices. Their system employs a series of features to streamline widget selection and organization. Fundamentally, our objectives are parallel; however we also believe that there are a number of possible avenues for improvement. It is one of these improvements that will act as both the foundation and inspiration for our research.

After reviewing their study we concluded that their overall approach is sound, as their results suggest. To accomplish widget organization they provide a “canvas” populated with icons representing widgets selected by the user. Each of these icons provides a quick link to a particular widget that may be executed when clicked. These icons are placed on the canvas individually through a process in which the user searches for, or browses for that particular widget in the repository. They augment this process using collaborative filtering techniques to offer potentially useful widgets based on a user’s current selection. Though a “correct” or “helpful” suggestion is useful to users, they are still required to add and remove the widgets individually. This is what we view as the greatest shortcoming of this interface.
Regardless of the technique for selection (browsing, searching or suggested), if a user should wish to change their primary task(s) it could necessitate the addition and removal of many widgets given that it may require an entirely different set to accomplish their new goals. This, in turn, ends up costing the user additional time and effort. It is this that serves as the basis for our own research. We intend to mitigate this shortcoming through the prediction of widget sets based user model learned from previously collected user data. Additionally, our new algorithm, KAWS, will afford the ability to accurately predict and update the screen with the appropriately associated widgets for a particular task. The current implementation as developed by Boström et al. [3] will act as our baseline case in which all new widgets must be manually added to the canvas individually.

**EXAMPLE SCENARIO**

In this section we hope to further solidify our motivations by providing a generic task example in which a user is utilizing a mobile device to facilitate their objective. We will execute the scenario on a device using Capricorn (the implementation by Boström et al.) [3] and then on a device employing an intelligent interface that returns a group of potentially useful widgets to the user based on their initial selection.

*Capricorn*

Starting with an empty canvas, assume that there is a set of five widgets that the user desires for a particular task. To begin, the user must first navigate from the blank canvas to the widget selection screen. At this screen, they must first choose the category...
of the widget that they currently want to add. Once they have selected the category from a drop down menu, the user is given a list of widgets in the category to choose from, as well as a list of suggested widgets. In either case, the user must scan each list until they happen upon the widget that they wish to add. At this point, they must highlight the widget in the list, and then click the button to include it.

Figure 1: Capricorn Canvas After One Manual Widget Addition

These exact steps must be completed 5 different times to add all of the desired widgets to the canvas. As is evident, the user spends a great deal of time searching through the list of widgets for each category; and that is assuming they know the category within which the widget resides. It could be that they select an incorrect category and spend time searching the list without ever finding the desired widget.
Intelligent Interface

In an intelligent interface, the user must still follow the same steps above for the initial widget selection. However, this is where the two implementations diverge. In this implementation the interface makes a prediction based on relationships between widgets and the first widget selection. It subsequently populates the canvas automatically with predicted, and potentially desirable widget icons for the user.

![Blank Widget Canvas](image1.png)  ![After One Addition](image2.png)

Figure 2: Intelligent Interface Canvas After One Manual Widget Addition

If it were one hundred percent accurate, the user would have all the widgets instantaneously, reducing the amount of time and interactions to 1/5\(^{th}\) that of Capricorn. However, given the nature of prediction-based interfaces it is unlikely that the implementation would maintain such a high accuracy. Nevertheless, if it were able to predict even one of the widgets correctly, it would be 20 percent less effort and time required of the user given an initially blank canvas. Logically, the more widgets one has, the more beneficial the intelligent interface would become as a result of the lengthy
Thus, we intend to implement an intelligent widget-based interface utilizing two methods founded in Machine Learning techniques.

**METHOD**

In our study, we have developed two methods by which to mitigate the cognitive and physical burdens placed on the user when utilizing a widget-based interface on a mobile device. These are a user model generalization method for generating a default user model and the implementation of a K-based algorithm for widget selection (KAWS). The following is a description of each method.

**USER MODEL GENERALIZATION**

The first of these is the aggregation and utilization of a generalized user model based on historical data collected from a large pool of users. Based on a successful study by Agichtein et al. in which they used historical user data to build a general user model for best bet web searches [5], it is our belief that this method can be used at the outset of device usage as a default user model. By using this generic, historical data as a training set, we can construct a baseline user model capable of providing accurate widget predictions while minimizing the learning curve typical of online machine learning algorithms. In other words, the collection of many previous users’ widget selections could be utilized to predict desired widgets more accurately earlier on in usage for a new user. By running this initial data set, against several machine learning algorithms and evaluating the performance of each, we can empirically evaluate how well this default
model has generalized. We will call this the ‘generalizability’ of the model. Consistently good performance across all algorithms will indicate good generalizability.

As it turns out the default model did, indeed, generalize well with a performance of 56.85 percent. This surpasses the 50 percent that Findlater et al. define as being the minimum percent accuracy for small-screen, adaptive interfaces to ensure no negative impact on usability or overall user satisfaction [4]. Additionally our results maintain performance consistent with other preference prediction methods in which the user model was tailored to the individual at the outset [26, 27].

**KAWS (K-Based Algorithm for Widget Selection)**

The second method is the implementation of a hybrid K-Nearest Neighbor algorithm capable of predicting the set of preferred user widgets based on the current user selection.

Typically the K-NN Algorithm uses a distance measure between the observation and examples to determine its nearest neighbors, or the K examples that are most closely related to it [12]. Our algorithm is almost identical in that respect. It takes the observation and compares all observation attributes to the attributes for every example in the training set. When an example has a differing attribute value from the observation, the “distance” of that example is incremented. For instance, if an observation had 10 attributes and the example shared 8 of the same values for those attributes, it would have a distance of 2. Once all of the examples have been compared, the top K examples with the smallest distances would be returned. In the event that examples share the same value, the first example(s) compared is used.
It is at this point that our algorithm begins to differ from the standard K-NN implementation. In the standard implementation, the algorithm would simply predict a class (in this case, a task) for the observation given the class majority of its K nearest neighbors [12]. However, in our implementation we assume that class is irrelevant as a function of the fact that tasks can be greatly varying. Assigning classes could potentially result in the pigeon holing of widgets into groups that might not make sense for a particular task. Instead, we concern ourselves with returning the actual predicted attributes (widget selections) of each nearest neighbor based on the current widget selection. This enables the algorithm predict a subset of widgets that might not have been previously related by task. To do this, each of the neighbors is compared. When a particular attribute with a positive value appears in enough of the examples to meet or exceed a designated probability threshold, it is returned as a predicted user selection.

```
1 Get observation
2   For all examples
3     Compare distance of observation to example
4     For each nearest neighbor compare example distance
5     If example distance < neighbor distance
6     Replace neighbor with example
7   End for
8 End for
9
10 For All neighbors
11   For all attributes
12     Compare attributes with values of ‘1’ (positive values)
13     If percentage examples with ‘1’ for this attribute > prob threshold
14     Add index to list
15   End for
16 End for
17 Return list of attribute indexes as predicted widget list
```

Algorithm 1: K-Based Algorithm for Widget Selection (KAWS)

By predicting desired widgets and then preemptively updating the user’s canvas there is potential to greatly improve upon the implementation by Boström et al. [3].
Correct predictions, would reduce the number of interactions necessary to populate the set of preferred widgets. Ultimately by implementing this algorithm and applying our user generalization method to it, we were able to reduce these interactions by 11.13 percent.

**EXPERIMENTAL STUDY**

The following sections will provide a detailed overview of our experimental procedure and analysis for both the user model generalization method and the KAWS algorithm.

**METHOD 1: MODEL GENERALIZATION**

In this section we will discuss the four learning algorithms selected to evaluate the model generalization method, the metrics used in that evaluation, the data set itself, the method by which the experiment was run, the results gathered from the experiment and an analysis of those results.

**ALGORITHMS**

The IBK, KStar, Naïve Bayes and J48 algorithms were selected to evaluate the generalizability of the user model. Each provides a different implementation and method by which the data is utilized for predictions. It should also be noted that each has been implemented in WEKA [14] and as such can be considered both a standard and generally accepted implementation.
The IBK algorithm is the least complicated of the four. The concept description is the stored training data and the learning element stores these examples. The performance element simply calculates the distance between the examples and the given observation. Then for the K-closest instances, it returns a class predication based on the most common class label returned amongst those instances [12].

KStar is similar to IBK in that the concept description, learning element and performance element are the same. The distinction lies with the calculation of the distance as defined in the performance element. IBK typically employs the Euclidian distance between numeric attributes and tally mismatches for symbolic attributes [12]. In contrast, KStar uses entropy. “The intuition is that the distance between instances be defined as the complexity of transforming one instance into another…” [13, p. 4] using a finite set of transformations. The algorithm essentially computes the relevancy of each neighbor with regard to individual attributes by utilizing a blending parameter, the result of which is then used in the complexity calculation. This blending parameter acts similarly to the K value in that it is utilized to determine the relevant neighbors for the observation. A value of 20 percent is typically considered be the standard value for optimal performance [13].

Naïve Bayes is significantly distinct from the two previously described algorithms. Its concept description is comprised of the prior probability of each class and the conditional probabilities of each attribute value given each of the classes. The prior probability is the frequency of examples from each class and the conditional probability is the frequency of each attribute value for each class. These are both derived from the training data. The algorithm then uses Bayes’ rule and calculates the posterior probability
of each class. The class label that has the highest probability is returned as its prediction. When there is a prediction made on new data, that example is subsequently added to the training data and the probabilities are re-calculated. It should be noted that the algorithm’s performance element assumes that the attributes are conditionally independent [14].

The fourth algorithm that we have chosen to use in our model generalization evaluation is J48, which is the Java implementation of Quinlan’s C4.5 [15] decision tree based algorithm. In his algorithm a tree is built recursively, where at each node, examples are split over a single attribute that best organizes them into groups of the same class. This attribute is selected using the normalized gain ratio (defined as the difference in entropy), where the attribute with the highest overall gain ratio is chosen. When building the tree, if all the examples in a subset are of the same class then a leaf node is created for that class. If the algorithm is unable to determine a splitting attribute given no information gain, then it will create a node above using the expected value of that class. And lastly, if a new class is encountered that had not been seen previously, the algorithm will similarly create a node above using the expected value of that class. Again, by using an algorithm that is completely different from the other three, it is our hope that it will strengthen our argument for user model generalization given good and consistent performance across all the algorithms.

METRICS

In this study we will use percentage of correct predictions as our primary performance metric. We will additionally calculate the variance across predictions as well
as the Kappa Statistic. These were selected based on metrics utilized by Calderón-Benavides et al. in their comparison of several predictive algorithms for collaborative filtering [10].

The percentage of correct predictions is simply calculated as the number of correct predictions over the number of total predictions. The variance is then calculated as a measure of an algorithm’s consistencey across predictions in that the larger the variance, the less consistent the algorithm is. Lastly, we use the *kappa statistic* to determine the strength of the correlation between data [16], which gives a measure of the agreement existing between the real and predicted values taking into account chance. As its value increases, so does the agreement. The kappa statistic $\kappa$ is defined as

$$\kappa = \frac{Pr(a) - Pr(e)}{1 - Pr(e)},$$

where $Pr(a)$ is the relative observed agreement among predictions with the class and $Pr(e)$ is probability for chance agreement.

**DATA SET**

For this study we required a large aggregated data set representing user’s widget selection for a series of tasks. To do this, we began by determining the tasks that would be presented to each participant. This was to ensure that our selection of available widgets did not bias task creation. In a further attempt at objectivity, we varied the granularity of each scenario from broad to specific in hopes that this would afford us the
opportunity to better evaluate the generalizability of the model given varying degrees of user flexibility in widget selection. A specific example would be “You are getting ready to pick up you friend from the airport downtown at 5:00pm on Friday evening.” while a less specific example would include “You just moved to a new country and have bought some property.” The final fifteen scenarios can be found in Appendix A.

Once satisfied with the tasks, we turned to the selection of widgets to be provided. Again, in an effort to remain impartial we simply took the top ten free applications available across 18 different categories from the iTunes online store [17]. Though not all of these applications are actually widgets, they function similarly in that they provide quick access to specific information when their icon is clicked. As such, we felt that they were adequately synonymous so as to satisfy our needs and not warrant the risk of prejudice by narrowing our selection to just those meeting the technical definition.

Next, we utilized Amazon’s Mechanical Turk to collect the user data. Each scenario was presented individually as a HIT (Human Intelligence Task) [18]. When a user accepted to participate they would see a particular task and then a table containing the 180 applications broken out by category. Each application had a short description associated with it to inform the users of its functionality. Users would then select those applications or widgets that they would use when accomplishing the given task by simply clicking in a checkbox next to the name of widget. Once completed, they could move on to another HIT or end their session as they were not required to finish all fifteen tasks.

Utilizing the average screen size, icon size and resolution of popular mobile devices [7, 8, 9] in conjunction with a set of HCI guidelines for mobile icon design [6], we derived that
the maximum number widgets available to users at one time should be roughly 16. As such we asked that users limit themselves to selecting 16 or fewer widgets per scenario.

In all, we were able to collect 393 records across the fifteen scenarios. It should be noted that there were some entries in which the user selected more than the maximum number of widgets. These were manually removed from the data set bringing the total number of entries to 385. This data was then manipulated for our two main tasks, testing the generalizability of the user default model and running it against KAWS.

For the model generalization task we arranged the data such that each example had 180 binary attributes, one for each of the available widgets. If a widget was selected then it has a value of 1, otherwise it is a 0. Additionally the scenario, or class, is maintained with each example. A sample of the formatting is shown in Figure 3.

```
@attribute Attribute158 {1, 0}
@attribute Attribute159 {1, 0}
@attribute Attribute160 {1, 0}
@attribute Attribute161 {1, 0}
@attribute Attribute162 {1, 0}
@attribute Attribute163 {1, 0}
@attribute Attribute164 {1, 0}
@attribute Attribute165 {1, 0}
@attribute Attribute166 {1, 0}
@attribute Attribute167 {1, 0}
@attribute Attribute168 {1, 0}
@attribute Attribute169 {1, 0}
@attribute Attribute170 {1, 0}
@attribute Attribute171 {1, 0}
@attribute Attribute172 {1, 0}
@attribute Attribute173 {1, 0}
@attribute Attribute174 {1, 0}
@attribute Attribute175 {1, 0}
@attribute Attribute176 {1, 0}
@attribute Attribute177 {1, 0}
@attribute Attribute178 {1, 0}
@attribute Attribute179 {1, 0}
@attribute Attribute180 {1, 0}
@attribute class {Scenario1, Scenario2, Scenario3, Scenario4, Scenario5,
     Scenario6, Scenario7, Scenario8, Scenario9, Scenario10, Scenario11, Scenario12,
     Scenario13, Scenario14, Scenario15}
@data
1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0
1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
```

Figure 3: WEKA .arff format

Given that the Mechanical Turk output simply displayed the selected widgets by ID for a particular task, we were forced to implement custom code to transform each
record to the WEKA format. This code would simply take the index IDs (returned by Mechanical Turk) and place a value of 1 at those locations as the attribute value. All other attributes (non-selected) were given a value of 0. The scenario associated with the attributes was added to the end of the line.

EXPERIMENTAL PROCEDURE

The data set was run against each of the four algorithms within the WEKA [14] framework; to do so we utilized 10 fold cross validation. In 10 fold cross validation, the data is partitioned and run such that one tenth of the data is randomly selected to be used as the test data while the remaining nine tenths is used as the training data. This is done ten times ensuring that each example has been used as an observation. Additionally, for IBK we tested ten different K values (1 through 10) in an attempt to best characterize performance on the generalized data set for the algorithm.

It should be noted that the observation’s class was being predicted by each algorithm as opposed to the actual widget set as done by KAWS. Given this fact, if good accuracy was achieved by the algorithms, it could be implied that users were selecting similar widgets for the same task. This would then suggest that the user model will generalize well to new users given that they will likely be selecting similar widgets.

Lastly, in an effort to further characterize the performance of our model generalization method we ran the data against the ZeroR algorithm as a reference. The ZeroR, or Zero Rule, algorithm naively classifies all instances to the majority class [14]. Essentially, given our 15 classes representing each of the 15 scenarios, if one were to randomly select a class for an observation given equal class distribution, the probability
that the classification is correct would be 1/15 (approximately 6.7 percent). ZeroR will represent this case, though given the unequal distribution of classes in our data set it could be slightly better or worse given classification based on class majority versus random selection. This performance will then be compared to the other algorithms to ensure that they are better than random classification.

We would have very much liked to include KAWS in our user model generalization experiment to quantify performance relative to the other algorithms. However, given the performance element for KAWS, in which a set of widgets is returned based on nearest neighbor, this would have been difficult as the other algorithms return a single class prediction given their various learning techniques. With single class predictions, the algorithm is either correct or incorrect when compared to the observation, while with KAWS there is potential for partial correctness given an accurate sub-set of predicted widgets. It is this inherent difference that made it difficult to include KAWS as one of the compared algorithms in the user generalization study.

Similarly, it would have been equally as tricky to generate algorithms based on Naïve Bayes or J48 that were capable of predicting widget sets given their performance elements and means by which they classify an observation. Ultimately to manipulate each algorithm to the extent that a comparison to KAWS would be possible, each would have had to be changed such that their performance elements would no longer be synonymous with their former implementations, implying that such a comparison would not be accurate.

Take Naïve Bayes for example, the very fact that it relies on class prior probabilities in its concept description and posterior probabilities for each class in its
performance element implies that it would have been difficult to modify as KAWS assumes class irrelevance and does not consider it when making predictions. For a true comparison to have been made, it would have been necessary to modify it such that the class is arbitrary and predictions may be made with any knowledge of it. Therefore, to modify Naive Bayes or J48 to this extent would have meant that we were essentially comparing algorithms no longer representative of their respective selves.

However, that said, a comparison between KAWS, IBK and KStar would have been reasonable given the fact that KAWS is essentially a modified IBK implementation. All three return nearest neighbors based on a distance which is calculated regardless of class. From these neighbors a set of potentially desired widgets may be selected based on frequency. Given these similarities, and the fact that KAWS was fundamentally derived from IBK, we believe that good user model generalization demonstrated by IBK and KStar would imply that KAWS will exhibit similarly good generalization even though we were not directly able to include it in this experiment.

RESULTS

In the following section we provide the empirical results for each of the four algorithms when run against the data set individually and then as an average of the four.

Beginning with IBK, we attempted to best characterize performance by varying the K value between 1 and 10. In doing so, it was our intention to identify any performance improvement or degradation imposed by the K value itself. As such, our results are broken out by each run of the data against the algorithm with a differing value
of K. These are available in Appendix B. These same results, however, are also depicted in Figure 4.

![Performance and Kappa Statistic for Varying K Values](image)

Figure 4: Performance and kappa statistic for varying K values when using the IBK algorithm.

Given this same data, we then calculated the average percent of correct predictions, the average variance and the average relative observed agreement between predictions and class over the entire data set. We then generated the overall IBK performance table (Table 1).

<table>
<thead>
<tr>
<th>Percentage of Correct Predictions</th>
<th>Variance</th>
<th>kappa Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>48.4% ±1.5032%</td>
<td>±1.9326%</td>
<td>.4448</td>
</tr>
</tbody>
</table>

Table 1: Average Performance of the IBK algorithm

KStar was the next algorithm employed to test our model generalization method. For the *blending parameter* we used a value of 20 percent as this was described to be the standard and also the most appropriate value for a wide variety of data sets [13]. Table 2 shows the results.
The third algorithm to be run the data was Naïve Bayes. Table 3 shows the results.

<table>
<thead>
<tr>
<th>Percentage of Correct Predictions</th>
<th>Variance</th>
<th>Kappa Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.5%</td>
<td>±1.9721%</td>
<td>.6522</td>
</tr>
</tbody>
</table>

Table 3: Average Performance of the Naïve Bayes algorithm

The last algorithm that we ran was the J48 implementation of C4.5. We used a confidence factor of 25 percent for pruning as this is considered to be the standard default value and is considered to work well across a wide variety of data sets [15]. Table 4 contains the performance results for the J48 algorithm.

<table>
<thead>
<tr>
<th>Percentage of Correct Predictions</th>
<th>Variance</th>
<th>Kappa Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>56.1%</td>
<td>±1.6317%</td>
<td>.5371</td>
</tr>
</tbody>
</table>

Table 4: Average Performance of the J48 algorithm

The average performance for all four of the algorithms is shown in Table 5.

<table>
<thead>
<tr>
<th>Percentage of Correct Predictions</th>
<th>Variance</th>
<th>Kappa Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>56.9%</td>
<td>±1.6317%</td>
<td>.5371</td>
</tr>
</tbody>
</table>

Table 5: Average Performance of the combined algorithms
ANALYSIS

The following is an analysis of each algorithm individually and then a comparison of all four to one another. Additionally, we will evaluate the ability, on average, for the model to generalize given these findings.

After having run all ten K values for the IBK algorithm, we assert that the average of the ten values affords a good depiction of overall performance for the algorithm and data set given that each result is close to 48 percent accurate. As depicted in Figure 4 there does not seem to be a trend in either the positive or negative direction for the smaller values of K, which is indicative of relatively consistent performance. However, as the value increases, it becomes clear that performance begins to drop off. The kappa statistic is consistent with this trend. The variance is low, demonstrating consistency across predictions.

As stated in the experimental design, it was our goal to obtain an accuracy of 50 percent or more given its definition by Findlater et al. as the minimum that must be achieved to avoid any negative impact on usability [4]. Subsequently, and somewhat intuitively, they found that as accuracy increased further above that threshold for real estate constrained adaptive interfaces, positive impact on performance and user satisfaction also increased. As such we utilize this threshold as an indicator of good performance for our data generalization tests. As depicted in Table 1, the IBK algorithm fell short by a mere 1.56 percent with regard to accuracy. We believe that this is due to the relatively small number of examples in a high-dimensional feature space. As more neighbors are returned, those that, in a larger data set, would be considered outliers are now being considered in the class majority calculations. This is detrimental in that a
particular example could have a relatively large distance by comparison, but is still referred as a nearest neighbor given that all other examples maintain an even greater distance. Thus, it would seem the small size of the data set is a limiting factor with regard to performance. This becomes especially pronounced in the larger K values.

However, even with this issue, Figure 4 clearly shows that general performance is relatively consistent. Also, it seems that the variance is low overall (around 1.5 percent) meaning that there were few inconsistencies over the predictions. This is encouraging as it implies overall consistency across the data which we previously described to be a good indicator of model generalizability. The kappa statistic seems consistent with the performance in that the agreement between the predicted and actual values is slightly lower indicating a somewhat weaker correlation.

KStar did well in that it achieved better than 50 percent correct predictions. The variance was also low, within 1.5 percent, implying good consistency. Lastly, the kappa statistic closely paralleled the overall performance in strength of agreement between real and predicted values.

With regard to accuracy, Naïve Bayes performed significantly above our 50 percent performance threshold. However, even though accuracy is good, the variance with is slightly high at nearly 2 percent. However, we do not believe that it is significant enough to suggest that the data is overly inconsistent or that the model does not generalize well. The kappa statistic coincides with performance in its agreement between predicted and realized values.
We see that J48 was over 6 percent more accurate than our minimum threshold requirement demonstrating a solid performance. Additionally the variance was small and the kappa statistic seemed reasonable given the number of correct predictions.

*Comparison of Algorithms*

![Comparison of Algorithms](image)

Figure 5: Comparison of algorithms on the data set

*Comparison of Algorithms*

With the exception of IBK, all the algorithms exceeded our minimum threshold of 50 percent accuracy, though it should be noted that the IBK algorithm was short by just over 1 percent. We believe that this was a function of its sensitivity to irrelevant attributes in conjunction with the large feature set and lack of data overall. KStar was the third best performer coming in at 55.3 percent of classes correctly predicted. Though the use of entropy as a distance measure aids in improving the performance, it seems that the algorithm is still somewhat susceptible to the effects of a small data set. J48 was the second best performer, beating out KStar by less than a percent. And lastly, the top performer was the Naïve Bayes algorithm which beat out J48 by an 11.4 percent margin.
Average Performance

As previously stated, it is inferred that consistently good accuracy across the algorithms is indicative of good generalizability as it implies that the data set and subsequent user model are not responsible for any decline in performance. In our study there was a 19 percent difference between the best performing algorithm (Naïve Bayes) and the worst (IBK). We believe that this difference can be attributed to the small size of our data set and the fact that we had 180 attributes in which only a handful of them were relevant. Assuming that the feature set stayed the same size, we hypothesized that a larger collection of user (training) data would reduce this difference and provide a greater number of similar sections within each class and greater variation between classes, which in turn, would provide better distance calculations for IBK and KStar as well as improved splitting in the J48 tree. To demonstrate this we created three additional data sets. The first maintained 25 percent of the original data set, the second, 50 percent and the third had 75 percent. Each was run against the algorithms and averages were plotted as shown in Figure 6.

![Average Performance Across Growing Data Set Size](image)

Figure 6: Average Performance and Data Set Growth
As illustrated in Figure 6, the average percent correct continues to increase as the size of the data set grows. This supports our hypothesis that a larger data set would improve performance.

When compared to the ZeroR algorithm we see that all the algorithms were significantly better. The ZeroR algorithm performance ended up being slightly less than random selection (5.7 percent verses 6.7 percent, respectively) which is likely due to unequal class distribution across the data set. Given this result, our four other algorithms were, on average, 10 times (56.9 percent) more accurate than ZeroR indicating that each was considerably better than random selection for classification.

Discussion

In this section we compare our results to other preference prediction studies to gauge the success of our generalized user model method.

Modeling user behavior has been, and continues to be a popular method for tailoring dynamic interfaces, recommendation systems and navigation schemes to users based on habit [26]. Collaborative filtering and machine learning techniques similar to ours have been one method by which this has been facilitated.

In a study by Iwata et al. [26] shopping behavior was utilized to determine how likely a user would be to purchase a particular item. Prior to receiving an item recommendation, the maximum entropy principle was utilized to determine the probability that it would be of interest to the shopper. If the probability was too low, the item was simply discarded, otherwise it was suggested to the user for purchase.
Predictive accuracies were determined based on whether or not that item was subsequently purchased; a fact maintained in the test data. Recommendations were tested starting with 1 item suggestion, then going up to 5. The maximum predictive accuracy achieved was 45.1 percent and occurred when 5 items were recommended for purchase.

In a separate study by Mitchell [27], an intelligent scheduling system named CAP (Calendar Apprentice) was developed to aid in scheduling tasks. Using an ID3 tree for learning and data collected over time, the tool would “provide interactive advice to the user as they schedule[d] future meetings, or offer[ed] to negotiate specific meetings on behalf of the user” [27, p. 82]. To assess the effectiveness of the tool, the suggestions made by CAP were compared with the actual choices made by the user. The accuracy was measured as a percentage of correct suggestions, or advice given by CAP that was then accepted by the user. Initial tests show that on average, CAP achieved an accuracy of 47 percent.

In both cases, suggestions made based on the user's behavioral model demonstrated an average accuracy around 50 percent. This indicates that our average predictive accuracy of 56.9 percent across our 4 algorithms is consistent with these other studies. It also further suggests that a generalized user model can be effectively used to make predictions for a new system user given that it was able to perform on par with models from the other two studies that were specifically tailored to each individual user.

Lastly, in a study conducted Findlater et al. [4], adaptive menus based on user behavior were deployed on mobile devices in an effort to quantify their effectiveness on small scale system. Their results showed that adaptive menus with high accuracy had a large positive impact on performance and user satisfaction when screen real estate was
constrained. They also found that there was little performance risk associated with the use of adaptive menus in small screens so long as a minimum accuracy of 50 percent was maintained. Thus we made it a goal to achieve a higher accuracy so as to mitigate any negative impact this method might have on usability. Given that our average accuracy was well over this mark at 56.9 percent, it can be inferred that our user model generalization method satisfies this requirement.

Thus, the overall performance of the algorithms was encouraging. The fact that they were able to maintain consistent performance during classifications based on the different scenarios (Appendix A) indicates that a general user model can be successfully applied to a default user.

METHOD 2:

KAWS (K-Based Algorithm for Widget Selection)

PERFORMANCE CHARACTERIZATION

In this section we will compare the manual widget selection method as implemented in the Boström et al. [3] interface with our method. Additionally we will describe the metrics used to evaluate performance for this method, the data set utilized in our evaluation, the experimental procedure and then the results and subsequent analysis of those results.
COMPARISON OF TECHNIQUES

Though Boström et al. [3] present an interface for widget selection and organization that improves what currently exists in the market (e.g., providing suggestions based on a user’s previously selected widgets), they still require users to individually add and remove each widget. This means the selection of \( n \) widgets would require, at a minimum (assuming the user does not first have to remove widget icons to make space), \( n \) user interactions (or units of work as defined in the Metrics section). However, KAWS predicts a user’s desired widgets and, based on that user’s selection preemptively populates the remaining widgets thus mitigating the number of interactions required by the user. For example, if the algorithm were able to achieve an percentage of correct predictions at 50 percent then the device would only require \( n / 2 \) user interactions to obtain the desired widgets (again, this assumes that there is no requirement to remove any of the predicted widgets given space constraints). Ultimately our algorithm was able to achieve an 11.13 percent improvement over the Boström et al. implementation’s best case (our worst case) scenario as described above.

METRICS

In the field of HCI, interactions between users are a standard means by which to characterize performance. These may include specific interactions such as mouse clicks [28] where as others provide more general metrics such as Savings, defined to be the “...manual effort that would have otherwise been required” [29, p. 2].

For our performance metrics we used variations on a single metric that we call “work”. This is more closely related to the savings metric above in that it is an abstract
interaction within the software that could require one or more physical actions (screen touches, screen swipes, etc.). It is defined as the number of interactions that a user must execute to obtain a canvas containing all the widget icons that they wish to use for a particular task. There are two distinct interactions possible. The first is simply adding a widget to the interface. In this interaction a user selects the desired widget and adds its icon to the canvas. This counts as one unit of work. The other interaction is a specific case in which the user wants to add a widget, but all 16 available spots on the canvas are already populated. In this scenario a user may choose to remove one of the widgets, making room for the new one. This removal is also considered to be a single unit of work. Given this definition, we maintain two metrics: The amount of total work and the percent efficiency.

The amount of total work is the number of interactions, or units of work, that must be completed by the user after their manual selection has been made and the set of widgets have been predicted and returned. For this calculation there are four variables:

- **obsCnt** – the number of widgets in the observation
- **corrCnt** – the number of correctly predicted widgets returned by our algorithm not already a part of those manually selected by the user.
- **manCnt** – the number of correctly predicted widgets returned by our algorithm that have already been added manually.
- **retCnt** – the total number of widgets returned by our algorithm
- **work** – the total amount of manual work required by the user after predictions have been made.

Algorithm 2 defines the calculation for total work.
Algorithm 2: Calculation for Total Work

As evidenced in Algorithm 2, this calculation is conditional. The first case occurs when the number of widgets that must be manually added by the user to get to their desired set is greater than the remaining open spaces on the canvas. Such a case could occur if the prediction utilized all 16 spots on the canvas but did not have 100% correct predictions resulting in a necessary removal of incorrectly predicted widgets to make room for the correct ones. Thus, the total work remaining in this case is calculated to be the sum of the number of widgets that must be added manually and the number that must be removed while accounting for those that were returned, but had already been manually selected by the user. The second case is far simpler in that no widgets must be removed given the fact that there are enough open spaces on the canvas to manually add those that were not predicted. Work in this scenario is simply the difference between the observation and the correctly predicted widgets (discounting those already selected by the user).

Percentage efficiency is calculated as the percentage of work completed by our algorithm’s prediction as compared to the worst case in which a user would have to add all the widgets manually. In other words, it is the percentage of work that the user will not have to do to obtain 100% of their desired widgets. For example, an efficiency of 20 percent means that a user is only required to do 80 percent of the total work they would
have otherwise had to do. The greater the efficiency percentage, the less work required of the user. Thus any positive percent efficiency can be considered an improvement. However, if a negative percentage were returned, it would imply that a user must do more work than what is typically required to add all the widgets manually. One example of this might include a situation in which the algorithm predicted a large, mostly incorrect set. Likely, such a scenario would result in the user having to remove some number of the incorrectly predicted widgets so that they could then add the correct selections. Unlike the more complex calculation for total work, the calculation for percent efficiency can be defined in a simple equation:

\[ e = 1 - \left( \frac{w}{o} \right) \]

where \( e \) is the percent efficiency, \( w \) is the work and \( o \) is the number of widgets in the observation.

Given these two definitions we calculated the averages for each across the entire data set.

DATA SET

We utilized the same Mechanical Turk output collected for the data generalization study. By modifying the same code that was used to generate the WEKA data we were able to update the format slightly to fit our new needs. Initially we had intended to run studies that would isolate users and scenarios to see how each set performed individually, but ultimately ended up only running the data set as a whole. As such all that was required was the comma separated attribute values given the assumption that class and user are arbitrary. An example of the data is shown in Figure 7.
Again, given the KAWS algorithm’s fundamental closeness to IBK and KStar, we decided that the application of the same aggregated data set used in our generalization study could be utilized to achieve similar advantages observed when running the data against those two algorithms.

EXPERIMENTAL PROCEDURE

For this task, it was imperative that the data be run to mimic a user selecting a single widget at a time. This was so that we could identify how much work was required before our algorithm could accurately predict the remaining widgets that the user would want. In other words, it would quantify the number of widgets a user must select before an accurate predication may be made. As far as the data itself, this meant that an example having \( n \) positive attributes would have to have \( n \) entries. Take, for instance, an example with 8 attributes. Imagine that widgets 1, 4, 5 and 7 have positive values (meaning they were selected by the user). The initial state of that example before processing would resemble the following:

```
Scenario1 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
Scenario1 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
Scenario1 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
Scenario1 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Figure 7: Algorithm input sample
1, 0, 0, 1, 1, 0, 1, 0

This output would be iteratively processed and run against our algorithm. The following depicts this iterative execution for the example above:

1, 0, 0, 0, 0, 0, 0, 0
1, 0, 0, 1, 0, 0, 0, 0
1, 0, 0, 1, 0, 0, 0, 0
1, 0, 0, 1, 0, 0, 0, 0
1, 0, 0, 1, 0, 1, 0, 0
1, 0, 0, 1, 0, 1, 0, 0
1, 0, 0, 1, 0, 1, 0, 0

Each line simulates a new widget being added to the interface by the user when updating their widget icon canvas.

The above is executed in a leave-one-out batch learning exercise. When running the data set we remove a single example from the set to use as the observation. What remains, is then considered to be the training data. This was done for each example over our entire data set.

*K-Value Selection*

Given that KAWS is based on K nearest neighbors our implementation utilizes a K-value to determine the closest examples in the training data to each observation. We varied this value between 1 and 10 to best characterize the algorithm for analysis.
Threshold Selection

One item warranting discussion prior to conducting further analysis is the probability threshold. We found that limiting the widgets returned by a percentage of probability was severely detrimental. In fact, it was so unfavorable that it produced predictions requiring more work from the user than our worst case scenario. In this situation a user would essentially be required to add all the widgets by hand and remove those that were predicted as few, if any, were correct. We believe this to be a function of a sparse data set. The fact that there was such a large feature set and so few relevant attributes combined with the fact that there were relatively few total instances resulted in nearest neighbors sharing only a select few widgets (often times only one or two). Even then, only a small subset of the neighbors would share these in common, meaning that the percentage of neighbors who did so was so small that it didn’t meet the threshold. Given this problem, we varied the percentage required and found that zero was optimal. By removing the probability threshold all together, each of the selected widgets (positive attribute values) included in the union of all nearest neighbors were returned as our algorithms prediction set. This appeared to mitigate some of the problems associated with the sparse data as we were able to reduce the amount of overall work instead of increasing it.

RESULTS

As depicted in the tables, we were able to achieve an average performance of 5.1 percent efficiency and a maximum performance of 11.13 percent efficiency. The
individual results for each K value can be found in Appendix C, though Figures 8, 9 and 10 illustrate the results more concisely.

Figure 8: Percent Efficiency for all K values

Figure 9: Percent Efficiency for all K values
Figure 10: Percent Efficiency for all K values for all manual additions

ANALYSIS

Average Performance

As shown in our results and evidenced by Figure 8, the average performance reaches its best performance when K is equal to 4 achieving a percent efficiency of 5.1 percent. However, as the value for K increases, the average performance begins to drop significantly. We believe that this overall decline in performance can be attributed to a combination of factors that result in a ripple effect in which each issue affects the next. These factors include the K value and data set size, the number of positive attributes returned for each neighbor and the K value combined with manual additions.

K-Value and Data Set Size

As the value of K increases, the number of potentially detrimental examples increases. In other words, the more neighbors the algorithm returns, the more likely it is that those neighbors will have widget selections that are not relevant to the user’s task.
given their manual selection(s). As stated in the data generalization evaluation, we believe this to be mostly a function of insufficient closeness between the observation and each neighbor resulting in relatively distant examples being returned as neighbors.

Increasing Positive Attributes for Each Neighbor

As more neighbors are returned due to an increasing K value, it becomes more likely that there will be an increase in the number of positive attributes being returned in the prediction. And since we did not impose a probability threshold for predictions, the greater number of incorrect predictions can hurt performance significantly by requiring their subsequent removal in an effort to free up much needed room on the canvas. It is for these reasons that we believe the best average performance was limited to a percent efficiency of 5.1.

Early (as shown in Figures 8 and 9), it seems that there is some correlation between the average performance and the maximum efficiency achieved. However, as the value for K increases, the average performance declines quickly when compared to the maximum. And though their trends remain somewhat similar in shape, the maximum performance for each K remains substantially higher. It is our belief that this disagreement between the two measures is a function of the manual addition values when assessed for each K value.

K-Value and Manual Additions

As shown in Figure 10, it is evident that the manual additions become increasingly detrimental as their number rise and the value for K increases. As the
manual additions grow there is obviously an inherent rise in the amount of work as the
users are manually adding more widgets. This, when combined with the issues described
above (lacking data set size, etc), only serves to further negatively impact overall
performance. Thus, it is for these reasons that we see a decreasing trend in average
performance.

Maximum Performance

When we look at the maximum performance achieved for each K-value when
paired with a particular number of manual additions (instead of the average of all manual
addition values) we see a dramatic increase in the percent efficiency. This is shown in
Figure 9. Since the average includes both the optimal combinations of K and manual
additions as well as the less accurate combinations, it is logical that Figure 9 (maximum
performance achieved) would depict a greater percent efficiency for each K value.

Given the fact that the data set we used in our tests was discrete, and the
assumption that the initial data set in the real-world system would be discrete, we could
adjust our algorithm to employ the K value that realized the maximum efficiency. This
would better tailor it for the outset of device usage. As can been seen in Figure 9 the K
value that achieved the best performance was 7. Figure 11 shows the performance for K =
7 across the 8 manual additions.
According to Figure 11, a manual addition of 1 widget affords the best performance. This is good in the sense that it requires the minimal amount of work possible for each user prior to predicting desirable widgets (as they only have to select 1). Given this, the maximum percent efficiency achieved by our algorithm combined with our data set was 11.13 percent. This implies that the overall work required by a user to get all their desired widgets can be reduced by 11.13 percent from the 100 percent that would otherwise be required in the Boström et al. implementation [3]. This improvement should, in turn, result in less overall time and effort required of the user when updating the interface.

**FUTURE WORK**

There are certainly a number of avenues that we would like to pursue with regard to future work, the first and most obvious being the collection and manipulation of a real data set. As a result of the impact that data set size had on the algorithms utilized for model generalization, we would like to gather a significantly larger pool of real data, pre-
process the training data to remove irrelevant attributes and then run each experiment again. As shown in Figure 6, such steps could garner considerable gains in performance.

Though it could be designed to begin with a clean canvas every time, the current interface implementation by Boström et al. [3] is always populated with the user’s previous widget selections. This implies that there is potential for work as might be manifested in the removal of widgets prior to adding any new ones. In other words, if a user had a shift in their primary task and wanted an entirely new set of widgets, it is likely that they would first have to remove some to make room for the new ones. This preparatory work was not considered in our experiment, though if it were it could mean that the manual addition of all widgets would no longer be the worst case scenario. Instead, it would be the removal of all widgets and then the addition of those desired, effectively doubling, or at least adding to the minimal amount of manual work. We believe that in this case the automatic prediction and population of widgets as provided by our interface would afford even further performance improvements.

Lastly, we would like to test our new algorithm as an online learner in conjunction with the data set used for generalization. By augmenting the initial model with the new user’s habits, it would stand to reason that performance would improve over time on a per user basis. Given this, we would want to quantify such improvements by tracking accuracy in predictions over time.

In all, it is clear that there is room for improvement. However, we ultimately believe that the integration of our machine learning techniques with the foundation laid by Boström et al. [3] has the potential to provide a truly improved and vastly superior widget-based interface for mobile devices than what currently exists today. Furthermore,
and on a considerably larger scale, it is evident that ubiquitous computing is quickly becoming more prevalent. As such it is supremely important that we continue to research ways to enhance the user experience through the seamless and elegant integration of such systems into everyday life.

CONCLUSION

In this study we began with a paper by Boström et al. [3], in which an interface was presented providing rapid access to widget applications on mobile phones. Icons representing widgets were added and removed by users to a “canvas” that enabled them to customize the interface to suit their current needs or primary objective. Though this implementation was sound and well received, we believed that it could be improved through the combination of two methods grounded in machine learning that would enable the system to predict useful widgets and automatically populate the canvas resulting in a greatly diminished cognitive and physical burden on the user.

The first of these methods was the use of a generalized user model capable of closely representing an initial user on the device. Such a representation would mitigate the learning curve typically required of machine learning algorithms while still affording accurate predictions. Thus, we collected a data set maintaining the widget selections for 15 different tasks and a number of different users and ran them against 4 machine learning algorithms (IBK, KStar, Naïve Bayes and J48) to test the generalizability. The data fared well returning an average 56.1 percent correct predictions, which was above our goal of 50 percent even though the average appeared to be brought down by the IBK and KStar algorithms as a function of their sensitivity to small data set size. However,
when compared to other preference prediction implementations [26, 27] our method was on par with regard to performance, indicating that the model could, indeed, generalize well to a default user. Overall improvements could be realized by both growing the set size and perhaps reducing the number of irrelevant attributes through various machine learning techniques prior to execution.

Building on results from our first method, we developed a new algorithm, KAWS, based on the K-NN algorithm in which nearest neighbors were calculated and returned based on their distance from the observation. Once identified, their widget selections were used to predict potentially desirable widgets for the user which, if implemented in a real-world interface, would be automatically displayed. To quantify performance we defined two metrics, total work and percent efficiency where total work is the number of interactions necessary to both add and remove widget icons from the user’s canvas and the percent efficiency is the amount of work over the worst case scenario subtracted from one. In other words, it was the percent of total work that they did not have to do. Given the KAWS implementation’s inherent closeness to IBK and KStar, and those algorithms relatively good performance in the model generalization study, we utilized the same data set in an effort to achieve similar advantages. Employing this data, we ran the algorithm with a range of different values for K (1-10) while varying the number of manual additions from 1 to 8. We ultimately found that the algorithm suffered from the same issues as the IBK algorithm from the previous method. The main issue was the utilization of a large feature set and an undersized number of attributes shared by neighbors. Ultimately, however, we were still able to improve upon the current implementation by Boström et al. [3], and reduce the amount of work by over 11 percent.
APPENDIX A - Scenarios

1) You wish to make travel and accommodation arrangements for a conference that you are attending in Japan.

2) You wish to make plans with an old friend to meet for dinner and then see a movie.

3) You are visiting Washington D.C. and wish to find activities to occupy a day of your stay in the city.

4) You own a small business and wish to check on the status of a major transaction.

5) You are a university student working on an art history project and are currently at the museum. You wish to make the most of your visit using your handheld device.

6) You are attending a concert of your favorite musician and want to make the most of the experience.

7) You are headed north to go on a ski vacation. Due to a snow storm you are stuck in traffic and want to find an alternate route. Additionally you would like to find a place to stop for a rest.

8) It is your friends daughters birthday and you wish to buy her a book from her favorite series on your way home from work.

9) You just won the lottery and have picked up your prize money.

10) You are on the subway and late for work.

11) You are planning a dinner party for a number of close friends.

12) You are attending a PGA golf tournament and want to make the most of the experience.

13) You were riding home in a taxi, and left your laptop on the seat. You wish to get it back.”

14) You are getting ready to pick up your friend from the airport downtown at 5:00pm on Friday evening.

15) You just moved to a new country and have bought some property.
### APPENDIX B - IBK Performance tables for K values 1 through 10

<table>
<thead>
<tr>
<th>Percentage of Correct Predictions</th>
<th>Variance</th>
<th>Kappa Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>50.7%</td>
<td>1.4152%</td>
<td>.4716</td>
</tr>
<tr>
<td>Table 6: IBK Performance when K = 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Percentage of Correct Predictions</th>
<th>Variance</th>
<th>Kappa Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>48.9%</td>
<td>1.5194%</td>
<td>.4495</td>
</tr>
<tr>
<td>Table 7: IBK Performance when K = 2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Percentage of Correct Predictions</th>
<th>Variance</th>
<th>Kappa Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>48.8%</td>
<td>1.4345%</td>
<td>.4524</td>
</tr>
<tr>
<td>Table 8: IBK Performance when K = 3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Percentage of Correct Predictions</th>
<th>Variance</th>
<th>Kappa Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>49.9%</td>
<td>1.4365%</td>
<td>.4636</td>
</tr>
<tr>
<td>Table 9: IBK Performance when K = 4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Percentage of Correct Predictions</th>
<th>Variance</th>
<th>Kappa Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>49.6%</td>
<td>1.4406%</td>
<td>.4609</td>
</tr>
<tr>
<td>Table 10: IBK Performance when K = 5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Percentage of Correct Predictions</th>
<th>Variance</th>
<th>Kappa Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>48.8%</td>
<td>1.5743%</td>
<td>.4526</td>
</tr>
<tr>
<td>Table 11: IBK Performance when K = 6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Percentage of Correct Predictions</th>
<th>Variance</th>
<th>Kappa Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>47.5%</td>
<td>1.6140%</td>
<td>.4388</td>
</tr>
<tr>
<td>Table 12: IBK Performance when K = 7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Percentage of Correct Predictions</th>
<th>Variance</th>
<th>Kappa Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>47.5%</td>
<td>1.5035%</td>
<td>.4389</td>
</tr>
<tr>
<td>Table 13: IBK Performance when K = 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Percentage of Correct Predictions</td>
<td>Variance</td>
<td>Kappa Statistic</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------</td>
<td>-----------------</td>
</tr>
<tr>
<td>47.3%</td>
<td>1.5075%</td>
<td>.4361</td>
</tr>
</tbody>
</table>

Table 14: IBK Performance when K = 9

<table>
<thead>
<tr>
<th>Percentage of Correct Predictions</th>
<th>Variance</th>
<th>Kappa Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>45.7%</td>
<td>1.5869%</td>
<td>.4195</td>
</tr>
</tbody>
</table>

Table 15: IBK Performance when K = 10
APPENDIX C - KAWS results for all K values across 8 manual additions

### K = 1

<table>
<thead>
<tr>
<th>Number of Manually Selected Widgets</th>
<th>Average Number of Desired Widgets</th>
<th>Average Amount of Work Remaining</th>
<th>Average Percent “Efficiency”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.091</td>
<td>7.075</td>
<td>0.36%</td>
</tr>
<tr>
<td>2</td>
<td>7.091</td>
<td>7.031</td>
<td>1.38%</td>
</tr>
<tr>
<td>3</td>
<td>7.091</td>
<td>6.983</td>
<td>1.76%</td>
</tr>
<tr>
<td>4</td>
<td>7.091</td>
<td>6.936</td>
<td>2.36%</td>
</tr>
<tr>
<td>5</td>
<td>7.091</td>
<td>6.973</td>
<td>1.85%</td>
</tr>
<tr>
<td>6</td>
<td>7.091</td>
<td>6.973</td>
<td>1.84%</td>
</tr>
<tr>
<td>7</td>
<td>7.091</td>
<td>6.997</td>
<td>1.60%</td>
</tr>
<tr>
<td>8</td>
<td>7.091</td>
<td>6.997</td>
<td>1.63%</td>
</tr>
</tbody>
</table>

Table 16: KAWS performance for K = 1

### K = 2

<table>
<thead>
<tr>
<th>Number of Manually Selected Widgets</th>
<th>Average Number of Desired Widgets</th>
<th>Average Amount of Work Remaining</th>
<th>Average Percent “Efficiency”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.091</td>
<td>6.992</td>
<td>1.74%</td>
</tr>
<tr>
<td>2</td>
<td>7.091</td>
<td>6.882</td>
<td>3.39%</td>
</tr>
<tr>
<td>3</td>
<td>7.091</td>
<td>6.866</td>
<td>3.55%</td>
</tr>
<tr>
<td>4</td>
<td>7.091</td>
<td>6.815</td>
<td>4.09%</td>
</tr>
<tr>
<td>5</td>
<td>7.091</td>
<td>6.879</td>
<td>3.23%</td>
</tr>
<tr>
<td>6</td>
<td>7.091</td>
<td>6.901</td>
<td>2.90%</td>
</tr>
<tr>
<td>7</td>
<td>7.091</td>
<td>6.903</td>
<td>2.88%</td>
</tr>
<tr>
<td>8</td>
<td>7.091</td>
<td>6.944</td>
<td>2.53%</td>
</tr>
</tbody>
</table>

Table 17: KAWS performance for K = 2

### K = 3

<table>
<thead>
<tr>
<th>Number of Manually Selected Widgets</th>
<th>Average Number of Desired Widgets</th>
<th>Average Amount of Work Remaining</th>
<th>Average Percent “Efficiency”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.091</td>
<td>6.785</td>
<td>5.13%</td>
</tr>
<tr>
<td>2</td>
<td>7.091</td>
<td>6.726</td>
<td>5.69%</td>
</tr>
<tr>
<td>3</td>
<td>7.091</td>
<td>6.734</td>
<td>5.49%</td>
</tr>
<tr>
<td>4</td>
<td>7.091</td>
<td>6.720</td>
<td>5.77%</td>
</tr>
<tr>
<td>5</td>
<td>7.091</td>
<td>6.833</td>
<td>4.40%</td>
</tr>
<tr>
<td>6</td>
<td>7.091</td>
<td>6.868</td>
<td>3.86%</td>
</tr>
<tr>
<td>7</td>
<td>7.091</td>
<td>6.860</td>
<td>3.88%</td>
</tr>
<tr>
<td>8</td>
<td>7.091</td>
<td>6.941</td>
<td>3.16%</td>
</tr>
</tbody>
</table>

Table 18: KAWS performance for K = 3
### K = 4

<table>
<thead>
<tr>
<th>Number of Manually Selected Widgets</th>
<th>Average Number of Desired Widgets</th>
<th>Average Amount of Work Remaining</th>
<th>Average Percent “Efficiency”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.091</td>
<td>6.745</td>
<td>5.91%</td>
</tr>
<tr>
<td>2</td>
<td>7.091</td>
<td>6.686</td>
<td>6.58%</td>
</tr>
<tr>
<td>3</td>
<td>7.091</td>
<td>6.694</td>
<td>6.40%</td>
</tr>
<tr>
<td>4</td>
<td>7.091</td>
<td>6.675</td>
<td>6.62%</td>
</tr>
<tr>
<td>5</td>
<td>7.091</td>
<td>6.788</td>
<td>5.15%</td>
</tr>
<tr>
<td>6</td>
<td>7.091</td>
<td>6.882</td>
<td>4.15%</td>
</tr>
<tr>
<td>7</td>
<td>7.091</td>
<td>6.941</td>
<td>3.52%</td>
</tr>
<tr>
<td>8</td>
<td>7.091</td>
<td>7.035</td>
<td>2.61%</td>
</tr>
</tbody>
</table>

Table 19: KAWS performance for $K = 4$

### K = 5

<table>
<thead>
<tr>
<th>Number of Manually Selected Widgets</th>
<th>Average Number of Desired Widgets</th>
<th>Average Amount of Work Remaining</th>
<th>Average Percent “Efficiency”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.091</td>
<td>6.723</td>
<td>6.50%</td>
</tr>
<tr>
<td>2</td>
<td>7.091</td>
<td>6.661</td>
<td>7.15%</td>
</tr>
<tr>
<td>3</td>
<td>7.091</td>
<td>6.675</td>
<td>6.98%</td>
</tr>
<tr>
<td>4</td>
<td>7.091</td>
<td>6.707</td>
<td>6.65%</td>
</tr>
<tr>
<td>5</td>
<td>7.091</td>
<td>6.833</td>
<td>4.98%</td>
</tr>
<tr>
<td>6</td>
<td>7.091</td>
<td>6.954</td>
<td>3.63%</td>
</tr>
<tr>
<td>7</td>
<td>7.091</td>
<td>7.100</td>
<td>2.17%</td>
</tr>
<tr>
<td>8</td>
<td>7.091</td>
<td>7.194</td>
<td>1.26%</td>
</tr>
</tbody>
</table>

Table 20: KAWS performance for $K = 5$

### K = 6

<table>
<thead>
<tr>
<th>Number of Manually Selected Widgets</th>
<th>Average Number of Desired Widgets</th>
<th>Average Amount of Work Remaining</th>
<th>Average Percent “Efficiency”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.091</td>
<td>6.604</td>
<td>9.26%</td>
</tr>
<tr>
<td>2</td>
<td>7.091</td>
<td>6.632</td>
<td>9.06%</td>
</tr>
<tr>
<td>3</td>
<td>7.091</td>
<td>6.677</td>
<td>7.44%</td>
</tr>
<tr>
<td>4</td>
<td>7.091</td>
<td>6.777</td>
<td>6.14%</td>
</tr>
<tr>
<td>5</td>
<td>7.091</td>
<td>6.949</td>
<td>4.00%</td>
</tr>
<tr>
<td>6</td>
<td>7.091</td>
<td>7.159</td>
<td>1.64%</td>
</tr>
<tr>
<td>7</td>
<td>7.091</td>
<td>7.339</td>
<td>-0.11%</td>
</tr>
<tr>
<td>8</td>
<td>7.091</td>
<td>7.479</td>
<td>-1.50%</td>
</tr>
</tbody>
</table>

Table 21: KAWS performance for $K = 6$
### Table 22: KAWS performance for $K = 7$

<table>
<thead>
<tr>
<th>Number of Manually Selected Widgets</th>
<th>Average Number of Desired Widgets</th>
<th>Average Amount of Work Remaining</th>
<th>Average Percent “Efficiency”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.091</td>
<td>6.516</td>
<td>11.13%</td>
</tr>
<tr>
<td>2</td>
<td>7.091</td>
<td>6.632</td>
<td>8.76%</td>
</tr>
<tr>
<td>3</td>
<td>7.091</td>
<td>6.702</td>
<td>7.79%</td>
</tr>
<tr>
<td>4</td>
<td>7.091</td>
<td>6.938</td>
<td>4.68%</td>
</tr>
<tr>
<td>5</td>
<td>7.091</td>
<td>7.172</td>
<td>1.75%</td>
</tr>
<tr>
<td>6</td>
<td>7.091</td>
<td>7.457</td>
<td>-1.56%</td>
</tr>
<tr>
<td>7</td>
<td>7.091</td>
<td>7.686</td>
<td>-3.78%</td>
</tr>
<tr>
<td>8</td>
<td>7.091</td>
<td>7.831</td>
<td>-5.24%</td>
</tr>
</tbody>
</table>

### Table 23: KAWS performance for $K = 8$

<table>
<thead>
<tr>
<th>Number of Manually Selected Widgets</th>
<th>Average Number of Desired Widgets</th>
<th>Average Amount of Work Remaining</th>
<th>Average Percent “Efficiency”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.091</td>
<td>6.573</td>
<td>11.05%</td>
</tr>
<tr>
<td>2</td>
<td>7.091</td>
<td>6.707</td>
<td>8.77%</td>
</tr>
<tr>
<td>3</td>
<td>7.091</td>
<td>6.825</td>
<td>6.91%</td>
</tr>
<tr>
<td>4</td>
<td>7.091</td>
<td>7.153</td>
<td>2.54%</td>
</tr>
<tr>
<td>5</td>
<td>7.091</td>
<td>7.387</td>
<td>-0.53%</td>
</tr>
<tr>
<td>6</td>
<td>7.091</td>
<td>7.785</td>
<td>-5.13%</td>
</tr>
<tr>
<td>7</td>
<td>7.091</td>
<td>8.046</td>
<td>-7.74%</td>
</tr>
<tr>
<td>8</td>
<td>7.091</td>
<td>8.245</td>
<td>-9.71%</td>
</tr>
</tbody>
</table>

### Table 24: KAWS performance for $K = 9$

<table>
<thead>
<tr>
<th>Number of Manually Selected Widgets</th>
<th>Average Number of Desired Widgets</th>
<th>Average Amount of Work Remaining</th>
<th>Average Percent “Efficiency”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.091</td>
<td>6.755</td>
<td>9.62%</td>
</tr>
<tr>
<td>2</td>
<td>7.091</td>
<td>6.871</td>
<td>7.39%</td>
</tr>
<tr>
<td>3</td>
<td>7.091</td>
<td>7.003</td>
<td>5.28%</td>
</tr>
<tr>
<td>4</td>
<td>7.091</td>
<td>7.457</td>
<td>-0.90%</td>
</tr>
<tr>
<td>5</td>
<td>7.091</td>
<td>7.734</td>
<td>-4.41%</td>
</tr>
<tr>
<td>6</td>
<td>7.091</td>
<td>8.183</td>
<td>-9.64%</td>
</tr>
<tr>
<td>7</td>
<td>7.091</td>
<td>8.524</td>
<td>-13.18%</td>
</tr>
<tr>
<td>8</td>
<td>7.091</td>
<td>8.750</td>
<td>-15.42%</td>
</tr>
</tbody>
</table>
### K = 10

<table>
<thead>
<tr>
<th>Number of Manually Selected Widgets</th>
<th>Average Number of Desired Widgets</th>
<th>Average Amount of Work Remaining</th>
<th>Average Percent “Efficiency”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.091</td>
<td>6.941</td>
<td>7.90%</td>
</tr>
<tr>
<td>2</td>
<td>7.091</td>
<td>7.075</td>
<td>5.43%</td>
</tr>
<tr>
<td>3</td>
<td>7.091</td>
<td>7.312</td>
<td>1.77%</td>
</tr>
<tr>
<td>4</td>
<td>7.091</td>
<td>7.860</td>
<td>-5.84%</td>
</tr>
<tr>
<td>5</td>
<td>7.091</td>
<td>8.169</td>
<td>-9.87%</td>
</tr>
<tr>
<td>6</td>
<td>7.091</td>
<td>8.651</td>
<td>-15.37%</td>
</tr>
<tr>
<td>7</td>
<td>7.091</td>
<td>9.075</td>
<td>-19.82%</td>
</tr>
<tr>
<td>8</td>
<td>7.091</td>
<td>9.350</td>
<td>-22.52%</td>
</tr>
</tbody>
</table>

Table 25: KAWS performance for $K = 10$
WORKS CITED

[1] C. Albanesius. iPhone, Blackberry, Razr, Most Popular Phones in ’09. PC Magazine online, http://www.pcmag.com/article2/0,2817,2357426,00.asp


