Generation of Infotips from Interface Labels

Eric White[1(✉)], Sandrine Fischer[2], and Foaad Khosmood[2]

[1] Physics Department, California Polytechnic State University (USA), San Luis Obispo, USA
ewhitell@calpoly.edu
[2] Computer Science Department, California Polytechnic State University (USA), San Luis Obispo, USA

Abstract. A method is presented for generating informative and natural-sounding infotips for graphical elements of a user interface. A domain-specific corpus is prepared using natural language processing techniques, and a term-frequency/inverse-document-frequency transform is used for vectorization of features. A *k*-means algorithm is then used to cluster the corpus by semantic similarity and retrieve the most similar infotips for any inputted interface label. We demonstrate the feasibility of this method and conclude by proposing several approaches to improve the selection of infotips by incorporating natural language processing and machine learning techniques.

Keywords: Natural language processing · Machine learning · Instructional design · Graphical user interfaces

1 Introduction

1.1 Context

Tooltips and infotips, collectively referred to as tips, are small pop-up windows that either label or explain, respectively, the function of elements found in a graphical user interface (GUI). They appear when a user hovers a cursor over GUI elements such as menu items, icons, and toolbar controls. This interaction principle, in which text does not need to be permanently displayed, constitutes a powerful way to un-clutter interfaces. Options and guidelines exist for customizing and optimizing tip appearance and behavior, notably their shape, position, discoverability, and duration. Several jQuery and Javascript web-based frameworks support developers in customizing the appearance and behavior of tips consistently throughout their application (e.g., Toolipster 3.0, qTip2, Tipso, Hovercard, Powertip, Darktooltip, see also WikiUp).

While the appearance and behavior of a tip are rendered consistently across the features in an interface, the textual content displayed in each tip must be written on a case-by-case basis. This requires further distinction between *tooltips*, which serve to simply label a feature (e.g., "share folder"), and *infotips*, which expand upon its role, meaning, or function (e.g., "Share the content of this folder with members of your network").

Infotips provide users with an on-demand benefit of learning more about features that are new or unfamiliar. Similarly, infotips can serve to extend the descriptiveness of

S. Yamamoto (Ed.): HIMI 2015, Part I, LNCS 9172, pp. 226–234, 2015.
DOI: 10.1007/978-3-319-20612-7_22

any interface so as to accommodate less knowledgeable users. Guidelines from Microsoft Developer Network [1] state that infotips should be informative (e.g., they supplement a given feature label with additional details) and supplemental (e.g., goes actually beyond common sense or self-descriptiveness). Judgment is required to determine if each feature is unknown, advanced, or innovative, and is thus in need of an infotip formulation that can be of service to users.

This reasoning is substantiated by instructional design studies concerning the learning benefits of added explanations. Such studies show that while less knowledgeable people do benefit from additional explanatory material, more knowledgeable people process the material better without the additions [2] (see also [3, 4]). This latter outcome, known as expertise reversal effect, comes from the fact that added information increases the cognitive load of knowledgeable people without being of any use to them (for a review, see [5]). These considerations transpose well to the addition of infotips to a GUI, in that a) infotips should target users in the need of a little more knowledge about the current features to progress in their activity, and b) since they appear only on demand, and not automatically, infotips minimize the risk of expertise reversal effect.

This suggests that designers should devote some effort to writing infotips. For interfaces having dozens (or even hundreds) of features, though, the task of creating meaningful infotips can be laborious. Indeed, none of the previously listed web-based frameworks address the generation of tip content. We explore an automatic generation of tip content using machine learning techniques and natural language processing. In this approach, labels from an interface are inputted, and descriptions are predicted from a domain-specific corpus. In addition, familiarity criteria and synonymy attributes are leveraged to determine the relevance of the infotip and expand its descriptiveness, respectively.

2 Methods

We gathered representative tip descriptions into a corpus. This data was collected from, an online database [6] of short, informative descriptions of keyboard shortcuts for various software applications. Nearly ten thousand shortcut descriptions are published under their parent menu, application name, and operating system (Mac, Windows, Linux, or web). Using the web crawler Scrapy [7], data for each shortcut was extracted and randomly ordered line-by-line into a comma-separated corpus. The data was preprocessed using standard NLP techniques (Sect. 2.1). A transformation related to term frequency was applied to convert each shortcut document (as a bag of words) into an n-dimensional vector (Sect. 2.2). Then, a k-means clustering algorithm was performed to find natural groupings in the data. Parameters were tuned to optimize performance, such as the cluster number, documents per cluster, and run number (Sect. 2.3).

2.1 Corpus Construction

Our corpus was constructed by converting rows of comma-separated shortcut descriptions and parent menus into “documents” with features. Features are informative

yet non-redundant values –in this case word stems– that serve to potentiate our ability to cluster input labels by their linguistic regularity. To create our features, we selected words from both shortcut description and menu membership (*i.e.,* parent menu fields).

After stripping away special characters and numbers from the text, the Natural Language Toolkit (NLTK) was used to tokenize the words and remove English stop words [8], or frequently occurring function words that are not contextually relevant, such as *the*, *is*, *a*, *that*, etc. The remaining words are then stemmed, which is the process of normalizing all inflected or derived words to their stem. Stemming, and other normalization techniques such as lowercasing the words, helps the clustering algorithm capture semantic relationships between words by transforming seemingly different tokens, such as *editing*, *Edit*, and *editor,* into a single feature (*edit*).

Although not used by the clustering algorithm, additional pre-processing of the text was applied for post-analysis. Each document was tagged for its part of speech (POS) using the maximum-entropy-based Stanford POS tagger [9]. As most shortcut descriptions begin with a verb, which is not a common way to begin written sentences in English, the Stanford POS tagger outperforms NLTK's default Treebank POS tagger [10] in correctly identifying leading verbs. In addition, each word of the shortcut description was lemmatized. This is similar to stemming, except the word context and POS is taken into.

We leveraged the word labels associated with the category of each label by pre-pending the token *tip-* or *menu-* to each word stem, which identifies whether the stem belongs to the shortcut description (*tip-*) or to the parent menu (*menu-*). For example, the tip *Copy a selection of text*, located in the parent menu *Edit*, would result in the following bag of words for the features: *copy select text tip-copy tip-select tip-text edit menu-edit*.

2.2 Vectorization and Clustering

Vectorization of the documents and clustering was performed with scikit-learn [11], which is a Python module designed for machine learning. Features are prepared by transforming them based on term frequency–inverse document frequency (tf–idf), which is a numerical statistic reflecting the relative importance of a word to a corpus [12]. The tf–idf transformation is often used in information retrieval and text mining for weighting words based on their frequency in a document and inversely weighting the number of documents across the collection in which they appear. A collection of features in all documents was first converted into a matrix of token counts. For each token in the count matrix, a composite weight was then created to reduce the impact of tokens that occur very frequently across all the documents. Tokens appearing in less than two documents were not included in the vectorization process. The tf-idf transformation thus serves to transform each document into an n-dimensional vector having one component for each word in the corpus.

Documents were clustered into similar categories by subjecting the above vector space to a k-means algorithm [13]. With this algorithm, N observations are partitioned into subsets of mutually exclusive clusters $\{c_1, c_2, \ldots, c_k\}$ so that each observation belongs to the cluster with the nearest mean. This is achieved by randomly selecting

k centroid seeds, then varying their locations until the within-cluster sum of squares is minimized. Clustering is unsupervised, meaning no label for each cluster is considered. We employ a "hard" clustering, meaning shortcuts were grouped into one and only one cluster. Distances between points were calculated using a Euclidean metric.

2.3 Optimization of Clustering Performance

A series of validations was conducted to improve clustering performance. Corpus documents were randomly assigned to training (90%) and test sets (10%), so that the test set corresponded to 1000 labels that were not used in the clustering. In machine learning, validation typically involves varying key parameters until optimal criteria are reached. In this case, we examined the optimal number of clusters, number of documents per cluster, and number of runs.

First, we examined dissimilarity among clusters, as clusters should be well separated and non-overlapping, with lower values indicating too many or too few clusters. This was accomplished with a standard measure of cluster dissimilarity called the silhouette score [14]. Let a_k be the average distance of each data point to its cluster centroid c_k, and let b_k be the average distance of each point in c_k to all other centroids. The silhouette score is then defined to be $1 - a_k/b_k$ for $a_k < b_k$, 0 for $a_k = b_k$, and $b_k / a_k - 1$ for $a_k > b_k$. By its definition, silhouette scores are always between -1 and 1, with values less than zero indicating poor clustering. The left side of Fig. 1 shows the silhouette vs. number of clusters k. The significant increase in silhouette score suggests a k value of at least 350. The right side of Fig. 1 shows the inertia vs. the cluster number k. The inertia is defined to be the sum of squares from each point to its closest centroid.

While the inertia is expected to decrease linearly with increasing k, a rapid convergence is observed toward the value $k = 350$. Combined, these results suggest that at least 350 clusters should be used for the clustering process.

Second, in order to return a consistent number of shortcut descriptions per cluster, we require that the number per cluster does not vary wildly. The left side of Fig. 2

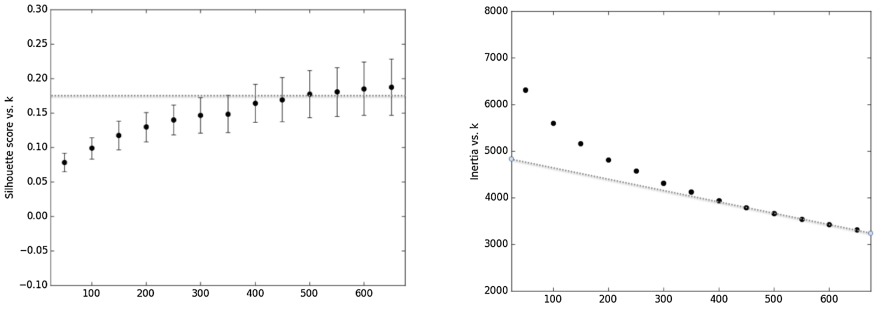

Fig. 1. Silhouette scores vs. number of clusters k (left). No significant improvement above 0.17 is observed in the silhouette score beyond 350 clusters. While inertia is expected to decrease linearly with increasing numbers of clusters (right), a more rapid decrease is observed up to a cluster number of 350. When combined, these graphs suggest that a minimum value of $k = 350$ be used in our analysis.

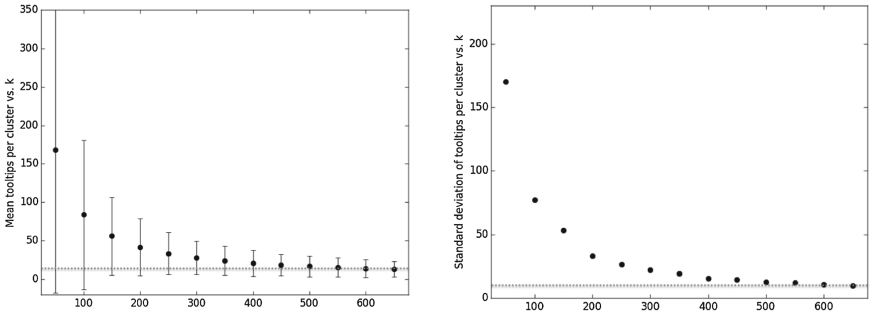

Fig. 2. Mean number of shortcuts per cluster vs. cluster number (left). In the right-hand figure, the standard deviation is plotted vs. cluster number. Stability in both distributions is achieved for values of k greater than 400.

shows the mean number of shortcuts per cluster vs. cluster number, along with standard deviation.

This data suggests that the mean number converges to a stable value for k > 400. The right side of Fig. 2 shows the size of the standard deviation vs. cluster number. No significant reduction in variation of number of shortcuts is achieved for k > 400. Thus, we select the value of k = 400 for our k-means clustering algorithm.

Third, we run the k-means algorithm multiple times with random starting centroids and keep the best result. While the k-means algorithm is one of the fastest clustering algorithms available, it is susceptible to falling into local minima. For each clustering, the algorithm is thus run five times with different centroid seeds, and the result with lowest inertia is kept. This ensures little-to-no variation in shortcut assignments when the algorithm is re-run in the future. In Fig. 3, we show the frequency distribution of shortcut descriptions by word length.

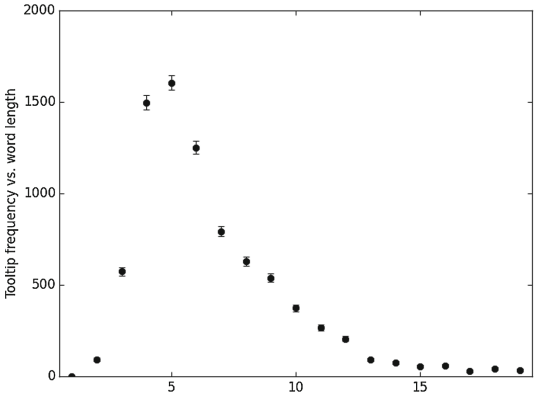

Fig. 3. Frequency of shortcuts by total number of words in their description. According to this graph, most shortcut descriptions in our corpus are five words in length, e.g., *Cancel command and close window*. Note: while stop words are counted in the above graph, non-alphanumeric characters and punctuation are not.

We found that the number of irrelevant candidates was lowered when the minimum number of shortcut descriptions in which a word must appear was set to two. In the following section, we discuss how candidate infotips are selected from the cluster that best matches a newly inputted label.

3 Results

3.1 Infotip Selection

Once the clustering has been performed, we inputted new labels from our test set and retrieved candidate shortcuts from the cluster with the highest degree of similarity. New labels were first preprocessed and vectorized in an identical manner as the shortcuts in the training set used by the *k*-means clustering algorithm. Once the best cluster had been found, the associated shortcut descriptions were extracted from this cluster. Stop words were removed and remaining words from the new label were compared against the candidate shortcuts. We calculated the percentage of normalized words from the new label that matched a word in the list of candidates. For example, consider the new label *Copy the text to clipboard*, which becomes *copy text clipboard* after normalization. If two of these words matched at least one word in the collection of candidate shortcut descriptions from the best cluster, the percentage for this new label would be 66.7%.

As we seek tips that are descriptive, we also considered candidate shortcut descriptions based on their word length. Synonyms for each word in the list of candidate were are also employed to improve the relevance of candidate shortcuts. This was accomplished with WordNet [15], a large lexical database of English nouns, verbs, adjectives, and adverbs that are grouped by their semantic relationship into *synsets*. We extracted word synonyms, hypernyms, hyponyms, and "similar" words from these synsets in order to both expand and refine the selection of shortcut candidates. If no word from the new label exactly matched a candidate shortcut description, the word was sought within a bag of "similar" words derived from the unmatched candidate. While no significant improvement was observed for "similar" words, the fraction of matches increased by about 10% when bags of synonym, hypernyms, and hyponyms were included. In Table 1, we list the percentages of new labels from the test set that were matched vs. the minimum word length for each shortcut candidate.

In the next section, we provide examples of candidate tips by our process for newly inputted labels chosen at random from our test set.

3.2 Examples

As GUI labels tend to be short in length, we skipped any labels from the test set greater than three words in length, after stop words had been removed. This served to remove shortcut descriptions that were wordier than a standard interface tooltip. We required that candidate infotips be at least three words long after stop-word removal. We thus differentiate between the length inputted labels and the resulting candidates of shortcut descriptions (infotips). This selection criterion guarantees that

Table 1. Fraction of words from new labels in test set that was matched to least one word from the candidate shortcut description in the best cluster. Minimum length is the minimum number of words required in the candidate shortcuts. Synonym, hypernyms, and hyponyms resulted in an increase of approximately 10%.

Min. length	Frac. of words matched in new label	Frac. with "similar" words	Frac. with synsets, hypo-/ hypernyms
1	0.62	0.63	0.74
2	0.60	0.60	0.71
3	0.52	0.53	0.63
4	0.45	0.45	0.55
5	0.36	0.37	0.46
6	0.29	0.29	0.38
7	0.22	0.23	0.29

the lengths of the infotip candidates are equal to or greater than those of the inputted labels. In this way, a label such as *Print all documents* would never return an uninformative infotip candidate such as *Print*. In Table 2, we show the result of several candidate infotips selected by our method, along with the corresponding label and parent menu inputted into our method.

Table 2. Example infotips generated from labels taken from the test set. Acceptable outcomes are listed for each label; those deemed unconvincing are listed in *italics*.

Menu / label	Infotip
Session / switch open window	- *Open new session*
	- *Open current session in PuTTY*
	- Cycle opened sessions
Image / file import	- Import multiple files or image sequences
	- Import one file or image sequence
Find / next in webpage	- Find next occurrence of search term
	- Find next find previous match
	- Find previous find next occurrence of search term
	- Find text on webpage
Compose message / paste without formatting	- *Paste as quotation*
Edit / deselect tracks	- Deselect all files
	- Deselect all the songs in the list
	- Select deselect file
Ribbon / expand ribbon	- Minimize or restore the ribbons
	- Show/hide the ribbon
	- Expand or minimize the ribbon
General / switch to debug mode	- Switch to Projects mode
	- Switch to Help mode
	- Switch to Edit mode

As Table 2 illustrates, several of the generated infotips match the inputted label rather well, such as *Cycle opened sessions* for the input label *switch open window*, and *Deselect all the songs in the list* for *deselect tracks*. As the sampling of infotips displayed in Table 2 was *ad hoc*, a more systematized rating scheme should be employed to determine the validity of our method in a more statistical way.

During the vectorization process, words occurring frequently throughout the training set receive less weight. This helps to offset the heavy weighting that ubiquitous words would receive otherwise. In our case, though, bias introduced by the tf-idf transform may not work as intended, as the impact of common interface labels, such as *paste and copy*, may be attenuated. However, common labels such as these correspond to basic functions that are familiar to most users. From the user-centered viewpoint advanced in introduction, this means that need for generating infotips is least justified for these labels. Conversely, the need for helpful infotips is greatest for interface labels that are less common or familiar, or too technical. Thus, the weighting scheme employed during vectorization may need to be tuned to match some predetermined threshold of familiarity established by corpus-based NLP techniques.

4 Perspectives

We envision several avenues along which the selection of infotips may be improved. For one, we could consider alternative vectorization transforms, such as a vector representation for words (VRW) model, which is an efficient implementation of the continuous bag-of-words and skip-gram architectures [16]. In the VRW model, a neural network is trained to generate a continuous vector representation of the words. While VRW models are known to suffer from their indifference to idiomatic phrases, they are particularly precise at capturing a large number of syntactic and semantic relationships among words. In the most general case, tf-idf and VRW transforms could be compared as components of a mixed model.

An altogether different clustering algorithm could be implemented, such as a Latent Semantic Indexing (LSI) model, which implements singular value decomposition in order to identify semantic patterns between terms and concepts in some reference corpus [17]. The LSI model is based on the idea that words used in the same context tend to have similar semantic relationships. This allows LSI models to accurately extract conceptual content of a text by establishing associations between terms that occur in similar contexts. Another refinement of our approach involves the expansion of the corpus used for the training process. For example, data dumps from technically relevant websites could be included, as well as help files and user documentation for various software applications. In addition, the method presented has potential for online learning, e.g., infotips may be hand-corrected by users and accepted back into the corpus for re-training. Finally, we are exploring several NLP techniques for improving infotip selection based on conditional word frequencies, named entity recognition, and grammatical features.

References

1. Microsoft Developer Network on Tooltips and Infotips. Accessed on January, 2015. http://msdn.microsoft.com/en-us/library/windows/desktop/dn742443%28v=vs.85%29.aspx
2. Yeung, A., Jin, P., Sweller, J.: Cognitive load and learner expertise: split-attention and redundancy effects in reading with explanatory notes. Contemp. Educ. Psychol. **23**, 1–21 (1998)
3. McNamara, D.S., Kintsch, W.: Are good texts always better? Interactions of text coherence, background knowledge and levels of understanding in learning from texts. Cogn. Instr. **14**, 1–43 (1996)
4. Mayer, R.E.: Research-based principles for the design of instructional messages. The case of multimedia explanations. Doc. Des. **1**, 7–20 (1999)
5. Sweller, J., Ayres, P.L., Kalyuga, S., Chandler, P.: The expertise reversal effect. Educ. Psychol. **28**(1), 23–31 (2003)
6. ShortcutWorld.com, a wiki-style reference database for keyboard shortcuts. http://www.shortcutworld.com
7. Hoffman, P.: Scrapy (2013). http://scrapy.org
8. Bird, S., Edwardd, L., Ewan, K.: Natural Language Processing with Python. O'Reilly Media Inc. (2009)
9. Toutanova, K., Klein, D., Manning, C., Singer, Y.: Feature-rich part-of-speech tagging with a cyclic dependency network. In: Proceedings of HLT-NAACL 2003, pp. 252–259 (2003)
10. Marcus, M.P., Santorini, B., Marcinkiewicz, M.A.: Building a large annotated corpus of english: the penn treebank. Comput. Linguist. **19**(2), 313–330 (1993)
11. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, p. 2825–2830 (2011)
12. Spärck Jones, K.: A statistical interpretation of term specificity and its application in retrieval. J. Documentation **28**, 11–21 (1972)
13. Lloyd, S.P.: Least squares quantization in PCM. IEEE Trans. Inf. Theor. **28**(2), 129–137 (1982)
14. Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Comput. Appl. Math. **20**, 53–65 (1987)
15. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database. MIT Press, Cambridge, MA (1998)
16. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. In: Proceedings of Workshop at the International Conference on Learning Representations 2013 (2013)
17. Rehurik, R.: Scalability of Semantic Analysis in Natural Language Processing. Ph.D. thesis, Masaryk University, Brno, Czech Republic (2011)