An AI for the Wikipedia Game

Alex Barron
admb@stanford.edu

Zack Swafford
zswaff@stanford.edu

1 Introduction

1.1 The game

We designed an AI to play The Wikipedia Game, a computer game played on wikipedia.org. In short, players start on a random Wikipedia article and then try to traverse the site using only the hyperlinks in the articles to get to some common destination article. The game can be played by humans either for speed or for fewest links used, but a key component of a human's strategy is that he never backtracks (i.e. his algorithm is entirely greedy).

An input to our AI will therefore be a random Wikipedia article, and the output will be a path from there to the destination article, which we chose to remain fixed at the 'Stanford University' page. Our AI will navigate from the start article to the goal one attempting to minimize both the link-distance and time but also attempting to minimize states viewed, to emulate the human player and because in a real application to the web each article would take non-trivial time to download and view.

1.2 Classes

This project incorporates a variety of concepts in artificial intelligence, game theory, and machine learning. Because all of the elements are used in concert and interrelated, the distinction between the subjects is difficult to make. In general, the search problem and modelling apply more specifically to the subject matter in CS 221, and the rest of the machine learning (i.e. the various ML algorithms, K-Means, and PCA) are geared toward CS 229.

1.3 Related work

While we were unable to find any academic articles directly investigating the link traversal which forms the bulk of our project, many groups have applied machine learning techniques to explore the structure and semantics of Wikipedia. Much of previous research focuses on text and document categorization and to that end, various techniques have been implemented to group Wikipedia articles.

Banerjee, Somnath, et al [1] along with a number of other groups have attempted to create "concepts" from the articles to provide more descriptive features for them. The most successful approaches such as that of Milne, David, et al [2] and Yazdani, Majid, et al [3], heavily utilized the links in each article for classification, which boded well for our link-based approach to explore the structure of Wikipedia. Hu, Jian, et al [4] introduced some representation of the hierarchical nature Wikipedia which improved their text classification, in particular differentiating category and link based articles from the rest. Our technique employs unsupervised learning to infer such relationships and build features representing these broader articles to augment our base bag of words model. Intuitively such higher level articles should be useful in linking diverse concepts and thus providing minimal path solutions in the game.

Many previous papers have investigated various natural language processing methods to optimally featurize and then cluster wikipedia pages in a similar manner to our unsupervised learning. Hu, Xia, et al [5] proposed the use of the Porter Stemmer to most effectively stem features, a technique we inherited, but also suggested the removal of date strings as features. For our specific application, date articles turned out to be very common on minimal path solutions and so we decided to retain them in our feature set.

2 Data and features

2.1 Dataset

The first thing we did to collect data was extensive human trials. We observed enough plays through the game to establish an oracle for human ability with confidence. A comparison of these results with results from various implementations of our AI can be found in Table 2 on page 4.

Our data was all downloaded directly from the Simple English version of Wikipedia, which can be found at simple.wikipedia.org. This website, and all Wikimedia websites, provides a free .xml dump of the entire site. We downloaded this single file and parsed it into a series of pages, each with its article text attached. We then extracted all of the hyperlink information from the pages and used this to form a directed graph, with each page represented by a node and each link from one page to another as an edge in that direction.

With this exceedingly simple implementation we were

able to implement basic graph search algorithms such as Uniform Cost Search (UCS). However, the more interesting problem began when we introduced a model to estimate the link-distance between two articles. To do this, we first had to generate training sets to train the models and testing sets to test them. We did this by running the simple (but slow and costly) exhaustive UCS algorithm to determine the actual minimum link distance d between a random given start page p_{start} and our p_{goal}, 'Stanford University'. This allowed us to generate a virtually infinite amount of training and testing data in pairs (p_{start}, d) representing the input and the desired output of our predictor. Essentially, this created an arbitrary dataset $\mathcal{S} \equiv \{(x^{(i)}, y^{(i)})\}$ for $i = 1, \ldots, m$ where $x^{(i)}$ is a random article and $y^{(i)}$ is the distance between that article and the goal.

2.2 Feature extraction

To extract meaningful data about each webpage for the ML algorithms, we had to featurize each article. We used a feature extractor $\phi(\cdot)$ which could operate on any article x. The page was modelled as a bag of words (i.e. we assumed that word order was irrelevant to our problem). We therefore implemented the feature extractor using the nltk package to tokenize and stem each word in the page, so that any pages with similar concepts might have features in common. We ignored commonly used words by removing any of nltk's stop words from the feature set. Furthermore, we added features for the number of words and the number of links in a page. Therefore $\phi(x^{(i)})$ essentially created a sparse feature vector with only the words in the page and a modicum of metadata about it.

3 Supervised Learning

3.1 Search

Initially we implemented a state-based model to frame the problem. Essentially, the goal of the algorithm is to find a path from the start article to the goal article via a series of other articles. The state-based model is simple: each state is based on a article, and must store all of the information that the article does. This is the only value intrinsic to a state, so it is the sole component of a state. The actions which can be taken from a given state are to transfer to a state representing any article that the given state links to. Following this action transfers to the given state, and the cost of taking this transition is uniform at 1 because each transition counts as one link used. Finally, a state is the goal state if the article it refers to is the goal article.

3.2 Machine Learning

This model was sufficient to use simple state-based search algorithms–namely, Depth-First Search (DFS), Breadth-First Search (BFS), and UCS. These algorithms (particularly UCS, unsurprisingly) performed well and found the minimal path within a reasonable amount of time. However, we were interested in trading the guarantee of the minimal path that UCS affords to decrease the number of states that the algorithm has to explore. The algorithms fell short of our goal because although they all found the minimum path between p_{start} and p_{goal}, in each case the number of states the algorithm had to explore was still very high. To remediate this, we applied a heuristic to UCS (thereby making it A*) which could estimate the distance from one article to another. This strategy was roughly based on ours as humans–when we play the Wikipedia Game, we follow a greedy algorithm, clicking on one page and never looking back. This means that we almost never find the shortest path (only an approximate one, on the order of the minimal path), but it also means that we are much more efficient and don't have to explore as many states. All of this is only possible because humans are able to incorporate knowledge of the content of the articles in their decisions about which link to follow.

To implement this heuristic, we made the assumption that the link-distance between two articles could be modelled directly and implemented ML algorithms to find it. As discussed above, we used a series of training examples with the input of a page $x^{(i)}$, a bag of words feature extractor $\phi(\cdot)$, and the actual minimum distance from that page to the goal $y^{(i)}$ to train different algorithms.

It is important to note that the output of our algorithms y is not binary; this is a multiclass application of ML. In principle, y can assume any value $1, \ldots, l$ (where l is the maximum minimum path length from any article to the goal) or the value ∞, representing the case where no path exists whatsoever. In practice, the range of values y could assume were limited to the range of the values assumed by any of training examples $y^{(i)}$. Usually, this range was about $[1:8], \infty$.

One common strategy to handle mutlticlass classification is One vs. Rest (OVR) classification. This involves training any ML algorithm as usual, but doing so once for each of the different possible outputs. For each separate model, only the training examples pertaining to that particular outcome are marked as positive outcomes. Each model is then able to generate a score for its outcome on a new example x. Whichever of the models gives the highest score to that example (implying that that outcome is, in some sense, the most likely result according to the model) predicts the overall result to be the corresponding outcome. We used the sklearn library to implement OVR Logistic Regression; Stochastic Gradient Descent (SGD) with Hinge

Loss and with Perceptron Loss; and a Support Vector Machine (SVM) with Hinge Squared Loss. None of these was the most effective algorithm, so we will not describe them here. See Table 1 on page 4 for details.

Given a random set of 1000 training examples and random test set of 200 examples, Multinomial Logistic Regression was able to consistently outperform all of the above methods. Multinomial Logistic Regression actually operates in a similar fashion to OVR, but the output of each of the classifiers is a probability that that classifier's outcome is the correct one. Normalizing over the results produces a probability distribution for the outcome, and selecting the one with the highest probability allows the algorithm to select a particular value. Each of the individual classifiers in the model is implemented as a linear combination of the weights of the example x (note that $x_0 \equiv 1$ for the intercept term). The classifier's score therefore varies with $w^\top \phi(x)$, where w is the weight vector learned in the training phase. Specifically, the classifier's score is

$$h_w(x) \equiv S(x) = \frac{1}{1 - e^{-w^\top x}}.$$

Typically, the parameter w is taken to be the maximum a posteriori (MAP) estimate

$$\begin{aligned} w &\equiv \operatorname*{argmax}_w L(w) \\ &= \operatorname*{argmax}_w p(\vec{y} \mid X : w) \\ &= \operatorname*{argmax}_w \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)} : w), \end{aligned}$$

which can be found by stochastic gradient ascent via the update rule

$$w_j := w_j + \alpha(y^{(i)} - h_w\left(x^{(i)}\right)) x_j^{(i)}.$$

When run to convergence, this algorithm produced a relatively powerful classifier for our data.

4 Unsupervised Learning

4.1 K-Means

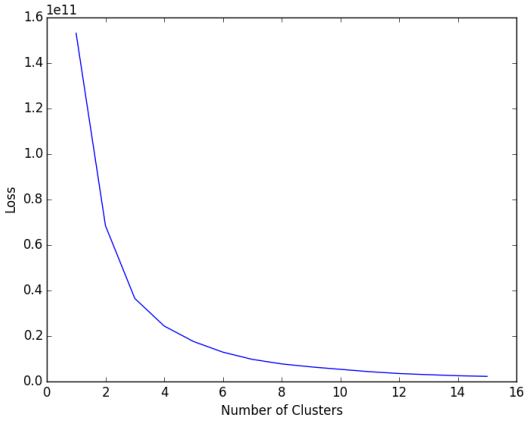
One method we used to improve our algorithm and to gain an intuition for our feature space was K-Means clustering, which we used to develop clusters of pages under our feature extractor. Our theory was inspired by human players, who frequently maintained the initial goal of getting to articles on the topic of the goal, then focused on moving from there to the real goal. The intuition was that if the machine could also take this topical context into account the algorithm would improve. Clustering the data allowed us to add another feature to the feature set–specifically, the identity of the cluster each page was in. K-Means is run in two steps: the assignment step, where each page is assigned to a centroid, mathematically represented by

$$c^{(i)} := \operatorname*{argmin}_j ||x^{(i)} - \mu_j||^2;$$

and the recentering step, where each centroid is moved to the average of its constituents places with

$$\mu_j := \frac{\sum_{i=1}^{m} 1\{c^{(i)} = j\} c^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)} = j\}}.$$

Figure 1: K-Means Loss vs. Num Clusters

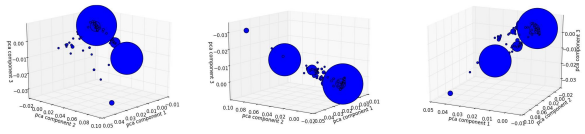


We clustered the data for various values of K and graphed the resulting total loss from all of the pages. The graph is displayed in Figure 1. Estimating the inflection point (whose K value is the optimal number of clusters) based on the shape of the graph, we found that optimal clustering was achieved by K=4.

4.2 Principal Component Analysis

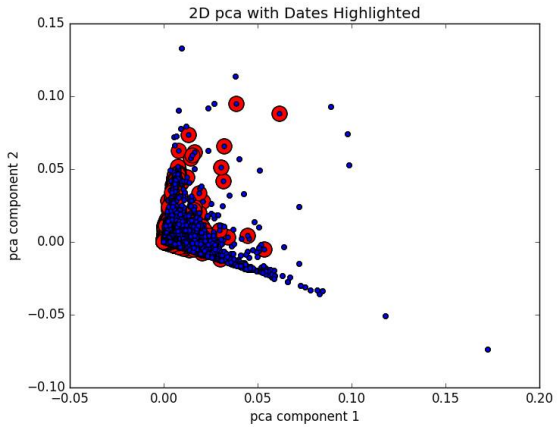
The unsupervised learning algorithm Principal Component Analysis (PCA) was also quite helpful in understanding our dataset and feature space. The PCA algorithm runs a Singular Value Decomposition (SVD) on the matrix X of all the training examples. This creates a list of eigenvectors of the space, ordered by their importance to the space. Creating a transformation matrix C as a slice of the first n eigenvectors and multiplying $C^\top X$ flattens X to n-space, assisting in visualization.

Figure 2: Different Perspectives of 3D PCA with More Important Nodes Emphasized



For instance, PCA showed us that date articles (e.g. "2007" or "12 January"), which intuitively connect sporadic ideas, proved to be a very common route to the goal for the minimum path. It can be seen in Figure 2 that certain nodes are disproportionately more valuable than others (represented by the large blue circles); these are primarily date and list nodes. Upon further investigation, we found that these types of articles are outliers in the 2D space (see Figure 3).

Figure 3: 2D PCA with Dates Highlighted



The fact that they are outliers suggests that something is different about the structure of these pages. Specifically, list and date articles have a lot of links to other pages and shockingly little text. These two types of articles, along with some of Wikipedia's administrative articles, comprised the vast majority of the outliers in the 2D PCA space.

5 Results

We found with high confidence that from 35% of randomly generated start articles, 'Stanford University' could not be reached at all. For 65% of start articles, there did exist a path from p_{start} to p_{goal}; on average the empirically minimal path had a cost (i.e. length) of 4.7 transitions.

The really interesting results had to do with the number of states explored under various state search algorithms, which can be seen in Table 2. BFS and DFS both by definition explored all 210,000 states per solution. The time complexity of UCS depends inherently on the structure and convexity of the search problem. Without an heuristic for the future cost of a path (i.e. the cost from p_{curr} to p_{goal}), the transition costs for the problem are uniform. This structure allowed UCS to achieve a much better result than DFS or BFS; when UCS was used to solve the problem it found a minimal path after exploring an average of 19,000 states for a speedup of more than 1,000%. All of these algorithms are guaranteed to find a minimal cost path when one exists.

To improve UCS, we undertook to implement an ML algorithm to predict the future cost of a path. Specifically, we tried the methods in Table 1.

Table 1: Comparison of ML Algorithms

Model	Accuracy	Distance
Logistic Regression	55%	.54
SGD with Hinge Loss	21%	1.2
SGD with Perceptron Loss	5.8%	1.8
SVM with Hinge Squared Loss	44%	.64
Multinomial Logistic Regression	59%	.50

Note that all but the ultimate algorithm is OVR.

The 'Distance' column in the table represents the average distance between the predicted value of y and the actual one. It is clear to see that Multinomial Logistic Regression yielded both the best accuracy and the lowest distance, so we implemented that for the algorithm.

Finally, consider the performance of A*, implemented with Multinomial Logistic Regression as an heuristic. It did not always find a minimum cost path, because the heuristic we used was not consistent: on average less than 40% of the time the heuristic would be incorrect (by a distance of 1.2, on average) and sometimes this error would result in an overestimate of the actual distance. Overestimating distance contradicts the definition of consistency for heuristics, which meant that A* achieved an average path length of 5.2, 11% more than the minimum. Furthermore, it took A* more than 25x as long to accomplish this result because it had to use the predictor on each of the states it visited.

A* did, however, drastically improve in our desiderata, the number of states explored. The heuristic allowed the algorithm to advance to p_{goal} while only examining an average of 510 states, which is a massive improvement even over UCS. Specifically, it is a twentyfold improvement over UCS and a two-hundredfold improvement over the naive algorithms. This result was exactly as we had expected and imagined.

Table 2: Comparison of Search Algorithms

Search Type	Path Length	States Explored	Time
Human	7.8	7.8	190 s
DFS	4.7	210,000	5.2 s
BFS	4.7	210,000	4.6 s
UCS	4.7	19,000	.45 s
A*	5.2	510	12 s

6 Discussion

Comparing these results to our extensive human trials, we see that A* can still do the problem 15x faster than humans can. Furthermore, A*'s result is within throwing distance of the minimal one; humans are nearly twice that. However, the massive advantage that humans obtain in the game from their ability to actually understand the articles and the context allows them to perform a greedy search and explore exactly as many states as their selected path. Humans still explore only 1.5% the number of states that A* does (and .037% the number UCS does). In a context without the whole of Wikipedia downloaded (i.e. with the introduction of network latency, a fact of life on the internet) humans could still conceivably beat A* at the game.

Nonetheless, the strides which A* exhibits demonstrate massive improvement over any non-heuristic based algorithm, which shows that our implementation really has learned something about the goal article. One of the most impressive aspects of this learning is that it also demonstrates serious latent structure in the data (structure which it learned). The classification is incorrect by an average of only 1.2 for the incorrectly classified examples (i.e. the output is only off by two or more about 10% of the time overall). This would not be possible if adjacent categories were not inherently very similar.

7 Future Work

Some avenues for continued improvement stand out. Firstly, our Multinomial Logistic Regression classifier could be improved. Our solution is almost certainly overfit to the data (i.e. we have a high variance) because the bag of words model usually errs in that direction. That model has so many features, and the training set is even in our context such a small portion of the space, that the classifier is extremely likely to overfit to certain words. Further investigation is required, but a simple path to improvement for the accuracy of the classifier might be to remove rare words from the feature space or to train on more of the data.

In this vein, implementing other or more complicated ML algorithms might also yield better results. Because the training set is virtually infinite and the output y is multiclass, this might very well be a good space for a neural network to be applied. If the neural net could even better classify and predict the distance, we could move even further toward a perfect heuristic for A*.

Once we settle on an optimal ML algorithm, it would be interesting to vary the weighting of the heuristic in A*. Although this will likely wreak havoc with the consistency of the estimate, it will produce interesting results. A* will likely be much less minimal in its pathfinding, but could potentially also explore fewer states. Examining this tradeoff between minimality and states explored is the theme of our problem. If the heuristic is good enough (especially among low-distance states) it is entirely possible that an implementation of weighted-DFS using the heuristic to calculate edge weights might produce similar results to the ones achieved by a human's greedy search.

8 References

[1] Banerjee, Somnath, Krishnan Ramanathan, and Ajay Gupta. "Clustering short texts using wikipedia." Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 2007.

[2] Milne, David, and Ian H. Witten. "Learning to link with wikipedia." Proceedings of the 17th ACM conference on Information and knowledge management. ACM, 2008.

[3] Yazdani, Majid, and Andrei Popescu-Belis. "Using a Wikipedia-based semantic relatedness measure for document clustering." Proceedings of Graph-based Methods for Natural Language Processing. Association for Computational Linguistics, 2011.

[4] Hu, Jian, et al. "Enhancing text clustering by leveraging Wikipedia semantics." Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 2008.

[5] Hu, Xia, et al. "Exploiting internal and external semantics for the clustering of short texts using world knowledge." Proceedings of the 18th ACM conference on Information and knowledge management. ACM, 2009.

[6] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[7] Tiago P. Peixoto, "The graph-tool python library", figshare. (2014) DOI: 10.6084/m9.figshare.1164194